

Für alle
Programm-
versionen von
2007 bis
2016

michael KOFLER
ralf NEBELO



Excel 2016

PROGRAMMIEREN

ABLÄUFE AUTOMATISIEREN,
(OFFICE-)ADD-INS UND
ANWENDUNGEN ENTWICKELN

HANSER



Im Internet: Zusatzmaterial zum Buch

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



Michael Kofler
Ralf Nebelo

Excel 2016 programmieren

Abläufe automatisieren, (Office-)Add-ins
und Anwendungen entwickeln

HANSER

Die Autoren:

Michael Kofler, Graz

Ralf Nebelo, Bocholt

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> abrufbar.

Dieses Werk ist urheberrechtlich geschützt.

Alle Rechte, auch die der Übersetzung, des Nachdruckes und der Vervielfältigung des Buches, oder Teilen daraus, 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: Brigitte Bauer-Schiewek

Copy editing: Petra Kienle, Fürstenfeldbruck

Herstellung: Irene Weilhart

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

Umschlagrealisation: Stephan Rönigk

Gesamtherstellung: Kösel, Krugzell

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

Printed in Germany

Print-ISBN: 978-3-446-44798-1

E-Book-ISBN: 978-3-446-45008-0

Inhalt

Vorwort	XIV
Konzeption des Buchs	XVIII

TEIL I: Intuitiver Einstieg	1
--	----------

1 Das erste Makro	3
1.1 Begriffsdefinition	3
1.2 Was ist Visual Basic für Applikationen?	6
1.3 Beispiel: Eine Formatvorlage mit einem Symbol verbinden	7
1.4 Beispiel: Makro zur Eingabeerleichterung	13
1.5 Beispiel: Einfache Literaturdatenbank	15
1.6 Beispiel: Formular zur Berechnung der Verzinsung von Spareinlagen	21
1.7 Beispiel: Benutzerdefinierte Funktionen	26
1.8 Beispiel: Analyse komplexer Tabellen	27
1.9 Beispiel: Vokabeltrainer	28
1.10 Weitere Beispiele zum Ausprobieren	34

2 Neuerungen in Excel 2007 bis 2016	41
2.1 Die Benutzeroberfläche RibbonX	42
2.2 Neue Programmfunktionen	45
2.3 Office-Add-ins	49
2.4 Neues in Sachen Programmierung	51
2.4.1 Kompatibilitätskrücke Add-ins-Register	52
2.4.2 Zu- und Abgänge im Objektmodell	53
2.4.3 Anpassen der Benutzeroberfläche	54
2.4.4 Die Grenzen von VBA	55
2.5 Probleme und Inkompatibilitäten	56

TEIL II: Grundlagen	59
3 Entwicklungsumgebung	61
3.1 Komponenten von VBA-Programmen	61
3.2 Komponenten der Entwicklungsumgebung	62
3.3 Codeeingabe in Modulen	69
3.4 Makros ausführen	73
3.5 Makroaufzeichnung	74
3.6 Tastenkürzel	76
4 VBA-Konzepte	79
4.1 Variablen und Felder	79
4.1.1 Variablenverwaltung	79
4.1.2 Felder	84
4.1.3 Syntaxzusammenfassung	87
4.2 Prozedurale Programmierung	89
4.2.1 Prozeduren und Parameter	89
4.2.2 Gültigkeitsbereich von Variablen und Prozeduren	98
4.2.3 Verzweigungen (Abfragen)	102
4.2.4 Schleifen	105
4.2.5 Syntaxzusammenfassung	108
4.3 Objekte	111
4.3.1 Der Umgang mit Objekten, Methoden und Eigenschaften	111
4.3.2 Der Objektkatalog (Verweise)	117
4.3.3 Übersichtlicher Objektzugriff durch das Schlüsselwort With	120
4.3.4 Objektvariablen	121
4.3.5 Syntaxzusammenfassung	123
4.4 Ereignisse	124
4.4.1 Ereignisprozeduren	125
4.4.2 Ereignisprozeduren deaktivieren	128
4.4.3 Überblick über wichtige Excel-Ereignisse	129
4.4.4 Ereignisse beliebiger Objekte empfangen	134
4.4.5 Ereignisprozeduren per Programmcode erzeugen	136
4.4.6 Syntaxzusammenfassung	138
4.5 Programmierung eigener Klassen	141
4.5.1 Eigene Methoden, Eigenschaften und Ereignisse	143
4.5.2 Collection-Objekt	146
4.5.3 Beispiel für ein Klassenmodul	147
4.5.4 Beispiel für abgeleitete Klassen (Implements)	149
4.5.5 Eine Klasse als FileSearch-Ersatz	153
4.5.6 Syntaxzusammenfassung	160
4.6 Operatoren in VBA	161
4.7 Virenschutz	164
4.7.1 Vorhandene Schutzmaßnahmen nutzen	165

4.7.2	Viren selbst entdecken	168
4.7.3	Vertrauenswürdige Makros ohne Einschränkungen ausführen	168
5	Programmiertechniken	171
5.1	Zellen und Zellbereiche	171
5.1.1	Objekte, Methoden, Eigenschaften	171
5.1.2	Anwendungsbeispiele	186
5.1.3	Syntaxzusammenfassung	195
5.2	Arbeitsmappen, Fenster und Arbeitsblätter	197
5.2.1	Objekte, Methoden und Eigenschaften	198
5.2.2	Anwendungsbeispiele	204
5.2.3	Syntaxzusammenfassung	208
5.3	Datentransfer über die Zwischenablage	210
5.3.1	Zellbereiche kopieren, ausschneiden und einfügen	210
5.3.2	Zugriff auf die Zwischenablage mit dem DataObject	212
5.3.3	Syntaxzusammenfassung	213
5.4	Umgang mit Zahlen und Zeichenketten	214
5.4.1	Numerische Funktionen, Zufallszahlen	214
5.4.2	Zeichenketten	216
5.4.3	Umwandlungsfunktionen	221
5.4.4	Syntaxzusammenfassung	223
5.5	Rechnen mit Datum und Uhrzeit	225
5.5.1	VBA-Funktionen	229
5.5.2	Tabellenfunktionen	231
5.5.3	Anwendungs- und Programmiertechniken	232
5.5.4	Feiertage	235
5.5.5	Syntaxzusammenfassung	241
5.6	Umgang mit Dateien, Textimport/-export	242
5.6.1	File System Objects - Überblick	243
5.6.2	Laufwerke, Verzeichnisse und Dateien	245
5.6.3	Textdateien (TextStream)	251
5.6.4	Binärdateien (Open)	253
5.6.5	Excel-spezifische Methoden und Eigenschaften	257
5.6.6	Textdateien importieren und exportieren	260
5.6.7	Textexport für Mathematica-Listen	268
5.6.8	Syntaxzusammenfassung	273
5.7	Benutzerdefinierte Tabellenfunktionen	277
5.7.1	Grundlagen	277
5.7.2	Beispiele	284
5.8	Schutzmechanismen	286
5.8.1	Bewegungsradius einschränken	287
5.8.2	Zellen, Tabellenblätter und Arbeitsmappen schützen	288
5.8.3	Schutzmechanismen für den gemeinsamen Zugriff	292
5.8.4	Programmcode und Symbolleiste schützen	293
5.8.5	Syntaxzusammenfassung	294

5.9	Konfigurationsdateien, individuelle Konfiguration	295
5.9.1	Optionen	295
5.9.2	Optionseinstellungen per Programmcode	296
5.9.3	Konfigurationsdateien	299
5.10	Tipps und Tricks	307
5.10.1	Geschwindigkeitsoptimierung	307
5.10.2	Zeitaufwendige Berechnungen	308
5.10.3	Effizienter Umgang mit Tabellen	312
5.10.4	Zusammenspiel mit Excel-4-Makros	314
5.10.5	Excel-Version feststellen	315
5.10.6	Hilfe zur Selbsthilfe	315
5.10.7	Syntaxzusammenfassung	317
6	Fehlersuche und Fehlerabsicherung	319
6.1	Hilfsmittel zur Fehlersuche (Debugging)	319
6.1.1	Syntaxkontrolle	319
6.1.2	Reaktion auf Fehler	320
6.1.3	Kontrollierte Programmausführung	323
6.2	Fehlertolerantes Verhalten von Programmen	325
6.3	Reaktion auf Programmunterbrechungen	330
6.4	Syntaxzusammenfassung	331
7	Dialoge	333
7.1	Vordefinierte Dialoge	333
7.1.1	Excel-Standarddialoge	333
7.1.2	Die Funktionen MsgBox und InputBox	337
7.1.3	Die Methode Application.InputBox	337
7.2	Selbst definierte Dialoge	339
7.2.1	Veränderungen gegenüber Excel 5/7	340
7.2.2	Einführungsbeispiel	342
7.3	Der Dialogeditor	346
7.4	Die MS-Forms-Steuerelemente	350
7.4.1	Beschriftungsfeld (Label)	351
7.4.2	Textfeld (TextBox)	352
7.4.3	Listenfeld (ListBox) und Kombinationslistenfeld (ComboBox)	355
7.4.4	Kontrollkästchen (CheckBox) und Optionsfelder (OptionButton)	361
7.4.5	Buttons (CommandButton) und Umschaltbuttons (ToggleButton)	362
7.4.6	Rahmenfeld (Frame)	363
7.4.7	Multiseiten (MultiPage), Register (TabStrip)	365
7.4.8	Bildlaufleiste (ScrollBar) und Drehfeld (SpinButton)	369
7.4.9	Anzeige (Image)	371
7.4.10	Formelfeld (RefEdit)	372
7.4.11	Das UserForm-Objekt	374
7.5	Steuerelemente direkt in Tabellen verwenden	377

7.6	Programmiertechniken	384
7.6.1	Zahleneingabe	384
7.6.2	Dialoge gegenseitig aufrufen	386
7.6.3	Dialoge dynamisch verändern	388
7.6.4	Umgang mit Drehfeldern	390

8 Die Benutzeroberfläche von Excel 2016 **393**

8.1	Menüs und Symbolleisten	393
8.1.1	Manuelle Bearbeitung von Menüs und Symbolleisten	395
8.1.2	Programmierte Veränderung von Menüs und Symbolleisten	401
8.1.3	Programmiertechniken	406
8.1.4	Blattwechsel über die Symbolleiste	409
8.1.5	Excel-Anwendungen in Befehlsleisten integrieren	412
8.1.6	Syntaxzusammenfassung	417
8.2	Das Menüband	418
8.2.1	Manuelle Anpassung des Menübands	419
8.2.2	Programmierte Anpassung des Menübands	423
8.2.3	RibbonX-Controls	430
8.2.4	Erweiterte Programmiertechniken	443
8.2.5	Klassische Menüs und Symbolleisten nachbilden	449
8.2.6	Anpassungen permanent verfügbar machen	452
8.2.7	Syntaxzusammenfassung	453
8.3	Die Symbolleiste für den Schnellzugriff	455
8.3.1	Symbolleiste für den Schnellzugriff manuell anpassen	455
8.3.2	Symbolleiste für den Schnellzugriff programmiert anpassen	457
8.3.3	Syntaxzusammenfassung	458
8.4	Kontextmenüs	458
8.4.1	Kontextmenüs programmiert anpassen	459
8.4.2	Syntaxzusammenfassung	461
8.5	Die Backstage-Ansicht	462
8.5.1	Grundlagen der Programmierung	462
8.5.2	Backstage-spezifische Steuerelemente	463
8.5.3	Befehle in den FastCommand-Bereich einfügen	464
8.5.4	Eigene Backstage-Tabs anlegen	465
8.5.5	Excel-eigene Backstage-Tabs anpassen	470
8.5.6	Syntaxzusammenfassung	473

TEIL III: Anwendung **475**

9 Mustervorlagen und „intelligente“ Formulare **477**

9.1	Grundlagen	477
9.1.1	Gestaltungselemente für „intelligente“ Formulare	479
9.1.2	Mustervorlagen mit Datenbankbindung	485
9.2	Beispiel: Das „Speedy“-Rechnungsformular	488

9.3	Beispiel: Abrechnungsformular für einen Car-Sharing-Verein	496
9.4	Grenzen „intelligenter“ Formulare	503
10	Diagramme und Zeichnungsobjekte	505
10.1	Umgang mit Diagrammen	505
10.1.1	Grundlagen	505
10.1.2	Diagrammtypen	506
10.1.3	Diagrammelemente (Diagrammobjekte) und Formatierungsmöglichkeiten	507
10.1.4	Ausdruck	511
10.2	Programmierung von Diagrammen	511
10.2.1	Objekthierarchie	512
10.2.2	Programmiertechniken	516
10.3	Beispiel: Automatische Datenprotokollierung	521
10.3.1	Die Bedienung des Beispielsprogramms	522
10.3.2	Programmcode	523
10.4	Syntaxzusammenfassung	534
10.5	Die Zelldiagramme der Bedingten Formatierung	535
10.5.1	Programmierung von Datenbalkendiagrammen	537
10.5.2	Programmierung von Farbskalendiagrammen	538
10.5.3	Programmierung von Symbolsatzdiagrammen	540
10.5.4	Syntaxzusammenfassung	542
10.6	Sparklines-Diagramme	543
10.6.1	Programmierung von Sparklines-Diagrammen	544
10.6.2	Syntaxzusammenfassung	548
10.7	SmartArt-Diagramme	548
10.7.1	Programmierung von SmartArt-Diagrammen	549
10.7.2	Benutzerdefinierte SmartArt-Diagramme	554
10.7.3	Syntaxzusammenfassung	555
10.8	Neue Diagrammtypen in Excel 2016	556
10.8.1	Programmierung von Wasserfall-Diagrammen	556
10.8.2	Programmierung von Histogrammen	558
10.8.3	Programmierung von Pareto-Diagrammen	560
10.8.4	Programmierung von Kastengrafik-Diagrammen	561
10.8.5	Programmierung von Treemap-Diagrammen	563
10.8.6	DirectoryMap – Inhaltsverzeichnisse visualisieren	564
10.8.7	Programmierung von Sunburst-Diagrammen	568
10.9	Zeichnungsobjekte (Shapes)	570
11	Datenverwaltung in Excel	575
11.1	Grundlagen	575
11.1.1	Einleitung	576
11.1.2	Kleines Datenbankglossar	577
11.1.3	Excel versus Datenbanksysteme	578

11.2	Datenverwaltung innerhalb von Excel	580
11.2.1	Eine Datenbank in Excel erstellen	580
11.2.2	Daten über die Datenbankmaske eingeben, ändern und löschen	583
11.2.3	Daten sortieren, suchen, filtern	585
11.3	Datenverwaltung per VBA-Code	592
11.3.1	Programmiertechniken	592
11.3.2	Syntaxzusammenfassung	595
11.4	Datenbank-Tabellenfunktionen	596
11.5	Tabellen konsolidieren	599
11.5.1	Grundlagen	599
11.5.2	Konsolidieren per VBA-Code	602
11.6	Beispiel: Abrechnung eines Car-Sharing-Vereins	603
11.6.1	Bedienung	603
11.6.2	Überblick über die Komponenten der Anwendung	606
11.6.3	Programmcode	608
12	Zugriff auf externe Daten	617
12.1	Grundkonzepte relationaler Datenbanken	617
12.2	Import externer Daten	623
12.2.1	Datenimport mit Power Query	624
12.2.2	Datenimport mit MS Query	632
12.2.3	Das QueryTable-Objekt	643
12.2.4	Excel-Daten exportieren	646
12.3	Datenbankzugriff mit der ADO-Bibliothek	647
12.3.1	Einführung	647
12.3.2	Verbindungsaufbau (Connection)	652
12.3.3	Datensatzlisten (Recordset)	655
12.3.4	SQL-Kommandos (Command)	662
12.3.5	SQL-Grundlagen	663
12.3.6	Syntaxzusammenfassung	666
12.4	Beispiel: Fragebogenauswertung	668
12.4.1	Überblick	668
12.4.2	Aufbau des Fragebogens	671
12.4.3	Aufbau der Datenbank	673
12.4.4	Programmcode	675
13	Datenanalyse in Excel	685
13.1	Daten gruppieren (Teilergebnisse)	685
13.1.1	Einführung	685
13.1.2	Programmierung	687
13.2	Pivot-Tabellen (Kreuztabellen)	689
13.2.1	Einführung	689
13.2.2	Gestaltungsmöglichkeiten	693
13.2.3	Pivot-Tabellen für externe Daten	698

13.2.4	Pivot-Tabellenoptionen	702
13.2.5	Pivot-Diagramme	703
13.3	Programmiertechniken	703
13.3.1	Pivot-Tabellen erzeugen und löschen	704
13.3.2	Aufbau und Bearbeitung vorhandener Pivot-Tabellen	708
13.3.3	Interne Verwaltung (PivotCache)	712
13.3.4	Syntaxzusammenfassung	718
14	XML- und Listenfunktionen	721
14.1	Bearbeitung von Listen	721
14.2	XML-Grundlagen	723
14.3	XML-Funktionen interaktiv nutzen	726
14.4	XML-Programmierung	730
15	Excel-Programmierung für Fortgeschrittene	737
15.1	Excel-Add-ins	737
15.2	Excel und das Internet	742
15.2.1	Excel-Dateien als E-Mail versenden	742
15.2.2	HTML-Import	744
15.2.3	HTML-Export	745
15.3	Smart Tags	747
15.4	Web Services nutzen	750
15.5	Dynamic Link Libraries (DLLs) verwenden	756
15.6	ActiveX-Automation (COM)	761
15.6.1	Excel als Client (Steuerung fremder Programme)	762
15.6.2	Excel als Server (Steuerung durch fremde Programme)	768
15.6.3	Neue Objekte für Excel (Clipboard-Beispiel)	772
15.6.4	Object Linking and Embedding (OLE)	774
15.6.5	Automation und Visual Basic .NET	778
15.6.6	Programme ohne ActiveX starten und steuern	786
15.6.7	Syntaxzusammenfassung	788
15.7	64-Bit-Programmierung	789
15.7.1	Kompatibilitätsprobleme	789
15.7.2	Ein problematisches (32-Bit-)Beispiel	790
15.7.3	Syntaxzusammenfassung	795
15.8	Visual Studio Tools for Office	796
15.8.1	Bestandsaufnahme: die Grenzen von VBA	796
15.8.2	VSTO: Profi-Werkzeug für Profi-Entwickler	797
15.8.3	Grundlagen des VSTO-Einsatzes	799
15.8.4	Beispielprojekte	803
15.8.4.1	Individuelle Aufgabenbereiche anlegen	803
15.8.4.2	Anpassen des Menübands	805
15.8.4.3	Abfragen von Web Services	808

15.9 Office-Add-ins	811
15.9.1 Bestandteile eines Office-Add-ins	812
15.9.2 Typen von Office-Add-ins	813
15.9.3 Werkzeuge für die Entwicklung von Office-Add-ins	815
15.9.4 Beispiel 1: SimpleApp	815
15.9.5 Das JavaScript-API für Office	822
15.9.6 Beispiel 2: ComplexApp	826

Anhang

831

A Inhalte der Download-Dateien zum Buch	831
A.1 Objektreferenz	831
A.2 Hyperlinks	831
A.3 Beispieldateien	832
B Verwendete Literatur	832
C Nachweis der Grafiken & Icons	833

Stichwortverzeichnis

835

Vorwort

Excel bietet von Haus aus ein riesiges Spektrum von Funktionen. Wozu sollten Sie dann noch selber Makros, Add-ins diverser Art und andere Programmerweiterungen mit VBA, Visual Studio oder anderen Werkzeugen entwickeln? Weil Sie damit ...

- ... eigene Tabellenfunktionen programmieren können, die einfacher anzuwenden sind als komplizierte Formeln.
- ... Excel nach Ihren Vorstellungen konfigurieren und auf diese Weise eine einfachere und effizientere Programmbedienung erreichen können.
- ... komplexe Arbeitsschritte wie etwa das Ausfüllen von Formularen durch „intelligente“ Formulare (alias Mustervorlagen) strukturieren und erleichtern können.
- ... immer wieder auftretende Arbeitsvorgänge automatisieren können. Das empfiehlt sich besonders dann, wenn regelmäßig große Datenmengen anfallen, die verarbeitet, analysiert und grafisch aufbereitet werden sollen.
- ... eigenständige und leistungsfähige Excel-Lösungen erstellen können, die sich durch maßgeschneiderte Bedienelemente nahtlos in das Menüband, die sogenannte Backstage-Ansicht (im Datei-Menü) oder andere Teile der Excel-Oberfläche integrieren.

Damit lassen sich Excel-Anwendungen in ihrer Bedienung so weit vereinfachen, dass sie von anderen Personen (auch von Excel-Laien) ohne lange Einweisung verwendet werden können.

Das notwendige Know-how für alle diese Formen der Excel-Programmierung finden Sie in diesem Buch. *Übrigens: Auch wenn Excel 2016 auf dem Titel steht, so gilt das Gesagte – oder besser: Geschriebene – doch für alle Programmversionen ab 2007 (und zum größten Teil auch für die Versionen davor).*

Wenn es Dinge gibt, die in einer älteren Version anders funktionieren als in Excel 2016, so wird das ausdrücklich erwähnt. Falls das wider Erwarten einmal nicht der Fall sein sollte, bitten wir schon jetzt um Verzeihung. Bei so vielen Versionen verlieren auch erfahrene Autoren manchmal den Überblick.

2007 bis 2016 – Neues in Excel

Mit der radikal neuen Multifunktionsleiste, die die früheren Menüs und Symbolleisten plötzlich sehr alt aussehen ließ (und letztlich in Rente schickte), war Excel 2007 eine echte Revolution. Excel 2010 ließ es entwicklungstechnisch deutlich ruhiger angehen und bescherte uns statt einer großen Revolution viele kleine Evolutionen.

Eine davon war der neue *Oberflächeneditor*, mit dem wir nicht mehr nur die unscheinbare „Symbolleiste für den Schnellzugriff“ nach unseren Wünschen konfigurieren dürfen, sondern die komplette Multifunktionsleiste. Die heißt nun übrigens „*Menüband*“ (siehe Abschnitt 8.2) und beschränkt sich auf solche Befehle, die der Bearbeitung von Dokumentinhalten dienen. Für alle anderen Befehle, die das Dokument als Ganzes betreffen (Speichern, Drucken etc.), hat Microsoft die sogenannte *Backstage-Ansicht* (siehe Abschnitt 8.5) erfunden, die das Office-Menü von Excel 2007 ersetzt. Menüband und Backstage-Ansicht bilden seither die Kommandozentrale von Excel und zeichnen sich durch eine konsequente Aufgabenteilung aus.

Konsequenz zeigte Microsoft auch bei der *Oberflächenprogrammierung*. Hier gilt seit Excel 2010 für alle Bestandteile – Menüband, Backstage-Ansicht, Symbolleiste für den Schnellzugriff und Kontextmenüs – das gleiche „duale Prinzip“: XML-Code bestimmt das Design, VBA-Code die Funktion. Mit dem Know-how, das Sie sich womöglich schon bei der Anpassung der früheren Multifunktionsleiste erworben haben, können Sie jetzt also die gesamte Excel-Oberfläche verändern und eigene Lösungen integrieren (Kapitel 8).

Evolutionär präsentierte sich Excel 2010 auch bei der Visualisierung von Zahlen. So fanden die *SmartArt-Diagramme* (Abschnitt 10.7), die mit der Version 2007 eingeführt wurden, Eingang in das Objektmodell, so dass man sie nun programmatisch erstellen oder verändern kann. Darüber hinaus hat Excel 2010 der Welt die sogenannten *Sparklines-Diagramme* (Abschnitt 10.6) beschert, ein seinerzeit völlig neuer und ebenfalls programmierbarer Diagrammtyp, der in eine einzelne Zelle passt und sich insbesondere für die Visualisierung von Trends eignet.

Wo Licht ist, ist bekanntlich auch Schatten. Und das gilt insbesondere für die Tatsache, dass es Excel seit der Version 2010 auch in einer *64-Bit-Version* zu kaufen gibt. Dass die nicht nur Vorteile hat, sondern auch massive Nachteile in Form von diversen Inkompatibilitäten, zeigt der Abschnitt 15.7 (und was Sie dagegen tun können, natürlich auch).

Excel 2013 präsentierte sich dem Anwender erstmals in einem nüchternen, von Schatten und Transparenzeffekten befreiten Look, der sich an der Optik von Windows 8 orientierte. Und dazu passend fand sich eine neuerlich aufgeräumte und entschlackte Menüband- und Backstage-Oberfläche, in der man so manchen Befehl aus früheren Versionen leider nicht mehr finden konnte.

Als Ausgleich gab es neue Funktionen wie *Schnellanalyse* und *Empfohlene Diagramme*, die die Erstellung von Diagrammen beschleunigten. Arbeitsmappen ließen sich standardmäßig „in der Cloud“ und somit online speichern, manche Diagramme in animierter Form anzeigen und Pivot-Tabellen auf der Basis mehrerer Listen beziehungsweise Tabellen generieren. Unter der Haube gab es die eine oder andere neue Tabellenfunktion zu entdecken, unter anderem für das direkte Anzapfen von Webdiensten (siehe Abschnitt 15.4).

Der wichtigste und aus Entwicklersicht interessanteste Neuzugang aber war die *App für Office*. Dabei handelte es sich um ein seinerzeit völlig neues Erweiterungskonzept, das Webtechniken an die Stelle von VBA-Makros setzen wollte (und weiterhin will). Wie (und ob) das funktioniert, ist detailliert in Kapitel 15.9 beschrieben.

Und was gibt es Neues in der jüngsten Excel-Version 2016? Aus Anwendersicht wären da wohl vor allem stark verbesserte Funktionen für die gemeinsame (Echtzeit-)Arbeit an Dokumenten zu nennen, die Anpassung an Windows 10 und Touch-Bedienung sowie die neue

Hilfefunktion „*Was möchten Sie tun?*“, die den User ohne Umweg über wortreiche Schritt-für-Schritt-Anleitungen direkt zur gesuchten Funktion führt.

Aus der Sicht des Programmierers sind andere Neuerungen aber viel interessanter. So gibt es jetzt eine Reihe neuer Diagrammtypen wie *Wasserfall*-, *Pareto*- oder *Treemap*-Diagramme, die nicht nur visuell überzeugen, sondern sich auch noch programmatisch generieren und verändern lassen. Wie das funktioniert, haben wir in Kapitel 10.8 detailliert beschrieben.

Ein weiteres Highlight: Microsoft hat die Funktionalität des ehemaligen *Power-Query*-Add-ins nun vollständig in Excel (und dessen Objektmodell) integriert. Damit steht uns Entwicklern jetzt ebenfalls ein extrem mächtiges Tool zur Verfügung, mit dem sich Daten aus nahezu jeder Datenquelle auswählen, aufbereiten und in ein Excel-Arbeitsblatt importieren lassen. Kapitel 12.2.1 vermittelt Ihnen das dazu notwendige Know-how.

Natürlich hat es auch wieder die eine oder andere Änderung im Vergleich zu früheren Excel-Versionen gegeben – was immer etwas Verwirrung stiftet. So heißen die frisch vorgestellten *Apps für Office* (alias *Office-Apps*) nun plötzlich *Office-Add-ins*, was dazu führt, dass wir es in Excel 2016 erstmals mit drei verschiedenen Arten von Add-ins zu tun haben, nämlich „Excel-Add-ins“ (siehe Kapitel 15.1), „COM-Add-ins“ (Kapitel 15.6 und 15.8) sowie besagte „Office-Add-ins“. Bei Letzteren hat es neben der neuen Namensgebung auch diverse Änderungen und Erweiterungen in Sachen Programmierung gegeben. Die Details finden Sie in Kapitel 15.9.

Warum dieses Buch?

Im Gegensatz zu anderen Titeln, die sich mit dem Thema Excel-Programmierung beschäftigen, liefert Ihnen dieses Buch *keine* systematische Beschreibung von Objekten, ihren Eigenschaften und Methoden oder VBA-Befehlen. Wer so etwas sucht, ist mit der Hilfefunktion des VBA-Editors und mit zahlreichen Internetquellen besser bedient.

Anstelle einer umfassenden Referenz stehen bei diesem Buch praktische Lösungen und Anwendungsmöglichkeiten im Vordergrund. Die zugehörigen Code-Beispiele lassen sich relativ leicht an eigene Bedürfnisse anpassen, was die Entwicklungszeit für manches berufliche oder private Programmiervorhaben spürbar verkürzen kann.

Dass man bei praxisbezogenen Projekten natürlich auch sehr viel über Objekte (die wichtigsten sogar!), vor allem aber über sinnvolle Formen ihres programmierten Zusammenarbeitens erfährt, ist quasi ein Nebeneffekt. Gleichzeitig nennen wir aber auch die Probleme Excels beim Namen, um Ihnen die langwierige Suche nach Fehlern zu ersparen, die Sie gar nicht selbst verursacht haben.

Neben konkreten Programmierlösungen liefert Ihnen dieses Buch aber auch sehr viel Insider-Wissen über die Bedienung von Excel. Damit werden Sie so ganz nebenbei zum „Power-User“ und können so manches Anwendungsproblem mit ein paar Mausklicks lösen, für das Sie ansonsten womöglich ein Programm geschrieben hätten ... ;-)

Jenseits von VBA

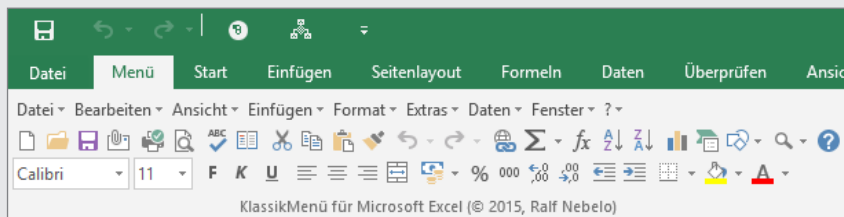
Obwohl VBA immer noch das wichtigste Werkzeug für die Entwicklung von Excel-Lösungen ist (und daher im Mittelpunkt dieses Buchs steht), stellen sich zunehmend mehr Aufgaben, die mit der „eingebauten“ Programmiersprache des Kalkulationsprogramms nur noch teilweise oder gar nicht mehr zu lösen sind. Beispiele sind etwa die Anpassung von Menüband

(siehe Abschnitt 8.2.2) und Backstage-Ansicht (8.5.1), die Programmierung individueller Aufgabenbereiche (15.8.4.1), die Abfrage von Web Services (15.4) oder die Integration von Webtechniken in Form der neuen Office-Add-ins (15.9).

Damit Sie solche Aufgaben dennoch meistern können, stellt Ihnen dieses Buch die erforderlichen Werkzeuge vor und liefert Ihnen das notwendige Know-how für den erfolgreichen Einsatz. Das erspart Ihnen mühsame Recherchen im Internet, den Kauf weiterer Bücher und lässt Sie mitunter auch kleine programmiertechnische „Wunder“ vollbringen – die Wiederbelebung der mit Excel 2003 „entschlafenen“ Menüs und Symbolleisten (siehe Abschnitt 8.2.5) beispielsweise. Darüber dürften sich insbesondere altgediente Excel-Anwender freuen, die sich selbst in der nunmehr vierten Ribbon-Version von Excel noch immer nicht im Menüband zurechtfinden.



Tipp



Die Datei *KlassikMenü.xlam* im Unterordner 8 der Beispieldateien enthält eine vollständige Nachbildung der Menü- und Symbolleiste von Excel 2003. Sie können diese Datei als sofort nutzbares Add-in in Excel ab Version 2007 einbinden. Abschnitt 8.2.6 verrät, wie Sie dazu vorgehen müssen.

Viel Erfolg!

Die Beispiele dieses Buchs zeigen, wie weit Excel-Programmierung gehen kann. Die Möglichkeiten sind beinahe unbegrenzt! Wer sie nutzen will, muss sich aber nicht mehr nur im komplexen Objektmodell von Excel und in VBA zurechtfinden, sondern zunehmend auch in angrenzenden Programmierwelten.

Dabei will Ihnen dieses Buch eine praktische Orientierungshilfe sein. Mit zunehmender Übersicht und Erfahrung beginnt dann die Office-Programmierung mit VBA, Visual Studio, XML, JavaScript und diversen anderen Werkzeugen richtig Spaß zu machen.

Und wenn das der Fall ist, lässt auch der gewünschte Erfolg nicht lange auf sich warten. Genau den wünschen wir Ihnen von Herzen!

Michael Kofler und Ralf Nebelo, März 2016

<http://www.kofler.info>

■ Konzeption des Buchs

Visual Basic für Applikationen (oder kurz: VBA) ist eine sehr leistungsfähige Programmiersprache. Die große Zahl von Schlüsselwörtern bringt aber auch viele Probleme mit sich. Während des Einstiegs ist es so gut wie unmöglich, auch nur halbwegs einen Überblick über VBA zu gewinnen. Und selbst nach monatelanger Programmierung mit VBA wird die Hilfe der wichtigste Ratgeber zu den Details eines bestimmten Schlüsselworts bleiben. Dieses Buch versucht deswegen ganz bewusst, das zu bieten, was in der Originaldokumentation bzw. in der Hilfe zu kurz kommt:

- detaillierte Informationen für die Entwicklung eigener VBA-Lösungen und deren Integration in Menüband, Backstage-Ansicht und andere Bestandteile der Excel-Oberfläche,
- „echte“ Anwendungen in Form von konkreten, realitätsbezogenen Beispielen,
- themenorientierte Syntaxzusammenfassungen (z.B. alle Eigenschaften und Methoden zur Bearbeitung von Zellbereichen),
- aussagekräftige Beschreibungen der wichtigsten Objekte von VBA und ihre Einordnung in die Objekthierarchie.

Darüber hinaus liefert Ihnen dieses Buch sehr viel Know-how für fortgeschrittene Programmierthemen, bei denen VBA nicht unbedingt im Mittelpunkt steht:

- Einsatz von DLL-Funktionen,
- ActiveX-Automation,
- Programmierung eigener Add-ins,
- Verwendung von Web Services,
- 64-Bit-Programmierung,
- Realisierung von Office-Anwendungen mit den Visual Studio Tools for Office (VSTO),
- Entwicklung von Office-Add-ins mit Hilfe von Webtechnologien.

Einem Anspruch wird das Buch aber ganz bewusst nicht gerecht: dem der Vollständigkeit. Es erscheint uns sinnlos, Hunderte von Seiten mit einer Referenz aller Schlüsselwörter zu füllen, wenn Sie beinahe dieselben Informationen auch in der Hilfe finden können. Anstatt zu versuchen, auch nur den Anschein der Vollständigkeit zu vermitteln, haben wir uns bemüht, wichtigeren Themen den Vorrang zu geben und diese ausführlich, fundiert und praxisorientiert zu behandeln.

Formalitäten

Die Namen von Menüs, Befehlsregisterkarten, Symbolen, Buttons und anderer Dialog- und Oberflächenelemente werden in Kapitälchen dargestellt: DATEI|ÖFFNEN, ABRUCH oder OK. Die Anweisung ÜBERPRÜFEN|BLATT SCHÜTZEN|ZELLEN FORMATIEREN meint, dass Sie zuerst die Befehlsregisterkarte ÜBERPRÜFEN öffnen, den Befehl BLATT SCHÜTZEN anklicken und im daraufhin erscheinenden Dialog das Kontrollkästchen ZELLEN FORMATIEREN auswählen sollen.

VBA-Schlüsselwörter, Variablen- und Prozedurnamen sowie Datei- und Verzeichnisnamen werden *kursiv* angegeben, etwa *Application*-Objekt, *Visible*-Eigenschaft, *strName*-Variable oder *C:\Muster.xlsm*. Tabellenfunktionen wie *WENN()* erscheinen in der gleichen Schrift,

aber in Großbuchstaben. (Tabellenfunktionen sind auch anhand der Sprache von VBA-Schlüsselwörtern zu unterscheiden: VBA-Schlüsselwörter sind grundsätzlich englisch, Tabellenfunktionsnamen immer deutsch.)

Beispielcode, Beispieldateien, Download-Dateien zum Buch

Aus Platzgründen sind in diesem Buch immer nur die wichtigsten Code-Passagen der Beispielprogramme abgedruckt. Den vollständigen Code finden Sie unter Download-Dateien zum Buch, die Sie unter der folgenden Internetadresse herunterladen können:

<http://downloads.hanser.de/>

Die Beispieldateien sind in Verzeichnissen angeordnet, deren Namen den Kapitelnummern entsprechen. VBA-Code in diesem Buch beginnt immer mit einem Kommentar im Format Verzeichnis\Dateiname, der auf die entsprechende Beispieldatei verweist:

```
' 01\format.xlsm
Sub FormatAsResult()
    Selection.Style = "result"
End Sub
```

Im Fall von XML-Code (den Sie hauptsächlich in Kapitel 8 finden) haben die Kommentare die folgende Form:

```
<!-- 08\Menüband_Button.xlsm -->
```

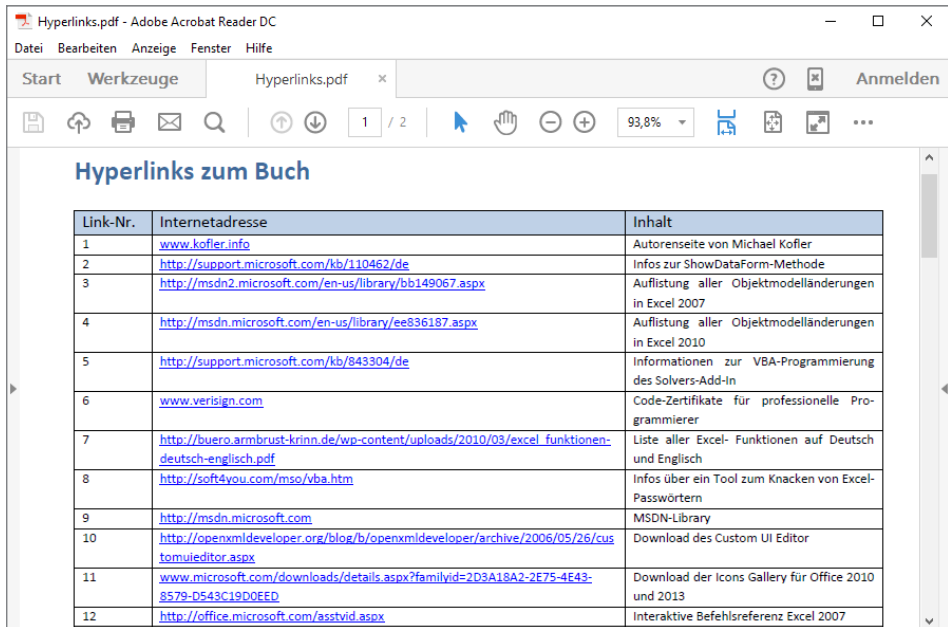
Kommentare in JavaScript-Dateien (Kapitel 15.9) schließlich sehen so aus:

```
/* 15\OfficeApps\SimpleApp\ComplexApp.js */
```

Internetadressen (Hyperlinks.pdf)

Der Text dieses Buchs enthält zahlreiche Verweise auf Internetadressen, wo Sie weiterführende Informationen finden, Tools herunterladen können etc. Da viele dieser „Links“ zu kompliziert sind, um sie abzutippen, haben wir sie in einem PDF-Dokument zusammengefasst. Es trägt den Namen *Hyperlinks.pdf* (siehe nächste Seite) und ist ebenfalls bei den Download-Dateien zum Buch im Ordner *Info* zu finden.

Die Links in diesem Dokument sind jeweils mit einer Nummer gekennzeichnet, die Sie auch im Buchtext in der Form *[Link x]* finden, wobei „x“ für die konkrete Link-Nummer steht. Zum Öffnen eines Links genügt ein Mausklick. Beim ersten Mal müssen Sie Ihrem Reader-Programm unter Umständen die Erlaubnis dazu erteilen.



Link-Nr.	Internetadresse	Inhalt
1	www.kofler.info	Autorenseite von Michael Kofler
2	http://support.microsoft.com/kb/110462/de	Infos zur ShowDataForm-Methode
3	http://msdn2.microsoft.com/en-us/library/bb149067.aspx	Auflistung aller Objektmodelländerungen in Excel 2007
4	http://msdn.microsoft.com/en-us/library/ee836187.aspx	Auflistung aller Objektmodelländerungen in Excel 2010
5	http://support.microsoft.com/kb/843304/de	Informationen zur VBA-Programmierung des Solvers-Add-In
6	www.verisign.com	Code-Zertifikate für professionelle Programmierer
7	http://buero.armbrust-krinn.de/wp-content/uploads/2010/03/excel_funktionen-deutsch-englisch.pdf	Liste aller Excel-Funktionen auf Deutsch und Englisch
8	http://soft4you.com/mso/vba.htm	Infos über ein Tool zum Knacken von Excel-Passwörtern
9	http://msdn.microsoft.com	MSDN-Library
10	http://openxmldeveloper.org/blog/b/openxmldeveloper/archive/2006/05/26/customuieditor.aspx	Download des Custom UI Editor
11	www.microsoft.com/downloads/details.aspx?familyid=2D3A18A2-2E75-4E43-8579-D543C19D0EED	Download der Icons Gallery für Office 2010 und 2013
12	http://office.microsoft.com/asstvid.aspx	Interaktive Befehlsreferenz Excel 2007

Die Internetadressen in der Datei *Hyperlinks.pdf* können Sie direkt per Mausklick öffnen.

Und eine Entschuldigung

Wir sind uns bewusst, dass unter den Lesern dieses Buchs auch zahlreiche Frauen sind. Dennoch ist in diesem Buch immer wieder von *dem Anwender* die Rede, wenn wir keine geschlechtsneutrale Formulierung gefunden haben. Wir bitten dafür alle Leserinnen ausdrücklich um Entschuldigung. Wir sind uns des Problems bewusst, empfinden Doppelgleisigkeiten der Form *der/die Anwender/in* oder kurz *AnwenderIn* aber sprachlich als nicht schön – und zwar sowohl beim Schreiben als auch beim Lesen.

TEIL I

Intuitiver Einstieg

1

Das erste Makro

Im Verlauf des Kapitels lernen Sie Begriffe wie Makro oder Visual Basic für Applikationen kennen, zeichnen selbst Makros auf, integrieren diese in die Symbolleiste für den Schnellzugriff, stellen eine einfache Datenbankanwendung zusammen etc. Das Kapitel gibt Ihnen einen ersten Einblick in Themen, die im Buch später viel ausführlicher aufgegriffen werden.

■ 1.1 Begriffsdefinition

Makro

Dieses Kapitel steht unter dem Motto „Das erste Makro“. Daher soll zunächst einmal der Begriff „Makro“ erklärt werden:

Ein Makro bezeichnet eine Reihe von Anweisungen an den Computer, die dieser ausführt, sobald er dazu aufgefordert wird.

Welchen Zweck haben Makros? Hier ein paar Antworten:

- Sie können Arbeitsschritte, die sich häufig wiederholen, automatisieren und vereinfachen.
- Sie können Excel und dessen Benutzeroberfläche ganz an Ihre Bedürfnisse anpassen.
- Sie können die Bedienung von Excel für andere Anwender vereinfachen, sodass diese praktisch ohne Schulung konkrete Excel-Anwendungen bedienen können.
- Und schließlich können Sie echte „Programme“ (Anwendungen) schreiben, denen man kaum mehr anmerkt, dass sie in Excel entstanden sind.

Da der Computer natürlichsprachliche Anweisungen wie „Speichere diese Datei!“ oder „Stelle die drei markierten Zellen in einer größeren Schrift dar!“ leider nicht versteht, müssen Makroanweisungen in einer speziellen Sprache formuliert werden. Aus Kompatibilitätsgründen stellt Excel zwei Sprachen zur Auswahl:

- Die *herkömmliche Makroprogrammiersprache* hat sich im Verlauf der ersten Excel-Versionen gebildet. Makros in dieser Sprache heißen herkömmliche Makros oder Excel-4-Makros, weil die Grundkonzepte dieser Sprache seit der Programmversion 4 nicht mehr verändert oder erweitert wurden. Die Schlüsselwörter der herkömmlichen Makroprogrammiersprache werden in deutscher Sprache angegeben.

- Mit Excel 5 wurde die Sprache *Visual Basic für Applikationen* (kurz VBA) eingeführt. Sie bietet mehr und ausgefeiltere Möglichkeiten der Programmsteuerung, wirkt aber vielleicht auf den ersten Blick etwas umständlich (insbesondere dann, wenn Sie schon einmal ein herkömmliches Makro geschrieben haben).



Hinweis

In Excel 5 war der deutsche VBA-Dialekt die Default-Einstellung. Bereits mit Version 7 vollzog Microsoft dann aber eine Kehrtwendung: Plötzlich wurden englische Schlüsselwörter bevorzugt. In Version 97 wurde der deutsche Dialekt dann vollständig gestrichen. Deutschsprachiger VBA-Code wird beim Laden alter Dateien automatisch konvertiert. Dieses Buch beschreibt daher ausschließlich die englische VBA-Variante!

Zu den Makrosprachen gleich ein Beispiel, das die aktuelle Arbeitsmappe speichert (zuerst als herkömmliches Makro, dann in VBA):

<code>=SPEICHERN()</code>	'herkömmliches Makro (Excel 4)
<code>AktiveArbeitsmappe.Speichern</code>	'VBA deutsch (Excel 5 und 7)
<code>ActiveWorkbook.Save</code>	'VBA englisch (Excel 97 und folgende)

Das zweite Beispiel stellt in zuvor markierten Zellen eine etwas größere Schriftart ein (wiederum zuerst als herkömmliches Makro, dann in VBA):

<code>=SCHRIFTART.EIGENSCHAFTEN(;;ZELLE.ZUORDNEN(19)+2)</code>	'Excel 4
<code>Selection.Font.Size = Selection.Font.Size + 2</code>	'VBA englisch

Aus dem Begriff Makro allein geht die gewählte Makrosprache nicht hervor. In diesem Buch meint Makro aber immer ein VBA-Makro.



Anmerkung

Die obigen Beispiele sind in dieser Form nicht verwendbar. Ein Excel-4-Makro muss mit dem Namen des Makros beginnen und mit dem Kommando `=RÜCKSPRUNG()` enden. VBA-Makros müssen zwischen `Sub Name()` und `End Sub` eingeklammert werden. Die Syntaxkonventionen von Visual Basic gehen aus den Beispielen dieses Kapitels hervor. Eine detaillierte Beschreibung der VBA-Syntax bietet Kapitel 4.

Makros aufzeichnen

Generell gibt es zwei Möglichkeiten, Makros zu erstellen: Entweder Sie tippen die Kommandos über die Tastatur ein oder Sie lassen sich das Makro von Excel „aufzeichnen“. Damit ist gemeint, dass Sie (über Maus und Tastatur) Daten eingeben, Zellen formatieren, Kommandos ausführen etc. Excel verfolgt Ihre Aktionen und schreibt die entsprechenden VBA-Anweisungen in ein Modul. Wenn Sie das so erzeugte Makro später ausführen, werden exakt dieselben Arbeitsschritte, die Sie vorher manuell ausgeführt haben, durch das Makro wiederholt.

In der Realität erfolgt die Erstellung von Makros zumeist in einem Mischmasch aus beiden Methoden. Sie werden zwar immer wieder einzelne Arbeitsschritte von Excel aufzeichnen lassen, ebenso wird es aber auch oft notwendig sein, diese Makros später zu ändern oder zu erweitern.

Makros ausführen

Die unbequemste Form, ein Makro auszuführen, bietet das Kommando **ANSICHT | MAKROS | MAKROS ANZEIGEN**. Es stellt Ihnen eine Liste mit allen definierten Makros in allen geladenen Arbeitsmappen zur Auswahl. Sobald Sie einen der Makronamen anklicken, wird das entsprechende Makro ausgeführt.

Daneben bestehen aber elegantere Varianten: Sie können ein Makro mit einem beliebigen Symbol in der Symbolleiste für den Schnellzugriff oder mit einer Tastaturabkürzung Strg+Anfangsbuchstabe verbinden. Sobald Sie das Symbol anklicken oder die Tastaturabkürzung eingeben, wird das Makro sofort ausgeführt. Solcherart definierte Makros können eine enorme Arbeitserleichterung bedeuten, wie die Beispiele der folgenden Abschnitte beweisen.

Bis Excel 2003 bestand zudem die Möglichkeit, Makros über selbst definierte Menübefehle zu aktivieren. Da es seit Excel 2007 keine klassische Menüleiste mehr gibt, fällt diese Möglichkeit weg. Ersatzweise können Sie aber das Menüband um eigene Makrostartbefehle erweitern. Das dazu notwendige Verfahren erfordert ein wenig Programmierung und wird ausführlich in Abschnitt 8.2 beschrieben.

Es besteht sogar die Möglichkeit, Makros automatisch beim Eintreten bestimmter Ereignisse ausführen zu lassen. Excel kennt eine ganze Menge solcher Ereignisse – etwa den Wechsel des aktiven Arbeitsblatts, die Neuberechnung des Blatts, das Speichern der Arbeitsmappe etc. Ereignisprozeduren werden in Abschnitt 4.4 ausführlich behandelt.

Makros und Programme

Vielen Excel-Anwendern – sogar solchen, die bereits Makros erstellt haben – stehen die Haare zu Berge, wenn sie den Begriff „programmieren“ hören. Programmieren, das sei nur etwas für Profis mit Fach- oder Hochschulausbildung, lautet eine immer wieder vertretene Ansicht. In Wirklichkeit sind Sie bereits ein Programmierer, sobald Sie das erste – vielleicht nur dreizeilige – Makro erstellt haben. Jedes Makro stellt im Prinzip ein echtes Programm dar.

In diesem Buch wird der Begriff Programm zumeist etwas weiter gefasst. Ein Programm meint eine eigenständige Excel-Anwendung, die sich zumeist durch eigene Menübandkommandos, eigene Dialoge und eine oft große Anzahl von Makros auszeichnet. Dieses Buch leitet Sie vom ersten Makro (in diesem Kapitel) bis zu umfangreichen Anwendungen.

■ 1.2 Was ist Visual Basic für Applikationen?

Visual Basic für Applikationen ist eine Makroprogrammiersprache. Mit VBA können Sie Excel-Anwendungen automatisieren oder in ihrer Bedienung vereinfachen. Die Einsatzmöglichkeiten von VBA reichen so weit, dass Sie damit vollkommen eigenständige Programme erstellen können, denen kaum mehr anzumerken ist, dass es sich eigentlich um Excel-Anwendungen handelt. Einführungs- und Anwendungsbeispiele einfacher Makros finden Sie in diesem Kapitel.

Geschichtliches

Die herkömmliche Makrosprache von Excel hat sich ursprünglich aus dem Wunsch heraus entwickelt, neue Tabellenfunktionen zu definieren und wiederholt auftretende Kommandos zu einer Einheit (zu einem Makro) zusammenzufassen. Um die Bedienung von Excel-Anwendungen möglichst einfach zu gestalten, wurden außerdem die Veränderung der (früheren) Menüs und die Definition eigener Dialoge ermöglicht. Verbunden mit dem riesigen Funktionsspektrum von Excel hat sich daraus bis Version 4 eine ziemlich unübersichtliche Makrosprache entwickelt.

Diese Makrosprache hat zwar eine fast uneingeschränkte Programmierung aller Excel-Funktionen erlaubt, viele Programmierprobleme ließen sich allerdings nur umständlich lösen. Die resultierenden Programme waren fehleranfällig und langsam. Bei größeren Projekten traten die Grenzen dieser Makrosprache besonders deutlich zum Vorschein. Für Anwender, die gleichzeitig mehrere Microsoft-Programme (Excel, Word, Access) verwenden, kam als weiteres Problem hinzu, dass jedes Programm mit einer eigenen Makrosprache ausgestattet ist.

Aufgrund all dieser Unzulänglichkeiten beschloss Microsoft, eine seinerzeit vollkommen neue Makroprogrammiersprache zu entwickeln, die zuerst für Excel zur Verfügung stand, inzwischen aber längst in alle Office-Anwendungen integriert wurde.

Die besonderen Merkmale von VBA

VBA ist im Gegensatz zu bisherigen Makrosprachen eine vollwertige Programmiersprache: VBA kennt alle in „echten“ Programmiersprachen üblichen Variablentypen, kann mit Zeichenketten umgehen, dynamische Felder verwalten, zur Definition rekursiver Funktionen eingesetzt werden etc.

- VBA ist **objektorientiert**: Als *Objekte* gelten beispielsweise markierte Zellbereiche, Diagramme etc. Typische Merkmale von Objekten – etwa die Ausrichtung des Zellinhalts, die Hintergrundfarbe eines Diagramms – werden über sogenannte *Eigenschaften* eingestellt. Eigenschaften sind also vordefinierte Schlüsselwörter, die zur Manipulation von Objekten vorgesehen sind. Neben den Eigenschaften gibt es noch *Methoden*, die zur Ausführung komplexer Operationen vorgesehen sind: etwa zum Erzeugen von Objekten (neuen Diagrammen, Pivot-Tabellen etc.) oder zum Löschen vorhandener Objekte. Methoden lassen sich am ehesten mit herkömmlichen Kommandos vergleichen. Der wesentliche Unterschied besteht darin, dass Methoden nur auf speziell dafür vorgesehene Objekte angewendet werden können.

- VBA ist **ereignisorientiert**: Das Anklicken eines Buttons oder eines Symbols führt zum automatischen Aufruf des dazugehörigen Makros. Als Programmierer müssen Sie sich nicht um die Verwaltung der Ereignisse kümmern, sondern lediglich Makros erstellen, die dann von Excel selbstständig aufgerufen werden.
- VBA stellt professionelle Hilfsmittel zur **Fehlersuche** zur Verfügung: Programmteile können Schritt für Schritt ausgeführt und die Inhalte von Variablen können überwacht werden. Die Programmausführung kann beim Eintreffen von bestimmten Bedingungen unterbrochen werden.
- VBA ist **erweiterungsfähig**: In jedem VBA-Dialekt kann auf Objekte anderer Anwendungen zugegriffen werden. Beispielsweise ist es möglich, in einem Excel-VBA-Programm auch die Schlüsselwörter (im Fachjargon: die *Objektbibliothek*) von Access oder Word zu nutzen. Mit Add-ins können Sie neue Excel-Funktionen und Objekte erstellen.
- Zur Ausstattung von VBA gehört ein leistungsfähiger **Dialogeditor**. Die Verwaltung von Dialogen erfolgt nach dem gleichen objekt- und ereignisorientierten Schema wie die Verwaltung von Excel-Objekten.



Hinweis

Gelegentlich stiftet der Umstand Verwirrung, dass es bei Microsoft mehrere Produkte gibt, die mit Visual Basic zu tun haben. Thema dieses Buchs ist Excel, das über die integrierte Sprache VBA gesteuert werden kann. Es gab und gibt aber auch die eigenständigen Produkte „Visual Basic 6“ und dessen Nachfolger „Visual Basic .NET“. Dabei handelt es sich um Programmiersprachen, mit denen Sie unabhängig vom Office-Paket Programme entwickeln können; die Ausführung solcher Programme setzt also nicht voraus, dass beim Anwender ebenfalls das Office-Paket installiert ist. VBA auf der einen Seite und VB6 bzw. VB.NET auf der anderen Seite weisen zwar Ähnlichkeiten auf, sind aber durchaus nicht immer kompatibel. (Insbesondere bei VB.NET gibt es sehr viele Änderungen.)

■ 1.3 Beispiel: Eine Formatvorlage mit einem Symbol verbinden

Im ersten Beispiel wird zuerst eine Formatvorlage definiert. (Eine Formatvorlage sammelt ein Bündel von Formatinformationen zu Schriftart, Ausrichtung, Rahmen und Farben. Formatvorlagen können zur Formatierung von Zellen verwendet werden.) Anschließend wird ein Makro aufgezeichnet, das den markierten Zellen diese Formatvorlage zuweist. Dann wird in die Symbolleiste für den Schnellzugriff ein neues Symbol eingefügt und diesem Makro zugewiesen. Damit besteht die Möglichkeit, die zuvor markierten Zellen durch einen Klick auf das neue Symbol mit der definierten Formatvorlage zu formatieren.



Tip

Alle Beispiele dieses Kapitels befinden sich natürlich auch in den Download-Dateien zum Buch (Verzeichnis 01 für das erste Kapitel).

Bevor Sie beginnen

Vorweg einige Tipps, die Ihnen das Leben und die Programmierung mit Excel erleichtern:

- Machen Sie die Befehlsregisterkarte **ENTWICKLERTOOLS** im Menüband sichtbar. Dazu öffnen Sie die Registerkarte **DATEI** und wählen **OPTIONEN**. Klicken Sie links im Dialogfeld auf **MENÜBAND ANPASSEN**, schalten Sie im rechten Listenfeld das Kontrollkästchen vor **ENTWICKLERTOOLS** ein und schließen Sie das Dialogfeld mit **OK**.
- Die nun sichtbare Befehlsregisterkarte stellt Ihnen alle Werkzeuge für die Aufzeichnung und Bearbeitung von Makros, den Entwurf von Formularen sowie die XML-Programmierung zur Verfügung.

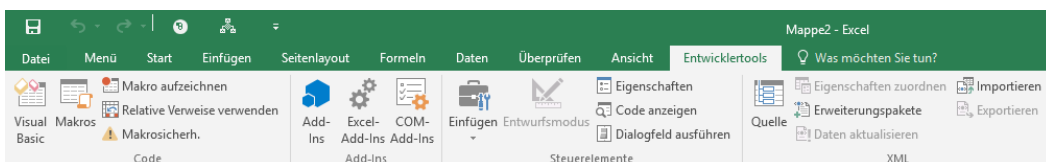


BILD 1.1: Die Registerkarte Entwicklertools

- Standardmäßig deaktiviert Excel sämtliche Makros. Beim Öffnen eines „makrohaltigen“ Dokuments erscheint unterhalb des Menübands eine Sicherheitswarnung, über deren **INHALT-AKTIVIEREN**-Schaltfläche Sie die Makroausführung ausdrücklich „freischalten“ müssen. Was als Schutz vor möglichen Makroviren gedacht ist, kann die Entwicklung eigener Makros aber ungemein behindern.
- Wenn Sie möchten, dass Excel alle Ihre Makros ohne Sicherheitswarnung aktiviert, stellen Sie die Makrosicherheitsstufe vorübergehend für die Zeit der Makroentwicklung auf das niedrigste Level ein. Dazu wählen Sie **ENTWICKLERTOOLS | MAKROSICHERH.**, aktivieren das Optionsfeld **ALLE MAKROS AKTIVIEREN** und wählen **OK**. Sofern Sie einen Virens Scanner installiert haben (was dringend zu empfehlen ist), überprüft Excel nun immer noch alle Dokumente auf mögliche Makroviren, sodass Sie kein allzu großes Risiko eingehen. (Weitere Informationen zur Sicherheit von Makros folgen in Abschnitt 4.7.)
- In der VBA-Entwicklungsumgebung (die Sie mit **Alt+F11** erreichen) sind einige Optionen verquer voreingestellt. Den Optionsdialog erreichen Sie dort mit **EXTRAS | OPTIONEN**.
- Dort deaktivieren Sie **AUTOMATISCHE SYNTAXÜBERPRÜFUNG**. (Die Syntax wird weiterhin überprüft, fehlerhafte Zeilen werden rot markiert. Es entfällt nur die lästige Fehlermeldung samt Piepston.)
- Dann aktivieren Sie die Option **VARIABLENDEKLARATION ERFORDERLICH**. (Eine ausführliche Begründung folgt in Abschnitt 4.1.)
- Im Dialogblatt **ALLGEMEIN** deaktivieren Sie die Option **KOMPILIEREN | BEI BEDARF** (siehe Abschnitt 3.2).

► Schritt 1: Definition der Formatvorlage „Result“

Zellen, die das (Zwischen-)Ergebnis einer Berechnung beinhalten, sollen folgendermaßen aussehen:

- Schrift: Arial, 14 Punkt, fett
- Rahmen: doppelte Linie unten
- Zahlenformat: zwei Dezimalstellen

Wenn Sie möchten, können Sie natürlich auch andere Formatierungsmerkmale auswählen. Es geht in diesem Beispiel nur darum, ein neues, eindeutig erkennbares Format zu definieren.

Zur Definition der Formatvorlage öffnen Sie eine neue Arbeitsmappe mit DATEI | NEU | LEERE ARBEITSMAPPE, schreiben in eine beliebige Zelle eine Zahl und formatieren diese Zelle anschließend mit den oben aufgezählten Merkmalen. Anschließend führen Sie das Kommando START | ZELLENFORMATVORLAGEN | NEUE ZELLENFORMATVORLAGE aus. Im nun erscheinenden Dialog geben Sie als Formatvorlagenname „Result“ ein und klicken OK an.

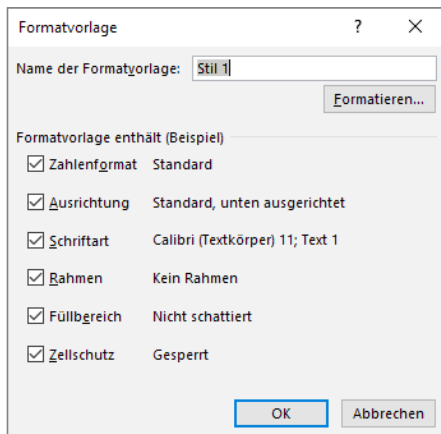


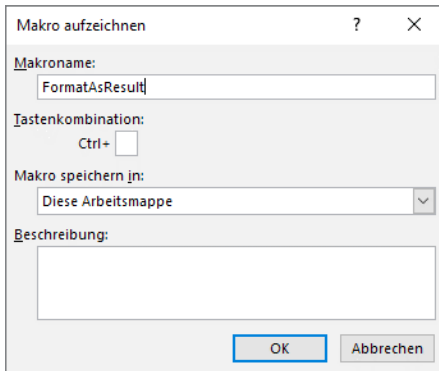
BILD 1.2:

Die Definition einer neuen Formatvorlage

Geben Sie in Ihrer Tabelle in einer beliebigen Zelle eine weitere Zahl ein, und testen Sie die neue Formatvorlage: Führen Sie START | ZELLENFORMATVORLAGEN aus, und wählen Sie als Vorlage „Result“. Die zweite Zelle sollte nun ebenso formatiert sein wie die erste.

► Schritt 2: Makro aufzeichnen

Die Arbeitsschritte, die Sie gerade zur Formatierung einer Testzelle ausgeführt haben, sollen in Zukunft automatisch von einem Makro erledigt werden. Dazu müssen diese Arbeitsschritte in einem Makro aufgezeichnet werden. Bewegen Sie den Zellzeiger in eine neue Zelle, und geben Sie dort (um das Ergebnis zu kontrollieren) eine Zahl ein. Schließen Sie die Eingabe mit Return ab, und bewegen Sie den Zellzeiger gegebenenfalls zurück in die gerade veränderte Zelle. Wählen Sie in Excel (nicht in der VBA-Entwicklungsumgebung) das Kommando ENTWICKLERTOOLS | MAKRO AUFZEICHNEN, und geben Sie als Makronamen „Format-AsResult“ ein.

**BILD 1.3:**

Der Dialog zum Aufzeichnen von Makros

Sobald Sie das Dialogfeld mit OK schließen, beginnt Excel mit der Aufzeichnung des neuen Makros. Formatieren Sie die gerade aktive Zelle mit dem Druckformat „Result“ (ebenso wie am Ende von Schritt 1, als Sie die Formatvorlage getestet haben). Beenden Sie die Makroaufzeichnung mit **ENTWICKLERTOOLS | AUFZEICHNUNG BEENDEN** oder durch das Anklicken des kleinen Quadrats, das seit Beginn der Aufzeichnung in der Statuszeile von Excel am unteren Bildschirmrand angezeigt wird.

Jetzt können Sie sich das fertige Makro ansehen, indem Sie mit **Alt+F11** in die Entwicklungsumgebung wechseln und dort „Modul1“ ansehen. (Dieses Modul wurde im Zuge der Makroaufzeichnung automatisch erzeugt. Wenn „Modul1“ schon existiert, legt Excel ein neues Modul mit dem Namen „Modul2“ an.) Das neue Modul sollte folgendermaßen aussehen:

```
Sub FormatAsResult()
'
' FormatAsResult Makro
'
    Selection.Style = "result"
End Sub
```

Jetzt sollten Sie das neue Makro noch testen: Wechseln Sie wieder zurück in das Tabellenblatt, geben Sie in einer beliebigen Zelle eine weitere Zahl ein, und schließen Sie die Eingabe mit Return ab. Wählen Sie mit dem Kommando **ANSICHT | MAKROS | MAKROS ANZEIGEN** das Makro „FormatAsResult“ aus. Excel führt Ihr Makro aus, die Zelle sollte anschließend in dem nun schon vertrauten Ergebnisformat erscheinen.

► Schritt 3: Definition eines neuen Symbols

Damit das Makro bequemer aufgerufen werden kann, soll jetzt ein neues Symbol in die Symbolleiste für den Schnellzugriff (die Sie links in der Titelleiste des Excel-Programmfensters finden) eingefügt werden. Dazu klicken Sie mit der rechten Maustaste auf die Symbolleiste und wählen den Befehl **PASSEN SIE DIE SYMBOLLEISTE FÜR DEN SCHNELLZUGRIFF AN**. Stellen Sie das Listenfeld **BEFEHLE AUSWÄHLEN** auf „Makros“ ein, markieren Sie „FormatAsResult“ in der Liste darunter, und klicken Sie auf **HINZUFÜGEN >>**. Der Name Ihres aufgezeichneten Makros sollte nun in dem rechten Listenfeld erscheinen.

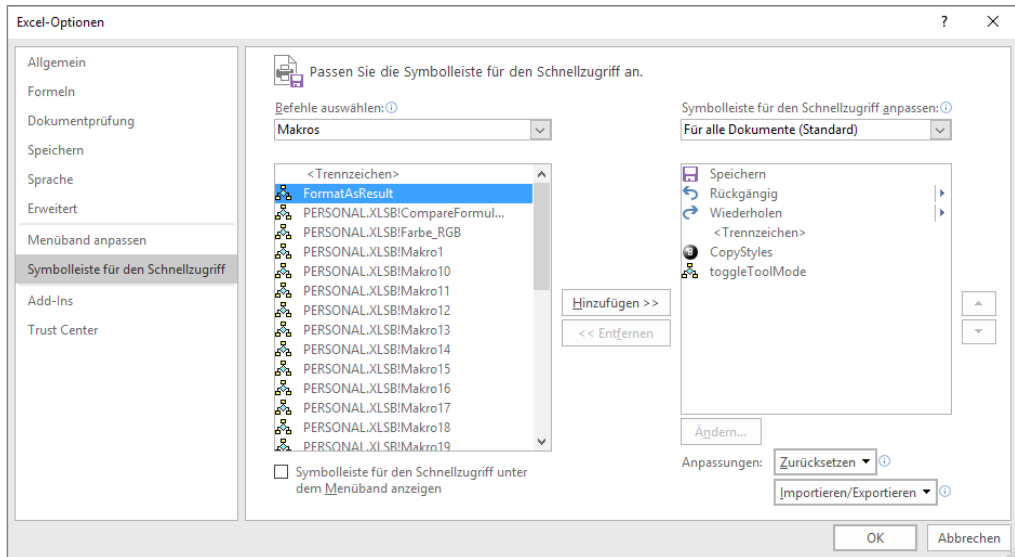
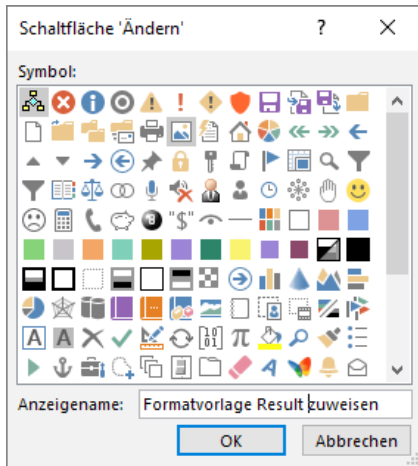


BILD 1.4: Der Dialog zur Anpassung der Symbolleiste für den Schnellzugriff

Würden Sie das ANPASSEN-Dialogfeld jetzt per OK-Schaltfläche schließen, wäre das neue Makrostartsymbol bereits in der Symbolleiste für den Schnellzugriff sichtbar. Beim „Überfahren“ des Symbols mit dem Mauszeiger würde allerdings nur der vielleicht etwas kryptische Makroname „FormatAsResult“ als sogenannter Tooltip-Text erscheinen. Sie sollten diesen Text gegen eine aussagekräftigere Bezeichnung ersetzen, welche die Funktion des Makros auch anderen Anwendern offenbart.

Und wo Sie schon gerade beim Anpassen sind, sollten Sie dem neuen Symbol auch gleich eine individuelle Grafik zuweisen, die das Standardbild, das voreinstellungsgemäß für alle Makros Verwendung findet, ersetzt.

Um beides zu bewerkstelligen, klicken Sie im immer noch geöffneten ANPASSEN-Dialog im rechten Listenfeld auf den Makronamen „FormatAsResult“ und anschließend auf die Schaltfläche ÄNDERN. Markieren Sie nun ein Symbol Ihrer Wahl, und ändern Sie den Anzeigenamen im Textfeld darunter, beispielsweise in „Formatvorlage Result zuweisen“. Nach einem Klick auf OK erscheint das neue Symbol in seiner endgültigen Fassung in der Symbolleiste für den Schnellzugriff.

**BILD 1.5:**

Das neue Symbol erhält ein individuelles Bild und einen Anzeigenamen zugewiesen.

Im Unterschied zu früheren Excel-Versionen gibt es keine Möglichkeit, einem Symbol eine selbst erstellte Grafik zuzuweisen. Die Auswahl ist somit auf die angebotenen Standard-Icons beschränkt.

Das Symbol wieder aus der Symbolleiste entfernen

Wenn Sie das Beispiel beendet haben, wird Ihnen das neue Symbol vermutlich im Weg stehen. Um es wieder zu entfernen, klicken Sie erneut mit der rechten Maustaste auf die Symbolleiste für den Schnellzugriff und wählen den Befehl **PASSEN SIE DIE SYMBOLLEISTE FÜR DEN SCHNELLZUGRIFF AN**. Markieren Sie im rechten Listefeld den Makronamen „Format-AsResult“, und klicken Sie auf **<< ENTFERNEN**. Nach einem Klick auf **OK** befindet sich die Symbolleiste wieder in ihrem ursprünglichen Zustand.

Anmerkungen für Fortgeschrittene

Obwohl das Beispiel recht einfach war, wirft es interessante Probleme auf. Es ist nicht ohne Weiteres möglich, die gesamten Informationen dieses Beispiels (also die Definition der Formatvorlage, des neuen Symbols und des Makros) in einer Makroarbeitsmappe so zu speichern, dass ein anderer Excel-Anwender das neue Symbol zur Formatierung von Zellen in seiner eigenen Tabelle verwenden kann. Selbst wenn Sie den Button nur dazu verwenden möchten, um eine Zelle in einer anderen Arbeitsmappe zu formatieren, kommt es zu einer Fehlermeldung. Die Gründe:

- Formatvorlagen gelten nur für eine Arbeitsmappe und können nicht problemlos in einer anderen Arbeitsmappe verwendet werden. (Das Makro könnte natürlich so erweitert werden, dass es die Formatvorlage zuerst in die jeweilige Arbeitsmappe kopiert. Das übersteigt aber den Charakter eines Einführungsbeispiels.)
- Die Anpassung der Symbolleiste für den Schnellzugriff wird in einer Datei gespeichert, die zu den persönlichen Konfigurationsdateien von Excel gehört. (Details zu den Speicherorten dieser Dateien finden Sie in Abschnitt 5.9.)

Das neu definierte Symbol steht daher nur zur Verfügung, wenn Sie sich unter Ihrem Namen anmelden. Andere Benutzer desselben Rechners (mit eigenen Login-Namen) können das neue Symbol somit nicht verwenden.

■ 1.4 Beispiel: Makro zur Eingabeerleichterung

Bei der Eingabe tabellarischer Daten kommt es häufig vor, dass in einer Zelle derselbe Wert bzw. derselbe Text eingegeben werden muss, der bereits in der unmittelbar darüber stehenden Zelle zu finden ist. Excel stellt zwar verschiedene Möglichkeiten zur Verfügung, mit denen Sie die Zelle nach unten kopieren können, alle Varianten setzen aber entweder die Verwendung der Maus voraus (das stört bei einer reinen Tastatureingabe erheblich) oder erfordern umständliche Cursorbewegungen. Es liegt nahe, hierfür wiederum ein Makro aufzuzeichnen, das mit einer einfachen Tastenkombination (z. B. Strg+K zum Kopieren) aufgerufen werden kann.

Vorbereitungsarbeiten

Excel kann bei der Makroaufzeichnung zwischen absoluten und relativen Zellbezügen unterscheiden:

- Normalerweise gilt der absolute Modus. Wenn Sie den Zellzeiger während der Aufzeichnung von B2 nach D4 bewegen, resultiert daraus das Kommando `Range("D4").Select`.
- Im relativen Modus würde das Kommando dagegen folgendermaßen aussehen: `ActiveCell.Offset(2, 2).Range("A1").Select`. Mit `ActiveCell.Offset(2,2)` wird also die Zelle zwei Zeilen unterhalb und zwei Spalten neben der gerade aktiven Zelle angesprochen. `Range("A1")` bezieht sich auf diese neue Adresse.

Unterschiede zwischen diesen beiden Varianten ergeben sich erst bei der Ausführung der Makros. Im ersten Fall wird immer die Zelle D4 bearbeitet, ganz egal, wo sich der Zellzeiger vorher befindet. Im zweiten Fall wird die zu bearbeitende Zelle relativ zur aktuellen Zelle ausgewählt.

Zur Umschaltung zwischen relativer und absoluter Aufzeichnung verwenden Sie den Befehl **RELATIVE VERWEISE VERWENDEN** in der Registerkarte **ENTWICKLERTOOLS**. Wenn der Befehl als gedrückter Button angezeigt wird, gilt die relative Aufzeichnung, sonst die absolute. Der Modus kann auch während der Aufzeichnung umgeschaltet werden. Für das Makro dieses Abschnitts sind relative Bezüge erforderlich.

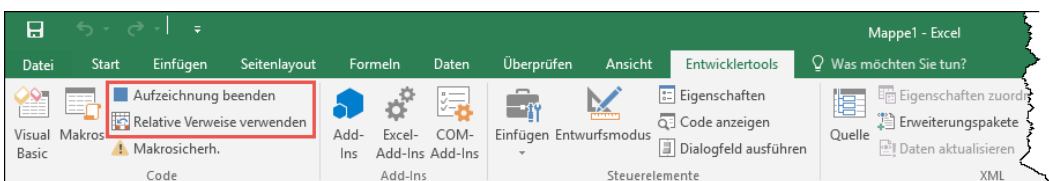


BILD 1.6: Zwei Befehle ermöglichen die absolute und relative Makroaufzeichnung.

Makroaufzeichnung

Bevor Sie mit der Aufzeichnung des Makros beginnen, bereiten Sie die Tabelle vor: Geben Sie in einer Zelle irgendeinen Text ein, und bewegen Sie den Zellzeiger anschließend in die Zelle unmittelbar darunter.

Starten Sie die Aufzeichnung mit **ENTWICKLERTOOLS | MAKRO AUFZEICHNEN**, und geben Sie dabei als Makronamen *CopyFromCellAbove*, als Shortcut (das ist die Tastaturabkürzung) **Strg+K** und als Zielort „Persönliche Makroarbeitsmappe“ an. Anschließend stellen Sie den Aufzeichnungsmodus auf **RELATIVE VERWEISE VERWENDEN**, falls dies nicht bereits der Fall ist. Das ist bei diesem Makro erforderlich, weil es an jeder beliebigen Stelle in der Tabelle funktionieren soll (und immer die – relativ zum Zellzeiger – darüber liegende Zelle kopieren soll).

Während die Aufzeichnung läuft, drücken Sie **Shift+↑**, um die aktuelle und die darüber liegende Zelle zu markieren. Anschließend wechseln Sie auf die Befehlsregisterkarte **START**, öffnen das nicht beschriftete **FÜLLBEREICH**-Menü im rechten Teil der Registerkarte und wählen **UNTEN**, um den Inhalt der ersten Zelle in die darunterliegende zu kopieren. Mit → schließlich bewegen Sie den Zellzeiger in die nächste Zelle nach rechts, wo die nächste Eingabe erfolgen kann. Beenden Sie die Aufzeichnung mit **ENTWICKLERTOOLS | AUFZEICHNUNG BEENDEN**.

In der Persönlichen Arbeitsmappe sollte sich jetzt folgendes Visual-Basic-Makro befinden:

```
' Beispiel 01\shortcut.xlsm
Sub CopyFromCellAbove()
    ActiveCell.Offset(-1, 0).Range("A1:A2").Select
    ActiveCell.Activate
    Selection.FillDown
    ActiveCell.Offset(0, 1).Range("A1").Select
End Sub
```

Wenn Sie das Makro nun ausprobieren, werden Sie feststellen, dass es zwar prinzipiell funktioniert, dass sich der Zellzeiger anschließend aber nicht wie bei der Aufzeichnung rechts von der Startzelle befindet, sondern eine Zeile weiter oben. Hier gibt es also einen kleinen Widerspruch zwischen den aufgezeichneten Kommandos und dem resultierenden Code (d.h., die automatische Makroaufzeichnung hat nicht ganz perfekt funktioniert). Sie können das Problem beseitigen, indem Sie den ersten *Offset*-Wert in der letzten Zeile des Makros folgendermaßen ändern:

```
ActiveCell.Offset(1, 1).Range("A1").Select
```



Anmerkung

Wenn Sie bei der Aufzeichnung vergessen, das Tastenkürzel **Strg+K** anzugeben, können Sie einem vorhandenen Makro auch nachträglich ein Kürzel zuweisen. Dazu führen Sie in Excel (nicht in der Entwicklungsumgebung) **ENTWICKLERTOOLS | MAKROS** aus, wählen das Makro aus und stellen das Kürzel mit **OPTIONEN** ein.

■ 1.5 Beispiel: Einfache Literaturdatenbank

Das folgende Beispiel ist eine schon ziemlich konkrete (wenn auch noch immer sehr einfache) Anwendung. Es befindet sich bei den Download-Dateien zum Buch unter dem Dateinamen *Books.xlsm*.

Die Arbeitsmappe (oder „das Programm“) ermöglicht die Verwaltung von Büchern, beispielsweise für eine kleine Bibliothek. Die Liste der Bücher kann beliebig erweitert, nach Autoren oder Titeln sortiert, nach verschiedenen Kriterien selektiert (z.B. nur Computerbücher), nach Begriffen durchsucht werden etc. Die Bedienung der Anwendung ist durch einige Buttons vereinfacht, sodass zum Suchen eines Buchs oder zur Erweiterung der Datenbank kaum Excel-spezifische Kenntnisse erforderlich sind.

► Schritt 1: Datenbank einrichten, Fenstergestaltung

Die Erstellung dieser Anwendung beginnt damit, dass Sie die Daten einiger Bücher eingeben und die Tabelle in etwa nach dem Vorbild der Abbildung 1.7 gestalten. Das hat vorläufig noch nichts mit Makroprogrammierung zu tun, es handelt sich lediglich um den ganz normalen Aufbau einer neuen Excel-Tabelle. Die Buttons und Filterpfeile sollten Sie vorläufig ignorieren.

(Wenn Sie möchten, können Sie genauso gut eine Adressdatenbank, eine Schülerkartei oder was auch immer anlegen. Sie müssen sich durchaus nicht exakt an die Vorlage halten! Lernen werden Sie umso mehr, je kreativer und eigenständiger Sie vorgehen.)

	A	B	C	D	E
	Autor	Titel	Kategorie	Erscheinungsjahr	Verlag
3	Preußler, Otfried	Alles vom Räuber Hotzenplotz	K	1988	dtv
4	Quirogas, Eduard	Auf fremder Erde	K	1991	Ravensburger
5	Roos, Otto	Buchführung mit Excel	C	1993	Addison Wesley
6	Sanzara, Rahel	Das verlorene Kind	B	1983	suhrkamp
7	Pohl, Peter	Der Regenbogen hat nur acht Farben	K	1993	Hanser
8	Satre, Jean-Paul	Die Kindheit eines Chefs	B	1985	rororo
9	Mitterer, Felix	Die Piefke-Saga	B	1991	Haymon
10	Brodsky, Joseph	Erinnerungen an Leningrad	B	1990	Fischer TB
11	Fried, Erich	Es ist was es ist	G	1983	Wagenbach
12	Schluiferer, Sepp	Fern von Europa	B	1909	Edition Löwenzahn
13	Maeder, Roman	Informatik für Mathematiker und Naturwissenschaftler	C	1993	Addison Wesley
14	Preußler, Otfried	Krabat	K	1981	Thienemann
15	Kofler, Michael	Linux, 4. Auflage	C	1999	Addison Wesley
16	Kofler, Michael	Linux, 6. Auflage	C	2001	Addison-Wesley
17	Kofler, Michael	Maple V, Release 4	C	1996	Addison Wesley

BILD 1.7: Eine einfache Datenbankanwendung

Einige Hinweise zur Formatierung der Tabelle: Alle Zellen der Tabelle wurden vertikal nach oben ausgerichtet (START | ZAHL | ZELLEN FORMATIEREN | AUSRICHTUNG). Bei den Zellen der Titelspalte wurde außerdem das Attribut *Zeilenumbruch* aktiviert, sodass längere Titel automatisch über mehrere Zeilen verteilt werden. Bei den beiden obersten Zeilen wurde die Zeilenhöhe deutlich vergrößert. Die gesamte zweite Zeile wurde mit der Hintergrundfarbe Hellgrau formatiert (START | ZAHL | ZELLEN FORMATIEREN | AUSFÜLLEN).

In Zelle C2 (Inhalt: Kategorie-Überschrift) wurde ein Kommentar gespeichert. (Kommentare werden mit ÜBERPRÜFEN | NEUER KOMMENTAR eingegeben. Seit Excel 7 werden Kommentare automatisch angezeigt, sobald die Maus über eine Zelle mit einem Kommentar bewegt wird.) Der Kommentar in Zelle C2 wird dazu verwendet, den Kategorie-Code zu erklären: B ... Belletristik, C ... Computerliteratur etc. Zellen, zu denen ein Kommentar gespeichert ist, werden in der rechten oberen Ecke mit einem roten Punkt markiert. Falls dieser Punkt bei Ihnen nicht erscheint, wählen Sie DATEI | OPTIONEN | ERWEITERT | ANZEIGE, und aktivieren Sie die Option NUR INDIKATOREN, UND KOMMENTARE NUR BEIM DARAUFGZEIGEN.

Einige Tipps zur Gestaltung des Fensters: Das Fenster wurde so fixiert, dass im oberen Fensterbereich die Überschrift der Datenbank (zwei ziemlich hohe Zeilen) dauerhaft zu sehen ist (Zelle A3 markieren, ANSICHT | FENSTER FIXIEREN | FENSTER FIXIEREN wählen). Mit SEITENLAYOUT | BLATTOPTIONEN wurde die Anzeige der Gitternetzlinien abgeschaltet.

Dass es sich bei der Tabelle um eine Datenbank handelt, brauchen Sie Excel nicht mitzuteilen. (Damit Excel die Datenbank erkennt, müssen Sie lediglich den Zellzeiger irgendwo in die Datenbank bewegen. Als „Datenbank“ gilt in Excel einfach jeder zusammenhängende Zellbereich.)

Sie können daher sofort die Datenbankkommandos ausprobieren, etwa zum Sortieren der Tabelle nach einem beliebigen Kriterium (Autor, Titel, Erscheinungsjahr etc.). Führen Sie einfach DATEN | SORTIEREN aus.



Tipp

Das Datenbankkommando zum Aufruf einer Eingabemaske (früher: DATEN | MASKE) steht in Excel 2016 standardmäßig nicht mehr zur Verfügung. Sie können es allerdings der Symbolleiste für den Schnellzugriff hinzufügen, indem Sie diese mit der rechten Maustaste anklicken, den Befehl PASSEN SIE DIE SYMBOLLEISTE FÜR DEN SCHNELLZUGRIFF AN wählen, das Listenfeld BEFEHLE AUSWÄHLEN auf „Alle Befehle“ einstellen und dann den Befehl „Maske...“ per HINZUFÜGEN-Schaltfläche in die Symbolleiste integrieren. Tipp im Tipp: Um mehr Platz für weitere Symbole zu schaffen, können Sie die Symbolleiste für den Schnellzugriff per Kontextmenübefehl unter dem Menüband anzeigen lassen.

► Schritt 2: Die Datenbank mit Filtern ausstatten

Mit dem Kommando DATEN | FILTERN werden die kleinen Filterpfeile in den Überschriftenzellen der Tabelle angezeigt. Wenn Sie diese Pfeile mit der Maus anklicken, können Sie Filterkriterien auswählen, z.B. einen bestimmten Verlag, ein Erscheinungsjahr etc. In der Datenbank werden dann nur noch jene Daten angezeigt, die dieses Kriterium erfüllen. Zur Kennzeichnung, dass nicht alle Daten sichtbar sind, zeigt Excel jetzt ein Filtersymbol auf

der Schaltfläche an. Es können mehrere Filterkriterien kombiniert werden (beispielsweise alle Bücher des Verlags *X* aus dem Jahr *Y*). Sie können sogar eigene Kriterien aufstellen, etwa um alle Bücher anzuzeigen, die zwischen 1980 und 1990 erschienen sind (TEXTFILTER | BENUTZERDEFINIERTER FILTER).

► Schritt 3: Buttons und Makros

Als erfahrener Excel-Anwender haben Sie vermutlich keine Probleme, die Datenbank im aktuellen Zustand zu bedienen. Sie können die Daten nach beliebigen Kriterien sortieren, Daten eingeben und verändern etc. Wenn Sie aber möchten, dass auch ein vollkommener Excel-Laie mit dieser Datenbank umgehen kann, müssen Sie die Bedienung noch ein wenig vereinfachen. Im vorliegenden Beispiel wurden dazu einige Buttons in die Tabelle eingefügt, mit denen die wichtigsten Funktionen ohne langes Suchen im Menüband ausgeführt werden können.

Zum Einfügen von Buttons aktivieren Sie die Registerkarte ENTWICKLERTOOLS. Anschließend klicken Sie auf EINFÜGEN, wählen das Befehlsschaltflächensymbol unter „ActiveX-Steuerelemente“ und fügen den Button mit der Maus in das Tabellenblatt ein. (Dabei wird automatisch der Entwurfsmodus aktiviert, der eine weitere Bearbeitung des Buttons ermöglicht.)

Nun folgt die Formatierung des Buttons: Per Kontextmenükommando BEFEHLSCHALTFLÄCHE-OBJEKT | BEARBEITEN können Sie die Beschriftung ändern. Strg+Return beginnt eine neue Zeile, Esc schließt die Eingabe ab.

Alle anderen Eigenschaften werden mit einem eigenen Fenster (siehe Abbildung 1.8) eingestellt, das ebenfalls per Kontextmenü oder mit dem EIGENSCHAFTEN-Befehl im Menüband aufgerufen wird. In diesem Eigenschaftenfenster sollten Sie die folgenden Einstellungen vornehmen:

- *Name*: Geben Sie dem Steuerelement einen aussagekräftigen Namen, etwa *btnSort* für den Button zum Sortieren.
- *Font*: Vergrößern Sie die Schriftart auf 10 Punkt, und wählen Sie das Attribut fett.
- *ForeColor*: Wenn Sie bunte Buttons mögen, stellen Sie eine andere Textfarbe ein.
- *TakeFocusOnClick*: Stellen Sie diese Eigenschaft auf *False*, damit der VBA-Code später korrekt verarbeitet wird (siehe auch Abschnitt 7.5).

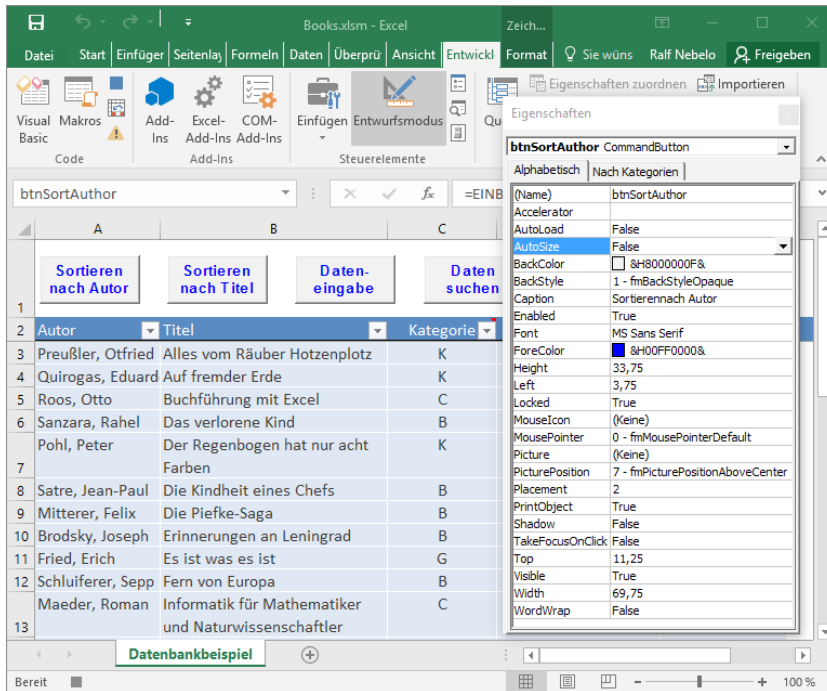


BILD 1.8: Links das Tabellenfenster, rechts das Eigenschaftenfenster mit den Einstellungen des ausgewählten Buttons



Tip

Sie können etwas Zeit sparen, wenn Sie diese Formatierungsschritte nur für den ersten Button durchführen. Anschließend kopieren Sie den Button mehrmals, indem Sie ihn mit der Maus bei gedrückter Strg-Taste verschieben. Wenn Sie zusätzlich Shift drücken, bleibt außerdem die horizontale Position gleich, sodass die Steuerelemente zueinander ausgerichtet erscheinen. Anschließend müssen Sie nur noch die Beschriftung und den Steuerelementnamen einstellen.

Jetzt müssen Sie die Buttons noch mit Programmcode verbinden. Ein Doppelklick auf den Button fügt im Modul *Tabelle1* eine Schablone für die Ereignisprozedur ein, die beim Anklicken des Buttons später automatisch ausgeführt wird. Der Name der Prozedur setzt sich aus dem Steuerelementnamen (etwa *btnSortAuthor*) und dem Ereignis (meist *Click*) zusammen:

```
' Beispiel 01\books.xlsm, Modul Tabelle1
Private Sub btnSortAuthor_Click()

End Sub
```

Zur Aufzeichnung des Programmcodes gehen Sie wie in den vorangegangenen Beispielen vor. Die resultierenden Anweisungen übertragen Sie anschließend mit KOPIEREN und EIN-

FÜGEN vom Aufzeichnungsmodul in die Codeschablone. Falls Sie zuletzt das Beispiel des vorherigen Abschnitts abgearbeitet haben, ist noch der Modus „Relative Verweise verwenden“ aktiv. Sie sollten diesen Modus bei Beginn der Aufzeichnung wieder deaktivieren.

Nun zum Inhalt der Makros: Für die beiden Sortiermakros klicken Sie zuerst die Zelle A2 an, führen anschließend DATEN | SORTIEREN aus und geben im Dialog das gewünschte Sortierkriterium (Autoren oder Titel) an. Bevor Sie das Makro ALLE DATEN ANZEIGEN aufzeichnen können, müssen Sie irgendein Filterkriterium auswählen (z.B. alle Bücher anzeigen, die 1993 erschienen sind). Das Kommando DATEN | LÖSCHEN steht nämlich nur zur Verfügung, wenn mindestens ein Filterkriterium aktiv ist. Für das *Speichern*-Makro führen Sie einfach das Kommando DATEI | SPEICHERN aus. Die Makros *Dateneingabe* und *Daten suchen* müssen Sie im Modul über die Tastatur eingeben (Code siehe unten). Die Makros sollten schließlich folgendermaßen aussehen:

```
' Beispiel 01\books.xlsm, „Tabelle1“
' Sortieren nach Autorennamen
Private Sub btnSortAuthor_Click()
    Range("A2").Select
    Selection.Sort Key1:=Range("A3"), Order1:= _
        xlAscending, Header:=xlGuess, OrderCustom:=1, _
        MatchCase:=False, Orientation:=xlTopToBottom
End Sub
' Sortieren nach Titeln
Private Sub btnSortTitle_Click()
    Range("A2").Select
    Selection.Sort Key1:=Range("B3"), Order1:= _
        xlAscending, Header:=xlGuess, OrderCustom:=1, _
        MatchCase:=False, Orientation:=xlTopToBottom
End Sub
' Datenbankmaske anzeigen, „Neu“-Button anklicken
Private Sub btnInput_Click()
    Range("A2").Select
    SendKeys "%n"
    ActiveSheet.ShowDataForm
End Sub
' Dialog zum Suchen anzeigen
Private Sub btnFind_Click()
    SendKeys "^f"
End Sub
' alle Datensätze anzeigen
Private Sub btnShowAll_Click()
    On Error Resume Next
    ActiveSheet.ShowAllData
End Sub
' Speichern
Private Sub btnSave_Click()
    ActiveWorkbook.Save
End Sub
```


**Tip**

Zum Ausprobieren der Buttons müssen Sie den Entwurfsmodus für die Steuerelemente über den Befehl in der Registerkarte ENTWICKLERTOOLS deaktivieren.

Anmerkungen für Fortgeschrittene

Das Anklicken der Zelle A2 beim Aufzeichnen der Makros (es könnte auch eine beliebige andere Zelle des Datenbankbereichs sein) ist notwendig, weil die Datenbankkommandos nur funktionieren, wenn der Zellzeiger sich im Datenbankbereich befindet. Da die Buttons später auch dann angeklickt werden können, wenn der Zellzeiger an anderer Stelle steht, muss er am Beginn des Makros explizit in den Datenbankbereich gestellt werden.

In das Makro *btnInput_Click* muss nach der Makroaufzeichnung das *SendKeys*-Kommando eingefügt werden. Es simuliert die Tastatureingabe Alt+N, durch die im Dialog der Datenbankmaske der Button NEU ausgewählt wird. Damit wird verhindert, dass der Anwender irrtümlich einen bereits bestehenden Eintrag überschreibt. (Wegen der nicht ganz einleuchtenden Bedienung der Datenbankmaske passiert das beim ersten Mal fast zwangsläufig.)

Die Anordnung der Kommandos *SendKeys* und *ShowDataForm* erscheint zugegebenermaßen unlogisch. Eigentlich sollte man meinen, dass zuerst der Dialog geöffnet und erst anschließend die Tastatureingabe simuliert werden muss. Das Kommando *SendKeys* bewirkt aber lediglich, dass die Tastenkombination Alt+N in einen Tastaturpuffer eingetragen und von dort (irgendwann) vom Windows-System weiterverarbeitet wird. Würde *SendKeys* unter *ShowDataForm* im Makro stehen, würde Excel mit der Ausführung von *SendKeys* warten, bis die Eingaben in der Datenbankmaske ausgeführt sind – und dann ist es natürlich zu spät.

Die Methode *ShowDataForm* setzt voraus, dass die Datenbank im Zellbereich A1:B2 anfängt. Die aktuelle Position des Zellzeigers ist egal. Wenn die Datenbank an einer anderen Position anfängt, müssen Sie den gesamten Zellbereich mit dem Namen *datenbank* benennen (FORMELN | NAMEN DEFINIEREN). Weitere Informationen gibt es auf der folgenden Internetseite [Link 2]:

<http://support.microsoft.com/kb/110462/de>

Das Makro *btnFind_Click* verwendet ebenfalls *SendKeys* zum Aufruf des Suchen-Dialogs. Es wäre zwar auch möglich, den Dialog mit *Dialogs(xlDialogFormulaFind).Show* anzuzeigen, es ist aber nicht möglich, tatsächlich Daten zu finden. (Die Suche beschränkt sich aus unerklärlichen Gründen auf die gerade aktuelle Zelle.) Dieses Problem besteht übrigens schon seit Excel 5!

Im Makro *btnShowAll_Click* wird Ihnen vielleicht die Anweisung *On Error Resume Next* auffallen. Diese Anweisung bewirkt, dass das Makro auch dann ohne Fehlermeldung in der nächsten Zeile fortgesetzt wird, wenn ein Fehler auftritt. In dem Makro kann es sehr leicht zu einem Fehler kommen: nämlich immer dann, wenn der Anwender den Button ALLE DATEN ANZEIGEN anklickt, obwohl in dem Moment gar kein Filterkriterium aktiv ist.

■ 1.6 Beispiel: Formular zur Berechnung der Verzinsung von Spareinlagen

Das nächste Beispiel dieses Kapitels beweist, dass die Gestaltung vorgefertigter Anwendungen nicht zwangsläufig mit Programmieren verbunden ist. In der in Abbildung 1.9 dargestellten Tabelle können Sie im hellblau unterlegten Bereich vier Parameter eingeben: die jährliche Verzinsung der Spareinlagen, die monatlichen Spareinlagen, den ersten Einzahlungstag und die Laufzeit des Sparvertrags. Die Tabelle ist so dimensioniert, dass die Laufzeit maximal sechs Jahre betragen kann.

Verzinsung monatlicher Einlagen

Jährliche Verzinsung: 2,25%

Monatlicher Sparbetrag: 200,00 €

Erste Einzahlung: 08.12.2015

Vertragsdauer in Jahren (max. 6): 2

Letzte Einzahlung: 08.11.2017

Vertragsende: 08.12.2017

Monatliche Verzinsung: 0,19%

Zinsgutschrift: 112,96 €

Endguthaben: 4.912,96 €

← Geben Sie in den blau markierten Zellen neue Werte ein; die wird automatisch neu berechnet.

Datum	Einzahlung	Zinsen	Guthaben	Datum	Einzahlung	Zinsen	Guthaben
08.12.2015	200,00	0,00	200,00				
08.01.2016	200,00	0,37	400,37				
08.02.2016	200,00	0,74	601,11				
08.03.2016	200,00	1,12	802,23				
08.04.2016	200,00	1,49	1003,72				
08.05.2016	200,00	1,86	1205,58				
08.06.2016	200,00	2,24	1407,82				
08.07.2016	200,00	2,61	1610,43				
08.08.2016	200,00	2,99	1813,42				

BILD 1.9: Verzinsung monatlicher Spareinlagen

Aus den Eingaben berechnet Excel selbstständig den Termin der letzten Einzahlung, das Vertragsende, die monatliche Verzinsung, die im Verlauf der Vertragszeit erreichte Zinsgutschrift sowie das Gesamtguthaben nach Ende der Laufzeit. Außerdem generiert Excel eine Tabelle mit den monatlichen Zinsgutschriften und Guthaben, sodass leicht festgestellt werden kann, wie groß das Guthaben zu einem beliebigen Zeitpunkt innerhalb der Laufzeit ist.

Die Tabelle eignet sich beispielsweise als Grundlage (und Werbemittel) in einem Bankinstitut, das die Kunden von der Sinnfälligkeit eines Sparvertrags überzeugen möchte. Das Erstellen einer Tabelle nach den Vorstellungen des Kunden erfolgt in Sekunden. Die Tabelle kann anschließend in einer ansprechenden Form ausgedruckt werden.



Verweis

Das Beispiel kommt zwar ohne Makroprogrammierung aus, basiert dafür aber auf ziemlich komplexen *WENN*-Formeln. Wenn Ihnen der Umgang mit *WENN*-Formeln Probleme bereitet, finden Sie in Abschnitt 9.1 dazu Informationen.

Das Formelmodell

Das Formular sieht einen vierzelligen Eingabebereich vor, in dem sich zur einfacheren Orientierung bereits voreingestellte Werte befinden:

- E5 (jährlicher Zinssatz): 2.5 Prozent
- E6 (Sparbetrag): 100 €
- E7 (erster Einzahlungstermin): =HEUTE()
- E8 (Laufzeit): 1 Jahr

Daraus werden drei Ergebnisse berechnet: das Datum der letzten Einzahlung (n Jahre minus 1 Monat nach der ersten Einzahlung), das Ende der Laufzeit (1 Monat später) und der monatliche Zinssatz.

Die Datumsberechnungen demonstrieren den Umgang mit der Funktion *DATUM*, mit der ein gültiges Datum aus der Angabe (*Jahr; Monat; Tag*) erstellt wird. Die *DATUM*-Funktion ist dabei unglaublich flexibel: *DATUM*(2010;13;1) führt zum 1.1.2011, *DATUM*(2011; 2; 31) zum 3.3.2011, *DATUM*(2011; -3; -3) zum 28.8.2010. Es kann also wirklich beinahe bedenkenlos gerechnet werden; ungültige Monats- und Tagesangaben werden automatisch in sinnvolle Daten umgerechnet.

Bei der Berechnung des monatlichen Zinssatzes wird angenommen, dass die Verzinsung wirklich monatlich erfolgt. Daher darf der Zinssatz nicht einfach durch 12 dividiert werden, weil der daraus resultierende Jahreszinssatz dann wegen des resultierenden Zinseszins zu hoch ausfallen würde.

- E10 (letzte Einzahlung): =*DATUM*(*JAHR*(E7)+E8;*MONAT*(E7)-1;*TAG*(E7))
- E11 (Vertragsende): =*DATUM*(*JAHR*(E7)+E8;*MONAT*(E7);*TAG*(E7))
- E12 (monatlicher Zinssatz): =(1+E5)^(1/12)-1

Die eigentlichen Ergebnisse des Formulars – die im Verlauf der Vertragsdauer gutgeschriebenen Zinsen und das Endguthaben – resultieren aus der Monatstabelle im unteren Bereich des Formulars (B17:I53). Die Zinsgutschrift ergibt sich aus der Summe aller monatlichen Zinsen, das Endguthaben aus dem maximalen Betrag, der in den beiden Guthabenspalten gefunden werden kann. (Da die Länge der Tabelle von der Laufzeit des Vertrags abhängt, existiert keine genau definierte Zelle, in der das Ergebnis steht.)

- E13 (gesamte Zinsgutschrift): =*SUMME*(D17:D53;H17:H53)
- E14 (Gesamtguthaben): =*MAX*(E17:E53;I17:I53)

Nun zur Monatstabelle, deren Aufbau die meisten Formelprobleme verursacht. Die Tabelle ist aus Platzgründen zweispaltig konzipiert. Dadurch kann das gesamte Formular bis zu einer Laufzeit von sechs Jahren auf einer einzigen Seite ausgedruckt werden.

Die erste Zeile der Tabelle ist trivial und verweist einfach auf die entsprechenden Zellen des Eingabebereichs. In der Zinsspalte steht statt der Wert 0, weil am ersten Einzahlungstag noch keine Zinsen angefallen sind.

- B17 (Datum): =E7
- C17 (Einzahlung): =E6
- D17 (Zinsen): 0
- E17 (Guthaben): =C17

Ab der zweiten Zeile beginnen die allgemeingültigen Formeln, die nach einer einmaligen Eingabe durch Ausfüllen bzw. Kopieren auf die gesamte Tabelle verteilt werden. Wesentlich beim Formelapparat ist, dass sich zwar in allen Zellen der Tabelle Formeln befinden, aber nur in einer durch die Laufzeit vorgegebenen Anzahl von Zellen Ergebnisse angezeigt werden sollen. In den verbleibenden Zellen müssen die Formeln erkennen, dass die Vertragslaufzeit überschritten ist, und daher als Ergebnis eine leere Zeichenkette „“ liefern.

In der Datumsspalte wird getestet, ob die Zelle oberhalb ein Datum enthält (also nicht leer ist) und ob dieses Datum kleiner als das Datum des Vertragsendes ist. Wenn das der Fall ist, wird das neue Datum durch die Vergrößerung des Monats um 1 berechnet. In der Einzahlungsspalte wird getestet, ob sich in der Datumsspalte des Vormonats ein Datum befindet. Wenn das der Fall ist, wird der monatliche Einzahlungsbetrag angezeigt, sonst „“. Der Vormonatstest ist deswegen notwendig, weil in der letzten Zeile der Tabelle (Vertragsende) keine Einzahlung mehr erfolgt, wohl aber ein letztes Mal Zinsen gutgeschrieben werden.

Auch in der Zinsspalte erfolgt der Datumstest. Die Formel liefert als Ergebnis das Vormonatguthaben multipliziert mit dem monatlichen Zinssatz oder „“. In der Guthabenspalte werden zum Guthaben des Vormonats die Zinsen und die Einzahlung des aktuellen Monats addiert.

- B18 (Datum): =WENN(UND(B17<>"";B17<\$E\$11);
DATUM(JAHR(B17);MONAT(B17)+1;TAG(B17)); "")
- C18 (Einzahlung): =WENN(B19<>"";C17;"")
- D18 (Zinsen): =WENN(B18<>"";E17*\$E\$12;"")
- E18 (Guthaben): =WENN(B18<>"";SUMME(E17;C18:D18);"")

Beachten Sie bei der Eingabe der Formeln, dass einige Zellbezüge (\$E\$11, \$E\$12) absolut sind, sonst gibt es beim Kopieren bzw. Ausfüllen der Zellen Probleme.

Die für eine Zeile eingegebenen Formeln können nun durch Ausfüllen nach unten kopiert werden. Markieren Sie dazu die vier Zellen B18:E18, und ziehen Sie das kleine Ausfüllkästchen (rechte untere Ecke des Zellbereichs) bis zur Zelle E53 nach unten.

Excels Ausfüllfunktion ist nicht in der Lage, die Formeln selbstständig so zu adaptieren, dass die Tabelle in der zweiten Spalte fortgesetzt wird. Sie können sich aber leicht behelfen, indem Sie die Formeln der ersten Spalte zwei Zeilen weiter ausfüllen (bis E55) und anschließend den Zellbereich B54:E55 nach F17 verschieben (Zellen markieren und am Rand der Markierung mit der Maus verschieben). Anschließend können Sie die zweite Spalte ebenso wie die erste Spalte der Tabelle mit Formeln ausfüllen.



Anmerkung

Die Datumsformel (B18) ist in einer Beziehung nicht optimal: Wenn als Startdatum der 31. 1.08 angegeben wird, dann ergibt sich als nächstes Datum der 2. 3. 08 (es gibt ja keinen 31. 2. 08). In der Folge verrutschen alle Einzahlzeiten um drei Tage, das Enddatum stimmt nicht mit E11 überein etc. Dieses Problem kann vermieden werden, wenn eine weitere Spalte mit durchlaufenden Nummern für die Einzahlungen eingeführt wird (1 für die erste Einzahlung, 2 für die zweite etc.). Damit kann das Einzahldatum in der Form

DATUM(JAHR(E7); MONAT(E7)+Laufnummer-1; TAG(E7))

berechnet werden.

Tabellenlayout, Zellschutz, Druckoptionen

Mit dem Aufbau des Formelmodells ist die mühsamste Arbeit erledigt. Sie müssen jetzt die Tabelle noch so formatieren, dass sie äußerlich ansprechend aussieht (kleine Schrift (8 Punkt) für die Monatstabelle, Rahmenlinien, Zahlen- und Datumsformate, Ausrichtung, Hintergrundfarbe für den Eingabebereich etc.). Über DATEI | OPTIONEN | ERWEITERT können Sie die Anzeige der Rasterlinien, der Zeilen- und Spaltenköpfe, der horizontalen Bildlaufleiste und der Blattregister deaktivieren.

Testen Sie mit ANSICHT | SEITENLAYOUT, ob das Formular auf einer Druckseite Platz hat und sie gut füllt. Gegebenenfalls können Sie die Breite und Höhe einzelner Zeilen oder Spalten anpassen, um eine bessere Nutzung der Seite zu erzielen. Über SEITENLAYOUT | SEITE EINRICHTEN verändern Sie Kopf- und Fußzeilen (am besten stellen Sie „keine“ ein) und im Dialogregister SEITENRÄNDER können Sie eine horizontale und vertikale Zentrierung des Ausdrucks einstellen.

Als Nächstes sollten Sie die Tabelle vor irrtümlichen Veränderungen durch den Anwender schützen. Markieren Sie dazu zuerst den Eingabebereich, und deaktivieren Sie für diese Zellen die Option „gesperrt“ (Kontextmenü ZELLEN FORMATIEREN | SCHUTZ). Anschließend schützen Sie die gesamte Tabelle (mit Ausnahme der soeben formatierten Zellen) durch ÜBERPRÜFEN | BLATT SCHÜTZEN. Auf die Angabe eines Kennworts sollten Sie dabei verzichten.

Validitätskontrolle

Die vier Eingabezellen des Tabellenblatts sind gegen fehlerhafte Eingaben abgesichert. Dazu wurden mit DATEN | DATENÜBERPRÜFUNG das gewünschte Datenformat, Gültigkeitsregeln, ein kurzer Infotext und eine Text für die Fehlermeldung (im Fall einer ungültigen Eingabe) formuliert. Die Möglichkeit, Validitätsregeln zu formulieren, besteht seit Excel 97.

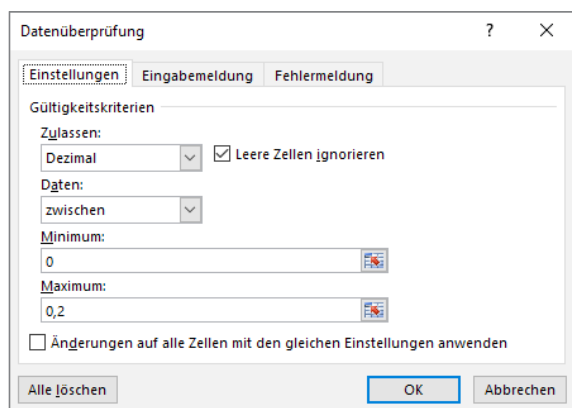


BILD 1.10:
Formulierung der Gültigkeitsregeln
für die Verzinsung

Mustervorlagen

Die Tabelle hat jetzt einen Zustand, in dem der Anwender sie mühelos verwenden kann: Er muss lediglich die vier Daten des Eingabebereichs verändern und kann das Ergebnis sofort ausdrucken. Damit die Tabelle in diesem Zustand bleibt und nicht unbeabsichtigt verändert wird, speichern Sie sie mit DATEI | SPEICHERN UNTER als Mustervorlage (der Dateityp heißt „Excel-Vorlage (*.xltx)“). Der Speicherort hängt von der konkreten Excel-Version ab. Wenn Sie Excel 2007 oder 2010 verwenden, sollten Sie dieses Verzeichnis wählen:

C:\Users\[Benutzername]\AppData\Roaming\Microsoft\Templates

Anwender von Excel 2013 (und neuer) speichern die Vorlagendatei im Unterordner *Templates\1031* des Office-Installationsordners. Der exakte Pfad könnte beispielsweise so lauten:

C:\Program Files (x86)\Microsoft Office\Templates\1031

Mustervorlagen sind Excel-Dateien, die als Vorlagen für neue Tabellen dienen. Der Benutzer lädt die Mustervorlage, ändert darin einige Informationen und speichert die Tabelle anschließend unter einem neuen Namen ab. Excel sorgt automatisch dafür, dass der Benutzer beim Speichern einen eigenen Dateinamen angeben muss und die Mustervorlage nicht durch eigene Veränderungen überschreiben kann. Damit Mustervorlagen von Excel als solche erkannt werden, müssen sie in einem eigenen Format (Dateikennung *.xltx) und an einem bestimmten Ort (siehe oben) gespeichert werden.

Sie stehen dann automatisch im DATEI-NEU-Dialog für die Gestaltung neuer Arbeitsmappen zur Wahl. Leider gilt das nur für die Excel-Versionen 2007 und 2010. In Excel ab Version 2013 muss man die Verwendung lokaler Mustervorlagen zunächst erzwingen. Und zwar durch eine Einstellungsänderung, die im Abschnitt „Mustervorlagen ab Excel 2013“ des Kapitels 5.9.3 beschrieben wird.



Hinweis

Um die Besonderheiten von Mustervorlagen richtig ausprobieren zu können, müssen Sie die Beispieldatei *form.xltx* in das oben genannte Verzeichnis kopieren. Außerdem müssen Sie zum Öffnen das Kommando DATEI | NEU verwenden, nicht aber DATEI | ÖFFNEN! (Damit würden Sie die Mustervorlage als solche öffnen, etwa um Veränderungen an der Vorlage durchzuführen.)

**Verweis**

Weitere Beispiele für Mustervorlagen und „intelligente“ Formulare finden Sie in Kapitel 9. Dieses Kapitel ist zur Gänze der Programmierung und Anwendung solcher Tabellen gewidmet. Beispielsweise kann durch Programmcode erreicht werden, dass die Vorlage beim Laden automatisch initialisiert wird, dass Buttons zum Ausdruck bereitgestellt werden etc.

■ 1.7 Beispiel: Benutzerdefinierte Funktionen

Das vorangegangene Beispiel hat gezeigt, wie eine Tabelle durch den Einsatz ziemlich komplizierter Formeln „intelligent“ gestaltet werden kann. Die Verwendung von Formeln wie im vorherigen Beispiel stößt allerdings bald an Grenzen – die Formeln werden zu lang und praktisch nicht mehr handhabbar. Dieses Problem können Sie vermeiden, indem Sie selbst neue Funktionen definieren. Eigene Funktionen können aber auch Aufgaben erfüllen, die durch Tabellenformeln unlösbar wären – etwa die Umwandlung einer Zahl in einen Text, wie sie bei der automatisierten Ausstellung von Schecks (die Zahl 12,34 beispielsweise wird hier zum Text „zwei Komma drei vier“) erforderlich ist.

Die Definition eigener Funktionen setzt bereits ein etwas tiefer gehendes Wissen über die Programmierung in VBA voraus. Die Makroaufzeichnung ist für diesen Zweck leider nicht brauchbar, weil es sich ja um eine Rechenfunktion und nicht um eine Kommandoabfolge handelt. Die folgenden Beispiele sind allerdings ganz einfach gehalten und sollten eigentlich auch ohne Programmierkenntnisse verständlich sein.

Die erste Funktion berechnet den Flächeninhalt eines Kreises. Bevor Sie die neue Funktion in einer Tabelle verwenden können, müssen Sie den Code in ein Modul eingeben. Dazu wechseln Sie mit Alt+F11 in die Entwicklungsumgebung und führen EINFÜGEN | MODUL aus.

```
' Beispiel 01\function.xlsm
Function CircleArea(radius As Double) As Double
    CircleArea = radius ^ 2 * Application.Pi
End Function
```

Function leitet ähnlich wie *Sub* aus den vorangegangenen Beispielen eine Prozedur ein. Der Unterschied zu einer *Sub*-Prozedur besteht darin, dass eine *Function*-Prozedur einen Wert zurückgibt. Aus diesem Grund wird dem Funktionsnamen *CircleArea* in der zweiten Zeile ein Berechnungsausdruck zugewiesen. *radius* ist ein Parameter der Funktion. Wenn Sie im Tabellenblatt die Formel `=CircleArea(5)` eingeben, dann führt Excel die Funktion aus und setzt dabei automatisch den Wert 5 in den Parameter *radius* ein. Mit *Application.Pi* greifen Sie auf die Zahl 3,1415927 zurück.

Nun zur zweiten Funktion, die schon ein bisschen sinnvoller ist: Sie berechnet das Produkt aus Preis und Anzahl. Dabei wird automatisch ein Rabatt von fünf Prozent berücksichtigt, wenn die Stückzahl mindestens zehn beträgt. Zur Erkennung dieses Sonderfalls wird eine *If*-Abfrage eingesetzt.

```
Public Function Discount(unitprice, pieces)
    If pieces >= 10 Then
        Discount = pieces * unitprice * 0.95
    Else
        Discount = pieces * unitprice
    End If
End Function
```



Verweis

Normalerweise werden benutzerdefinierte Funktionen natürlich für anspruchsvollere Aufgaben eingesetzt. Details zur Programmierung benutzerdefinierter Funktionen finden Sie in Abschnitt 5.7.

■ 1.8 Beispiel: Analyse komplexer Tabellen

Als Excel-Anwender sind Sie immer wieder mit komplexen Tabellen konfrontiert, die Sie nicht selbst (oder vor sehr langer Zeit) erstellt haben. Im Regelfall ist es schwierig, sich in solchen Tabellen zu orientieren. Es ist nicht klar, welche Zellen aus Eingaben resultieren, welche Zellen sich aus Formeln ergeben etc. Die Befehlsgruppe FORMELÜBERWACHUNG in der Befehlsregisterkarte FORMELN ist bei der Analyse eines komplexen Formelmodells zwar äußerst hilfreich, ist für eine erste Orientierung aber auch nicht optimal geeignet. Eben diese Aufgabe übernimmt das hier vorgestellte Makro: Es analysiert alle Zellen des aktiven Arbeitsblatts. Zeichenketten werden blau, Formeln rot formatiert. (Natürlich wären auch weitere Untersuchungen des Inhalts oder andere Formatierungen möglich.)

Das Beispiel zeichnet sich unter anderem dadurch aus, dass es nicht mit der Makroaufzeichnung erstellt werden kann – es gibt ja keine vergleichbaren eingebauten Funktionen in Excel. Die Programmierung eines Makros in dieser Art erfordert daher schon ein relativ ausführliches Wissen über die Objektbibliothek von Excel und insbesondere über den Umgang mit Zellen (siehe Abschnitt 5.1).

Der Programmcode beginnt mit einem Test, ob es sich beim aktiven Blatt überhaupt um ein Tabellenblatt handelt (es könnte ja auch ein Diagramm sein). *TypeName* liefert als Ergebnis den Namen des Objekttyps, also beispielsweise *Worksheet* oder *Chart*. Wenn ein Tabellenblatt vorliegt, werden aus Geschwindigkeitsgründen vorübergehend die automatische Neuberechnung und die Bildschirmaktualisierung abgeschaltet. Anschließend werden alle benutzten Zellen der Reihe nach analysiert:

Mit *HasFormula* kann ganz einfach festgestellt werden, ob es sich um eine Formel handelt. Mit *TypeName(c.Value)="String"* werden Zeichenketten erkannt. (Mit vergleichbaren Abfragen könnten Sie auch Daten oder Währungswerte – etwa 25,50 Euro – feststellen.) Zur Formatierung wird die *Color*-Eigenschaft des *Font*-Objekts der gerade bearbeiteten Zelle verändert.


```
' Beispiel 01\analyse.xlsm
Sub AnalyseWorksheet()
    Dim c As Range 'cell
    If TypeName(ActiveSheet) <> "Worksheet" Then Exit Sub
    Application.Calculation = xlCalculationManual
    Application.ScreenUpdating = False
    For Each c In ActiveSheet.UsedRange
        If c.HasFormula Then
            c.Font.Color = RGB(192, 0, 0)
        ElseIf TypeName(c.Value) = "String" Then
            c.Font.Color = RGB(0, 0, 192)
        Else
            c.Font.Color = RGB(0, 0, 0)
        End If
    Next
    Application.Calculation = xlCalculationAutomatic
    Application.ScreenUpdating = True
End Sub
```

■ 1.9 Beispiel: Vokabeltrainer

Beim letzten Beispiel dieses Kapitels ist der spielerische Aspekt ein wenig stärker ausgeprägt. Das Programm hilft beim Lernen von Vokabeln. Ausgangspunkt ist die in Abbildung 1.11 dargestellte Tabelle mit Vokabeln in zwei Sprachen (hier Schwedisch und Deutsch). Die Spalten C, D, E und F geben an, ob das Wort in die eine oder in die andere Richtung (also Schwedisch → Deutsch oder Deutsch → Schwedisch) schon einmal erkannt bzw. erraten worden ist bzw. wie oft es danach abgefragt wurde.

Vokabeltrainer starten		Sprache 1 --> Sprache 2		Sprache 2 --> Sprache 1	
Schwedisch	Deutsch	richtig	abgefragt	richtig	abgefragt
kvällen	abends	1	1		
på universitet	an der Univ.	1	1		
törst, -en, -er	Durst	1	1	1	1
biljett, -en, -er	Eintritts-/Fahrkarte	1	1		
eld, -en, -ar	Feuer				1
juridik, -en	Jura				

BILD 1.11: Die Vokabelliste mit Abfrage- und Lernergebnissen (Spalte C bis F)

Wenn Sie den Trainer starten, erscheint der Dialog aus Abbildung 1.12. Aus der Vokabelliste wird zufällig ein Wort ausgewählt, wobei Vokabeln bevorzugt werden, die noch nie korrekt erraten wurden. Die Vokabelabfrage erfolgt (ebenfalls zufällig) in beide Richtungen. Wenn Sie das Wort schon kennen, drücken Sie OK, sonst SPÄTER NOCHMALS FRAGEN.

Mit EINTRAG KORRIGIEREN und TRAINER BEENDEN verlassen Sie den Dialog. Im ersten Fall wird der Eingabecursor in die Zeile der Vokabeltabelle gesetzt, aus der das zuletzt abgefragte Wort stammt. Das ermöglicht eine einfache Korrektur von Vokabeln.

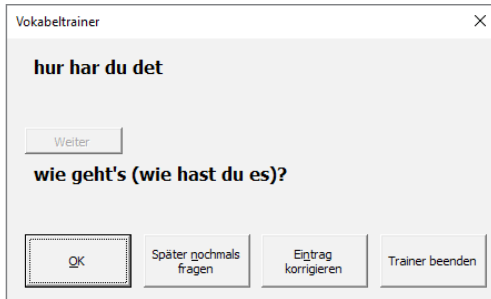


BILD 1.12:
Der Dialog des Vokabeltrainers

Dialogaufbau



Anmerkung

Fast der gesamte Programmcode dieses Beispiels ist mit dem Dialog aus Abbildung 1.13 verbunden. Die größte Hürde besteht denn auch darin, diesen Dialog zu „bauen“. Wenn Sie noch nie mit einem Dialogeditor gearbeitet haben, sollten Sie vielleicht zuerst einen Blick in Kapitel 7 werfen, um die folgenden, relativ knappen Ausführungen nachvollziehen zu können.

Die Arbeit beginnt in der VBA-Entwicklungsumgebung (also Alt+F11). Dort erzeugen Sie mit EINFÜGEN | USERFORM einen neuen Dialog. Mit ANSICHT | EIGENSCHAFTENFENSTER öffnen Sie das Eigenschaftensfenster, dessen Inhalt sich immer auf das gerade ausgewählte Objekt im Dialog bezieht. Den internen Namen des Dialogs sowie seine Beschriftung stellen Sie über die Eigenschaften *Name* und *Caption* ein. Im Beispiel wird *formQuery* als Objektname und *Vokabeltrainer* als Überschrift verwendet.

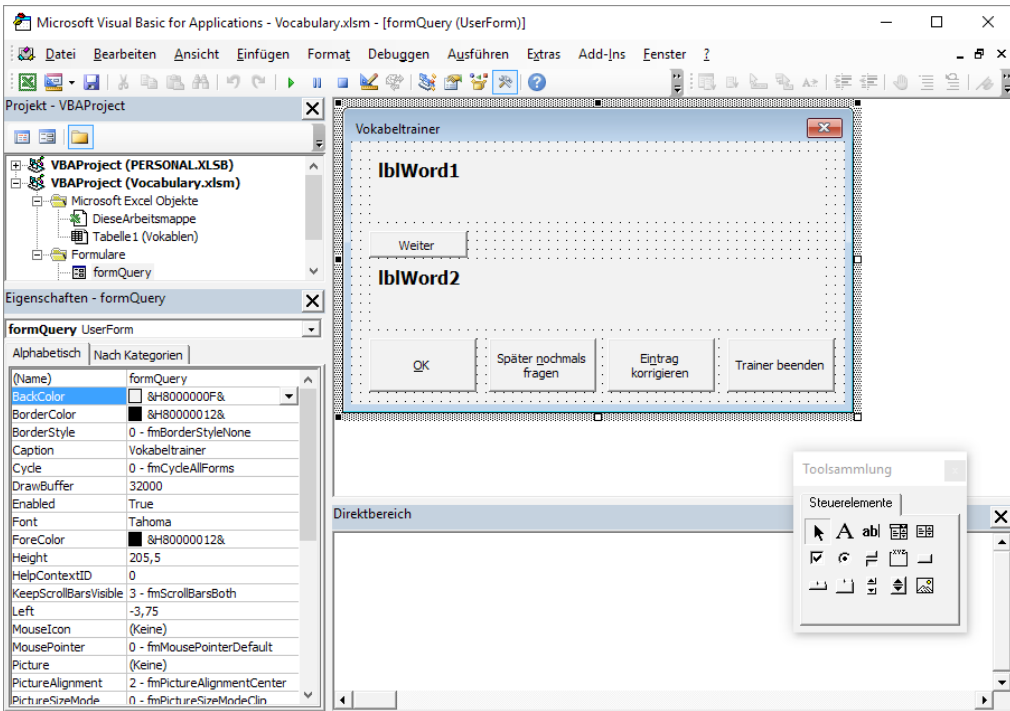


BILD 1.13: Dialogentwurf

Jetzt fügen Sie wie in Abbildung 1.13 erkennbar zwei Beschriftungsfelder (Label) und fünf Buttons in den Dialog ein. Dazu wählen Sie zuerst das betreffende Steuerelement in der Werkzeugsammlung aus (ANSICHT | WERKZEUGSAMMLUNG) und zeichnen dann mit der Maus den Rahmen des Steuerelements im Dialog. Das Beschriftungsfeld ist in der Werkzeugsammlung durch den Großbuchstaben A gekennzeichnet.

Bei allen sieben Steuerelementen müssen Sie wie schon zuvor für den Dialog die *Name*- und *Caption*-Eigenschaften einstellen. Im Beispielpogramm gelten die folgenden Einstellungen:

Name	Caption	Verwendungszweck
<i>lblWord1</i>	lblWord1	Anzeige der ersten Vokabel
<i>lblWord2</i>	lblWord2	Anzeige der zweiten Vokabel
<i>btnNext</i>	Weiter	zweite Vokabel anzeigen
<i>btnOK</i>	OK	Wort erkannt, weiter mit nächstem Wort
<i>btnAgain</i>	Später nochmals ...	Wort nicht erkannt, weiter
<i>btnEdit</i>	Eintrag korrigieren	Dialog verlassen, Wort in Tabelle ändern
<i>btnEnd</i>	Trainer beenden	Dialog verlassen

Einige weitere Einstellungen sind für die korrekte Funktion des Programms zwar nicht unbedingt erforderlich, erleichtern aber die Bedienung: Bei den beiden Beschriftungsfel-

dern können Sie im Eigenschaftenfenster bei der Eigenschaft *Font* eine größere Schrift einstellen. Bei den Buttons können Sie bei der Eigenschaft *Accelerator* einen Buchstaben angeben – dann können Sie den Button später mit Alt+Buchstabe bequem auswählen. Und schließlich können Sie für den Trainer-beenden-Button die Eigenschaft *Cancel* auf *True* stellen – damit gilt dieser Button als Abbruch-Button und kann jederzeit mit Esc ausgewählt werden.

Programmcode zum Dialog



Hinweis

Auch der Programmcode dieses Beispiels ist schon ein wenig fortgeschritten. Wenn Sie noch gar keine Programmiererfahrung haben, sollten Sie vielleicht zuerst einen Blick in Kapitel 4 werfen, in dem elementare Grundbegriffe (Variablen, Schleifen etc.) behandelt werden.

Die Vorbereitungsarbeiten sind jetzt abgeschlossen. Jetzt geht es noch darum, den Dialog mit Prozeduren zu erweitern, die bei der Anzeige des Dialogs bzw. beim Anklicken der diversen Buttons ausgeführt werden. Zur Kommunikation zwischen diesen Prozeduren müssen einige Informationen in Variablen gespeichert werden, die am Beginn des Code-Moduls zum *queryForm*-Dialog definiert werden. (Das Zeichen & dient übrigens zur Identifizierung von *Long*-Variablen für die Speicherung ganzer Zahlen.)

```
' Beispiel 01\vocabulary.xlsm
Option Explicit
Dim firstline&      'erste Zeile mit Vokabeln
Dim lastline&       'letzte Zeile mit Vokabeln
Dim linenr&         'aktuelle Zeile in der Vokabeltabelle
Dim querymodus&     'Fragemodus (0: Sprache 1 --> Sprache 2,
                    '          1: Sprache 2 --> Sprache 1)
Dim startcell As Range 'Zelle, bei der die Vokabelliste beginnt
Const maxTries = 20   'Suche nach noch unbekannten Vokabeln
```

Die Prozedur *UserForm_Initialize* wird automatisch ausgeführt, sobald der Dialog angezeigt wird. Solange Sie sich in der Entwicklungsumgebung befinden, können Sie dazu einfach F5 drücken.

In der Prozedur wird der Inhalt der beiden Beschriftungsfelder gelöscht. Außerdem werden die Variablen *startcell*, *firstline* und *lastline* initialisiert. *startcell* bezeichnet die erste Tabellenzelle der Vokabelliste und wird im Rest des Programms als Startpunkt für die Adressierung der weiteren Zellen der Vokabelliste verwendet. *firstline* und *lastline* geben die erste und die letzte Zeilennummer des Vokabelbereichs an.

Bei der Berechnung von *lastline* wird *CurrentRegion* verwendet, um den gesamten Tabellenbereich (inklusive Überschrift) der Vokabelliste zu ermitteln. *Rows* zerlegt diesen Bereich in Zeilen, *Count* ermittelt deren Anzahl. (Diese Eigenschaften werden in Abschnitt 5.1 ausführlich beschrieben.)

```

Private Sub UserForm_Initialize()
    lblWord1 = "" 'Inhalt der beiden Beschriftungsfelder löschen
    lblWord2 = ""
    Set startcell = Worksheets(1).Range("a3")
    firstline = startcell.Row
    lastline = startcell.CurrentRegion.Rows.Count
    Randomize      'Zufallszahlengenerator neu initialisieren
    ShowNewWord    'erstes Wort anzeigen
End Sub

```

Die Prozedur *ShowNewWord* hat die Aufgabe, ein (möglichst noch ungelerntes) Wort aus der Tabelle zu lesen und im ersten Beschriftungsfeld anzuzeigen. Der Suchalgorithmus ist ziemlich trivial: Mit der Zufallszahlenfunktion *Rnd*, die Zahlen zwischen 0 und 1 liefert, wird eine Zeile (*linenr*) und eine Abfragerichtung (*querymodus*) erzeugt. Mit der Methode *Offset(zeile, spalte)* wird dann je nach *querymodus* die Spalte C oder E der Vokabeltabelle überprüft (siehe Abbildung 1.11). Wenn die entsprechende Zelle leer ist bzw. den Wert 0 enthält, gilt das Wort noch als ungelernt, und die Schleife wird vorzeitig verlassen.

Falls auch nach *maxTries* Versuchen kein ungelerntes Wort entdeckt wird, wird eben ein schon bekanntes Wort abgefragt. Für die Vorgehensweise spielt das ohnedies keine Rolle – das Wort wird abermals via *Offset* aus der Tabelle gelesen und im ersten Beschriftungsfeld angezeigt. Der Inhalt des zweiten Beschriftungsfelds, in dem noch die Vokabel der vorigen Abfrage steht, wird gelöscht. Die folgenden drei Anweisungen aktivieren den Button **WEITER** und deaktivieren die Buttons **OK** und **NOCHMAL**S. Außerdem wird der Eingabefokus in den **WEITER**-Button versetzt, sodass dieser Button bequem mit **RETURN** ausgewählt werden kann.

```

' ein Wort zufällig auswählen und anzeigen
Sub ShowNewWord()
    Dim i&
    ' versucht, ein noch unbekanntes Wort zu finden
    For i = 1 To maxTries
        linenr = Int(Rnd * (lastline - firstline + 1))
        querymodus = Int(Rnd * 2)
        If Val(startcell.Offset(linenr, 2 + querymodus * 2)) = 0 Then
            Exit For
        End If
    Next
    lblWord1 = startcell.Offset(linenr, querymodus)
    lblWord2 = ""
    btnNext.Enabled = True
    btnOK.Enabled = False
    btnAgain.Enabled = False
    btnNext.SetFocus
End Sub

```

Der Anwender sieht jetzt den Dialog mit nur einem Wort und versucht, das andere zu erraten. Schließlich klickt er den Button **WEITER** an. In der Prozedur *btnNext_Click* wird die Vokabel in der jeweils anderen Sprache im zweiten Beschriftungsfeld *lblWord2* angezeigt. Der Button **WEITER** wird deaktiviert, dafür werden **OK** und **NOCHMAL**S aktiviert.

```
' das passende Wort in der anderen Sprache anzeigen
Private Sub btnNext_Click()
    lblWord2 = startcell.Offset(linenr, 1 - querymodus)
    btnNext.Enabled = False
    btnOK.Enabled = True
    btnAgain.Enabled = True
    btnOK.SetFocus
End Sub
```



Tipp

Der Prozedurname *btnNext_Click* ergibt sich aus dem Namen des Objekts (hier also *btnNext*) und dem Namen des Ereignisses (*Click*). Zur Codeeingabe führen Sie im Dialogfenster einfach einen Doppelklick für das betreffende Steuerelement aus. Damit werden die Zeilen *Private Sub name* und *End Sub* automatisch in den Programmcode eingefügt.

Wenn der Anwender das Wort erkannt hat, drückt er auf OK. In *btnOK_Click* wird daraufhin in der Spalte C oder E (abhängig von *querymodus*) gespeichert, wie oft das Wort bereits richtig erraten wurde. Weiter wird in Spalte D oder F gespeichert, wie oft bereits gefragt wurde. Der Aufruf von *ShowNewWord* löst die Anzeige des nächsten Words aus.

```
' Vokabel ist bekannt
Private Sub btnOK_Click()
    ' Spalte C/E (richtig)
    startcell.Offset(linenr, 2 + querymodus * 2) = _
        Val(startcell.Offset(linenr, 2 + querymodus * 2) + 1)
    ' Spalte D/F (abgefragt)
    startcell.Offset(linenr, 3 + querymodus * 2) = _
        Val(startcell.Offset(linenr, 3 + querymodus * 2) + 1)
    ShowNewWord
End Sub
```

btnAgain_Click funktioniert wie *btnOK_Click*. Der einzige Unterschied besteht darin, dass nur die Spalten D und F verändert werden, nicht aber die Spalten C und E.

```
' Vokabel ist noch unbekannt
Private Sub btnAgain_Click()
    startcell.Offset(linenr, 3 + querymodus * 2) = _
        Val(startcell.Offset(linenr, 3 + querymodus * 2) + 1)
    ShowNewWord
End Sub
```

Die beiden Prozeduren *btnEdit_Click* und *btnEnd_Click* beenden beide den Dialog. Dazu wird die Anweisung *Unload Me* verwendet. Im ersten Fall wird der Zellzeiger anschließend zum zuletzt angezeigten Wort bewegt, damit dieses korrigiert werden kann. Im zweiten Fall wird in einer Dialogbox gefragt, ob die geänderte Vokabelliste gespeichert werden soll.

```

' Vokabel soll korrigiert werden
Private Sub btnEdit_Click()
    Worksheets(1).Activate
    startcell.Offset(linenr).Activate
    Unload Me
End Sub
' Dialog beenden, Tabelle speichern
Private Sub btnEnd_Click()
    Dim result&
    Unload Me
    result = MsgBox("Soll die Vokabelliste gespeichert werden?", _
        vbYesNo)
    If result = vbYes Then ActiveWorkbook.Save
End Sub

```

Sonstiger Code

Damit der Dialog aus der Vokabeltabelle bequem gestartet werden kann, wird dort ein Button (*btnStartTrainer*) eingefügt. In der Ereignisprozedur wird der Dialog mit *Show* angezeigt. Dadurch kommt es automatisch zum Aufruf von *UserForm_Initialize*, und der weitere Programmablauf erfolgt wie oben beschrieben.

```

' Beispiel 01\Vocabulary.xls, Tabelle 1
Private Sub btnStartTrainer_Click()
    formQuery.Show
End Sub

```

Verbesserungsmöglichkeiten

Natürlich gäbe es zahllose Möglichkeiten, dieses Programm zu verbessern: durch einen komfortablen Eingabedialog für neue Vokabeln, durch einen Optionsdialog zur Steuerung des Abfragemodus (z. B. Abfragen nur in eine Richtung), durch einen raffinierteren Algorithmus zur Auswahl der abzufragenden Vokabeln, durch die Ergänzung der Vokabeltabelle um eine zusätzliche Lautschriftspalte etc. Der Fantasie sind diesbezüglich keinerlei Grenzen gesetzt!

■ 1.10 Weitere Beispiele zum Ausprobieren

Dieser Abschnitt gibt eine kurze Beschreibung der interessantesten Beispiele dieses Buchs. Die Abbildungen sollen dazu einladen, die Beispiele einfach einmal zu laden und anzusehen. Gleichzeitig soll der Abschnitt vermitteln, wie weit die Möglichkeiten der VBA-Programmierung reichen.



Hinweis

Ein Teil der Beispieldateien kann nicht direkt von den Download-Dateien zum Buch verwendet werden. Installieren Sie die Beispieldateien zuerst wie im Anhang beschrieben in ein Verzeichnis Ihrer Festplatte! Wenn die Programme in etwa dem entsprechen, was Sie selbst mit Excel vorhaben, können Sie in den angegebenen Abschnitten die Details nachlesen.

Kalender und Feiertage

In vielen Excel-Anwendungen tritt das Problem auf, dass Feiertage korrekt berücksichtigt werden sollen. 05\Holidays.xlsm zeigt, wie Feiertage berechnet werden. Ganz nebenbei fällt noch ein kleines Programm ab, das einen Kalender für ein beliebiges Jahr erzeugt.

	Januar	Februar	März	April	Mai	Juni	Juli
1	Neujahr	1	1	1	1. Mai	1	1
2	2	2	2	2	2	2	2
3	3	3	3	Karfreitag	3	3	3
4	4	4	4	4	4	Fronleichnam	4
5	5	5	5	Ostersonntag	5	5	5
6	6	6	6	Ostermontag	6	6	6
7	7	7	7	7	7	7	7
8	8	8	8	8	8	8	8
9	9	9	9	9	9	9	9
10	10	10	10	10	10	10	10
11	11	11	11	11	11	11	11
12	12	12	12	12	12	12	12
13	13	13	13	13	13	13	13
14	14	14	14	14	Christi Himmelfahrt	14	14
15	15	15	15	15	15	15	15
16	16	16	16	16	16	16	16
17	17	17	17	17	17	17	17
18	18	18	18	18	18	18	18
19	19	19	19	19	19	19	19
20	20	20	20	20	20	20	20
21	21	21	21	21	21	21	21

BILD 1.14: Ein mit Excel erstellter Kalender

Eigene Dialoge gestalten

Excel bietet die Möglichkeit, selbst Dialoge zu gestalten, anzuzeigen und per Programmcode auszuwerten. Eine große Anzahl solcher Dialoge finden Sie in der Datei 07\Userform.xlsm. Die Dialoge können per Mausklick aufgerufen werden.

Automatische Datenprotokollierung mit Diagrammen

Die umfassenden Möglichkeiten zur Gestaltung von Diagrammen in Excel werden oft dazu verwendet, um große Datenmengen in grafischer Form zu protokollieren. Dieser Prozess lädt natürlich zur Automatisierung ein. In *10\Chart.xlsm* wird gezeigt, wie (ausgehend von simulierten Testdaten) automatisch Tages- und Monatsprotokolle erstellt werden können.

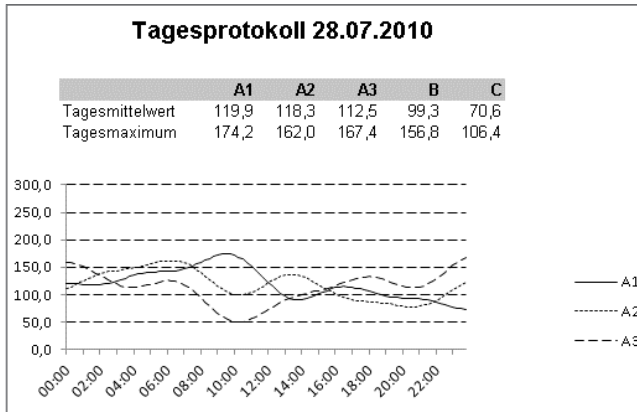


BILD 1.17:
Ein automatisch erzeugtes Diagramm

Abrechnung eines Car-Sharing-Vereins

Als Mitglied eines Car-Sharing-Vereins besitzen Sie kein eigenes Auto, sondern leihen es sich aus, wenn Sie es benötigen. Das Beispiel *11\DB_Share.xlsm* zeigt, wie ein „intelligentes“ Formular (nämlich *09\Share.xlsm*) zu einer einfachen Datenbankanwendung ausgebaut werden kann. Mit DB_Share verwalten Sie den Fuhrpark, die Teilnehmeradressen und drucken die Abrechnungen für einzelne Fahrten aus.

Zeittarif

Stundentarif	von	bis	
Uhrzeit	11:15	23:30	
ergibt			
Stunden Tarif I	9		10,80
Stunden Tarif II	4		2,60
Stunden Tarif III	0		0,00

und/oder

Tagestarif	von	bis
Datum		
abzüglich Wochenbonus	---	

plus

Kilometertarif

gefahrte Kilometer	220	40,70
--------------------	-----	-------

abzüglich

ausgelegte Treibstoffkosten		0,00
-----------------------------	--	------

Drucken und Speichern

Neue Rechnung

Endsumme **54,10**

19 % Ust 8,64

Nettobetrag 45,46

BILD 1.18:
Das Formular ist mit einer kleinen Datenbankanwendung verbunden.

Tarife	in €
Stundentarif I (8-20 Uhr)	1,20

alle Beträge in €

Auswertung von Fragebögen

Die Auswertung von Fragebögen ist eine mühsame Angelegenheit – es sei denn, man lässt sich von Excel helfen. Im Verzeichnis 12\survey finden Sie sowohl ein Beispiel für einen in Excel realisierten Fragebogen als auch Makros für deren automatische Auswertung.

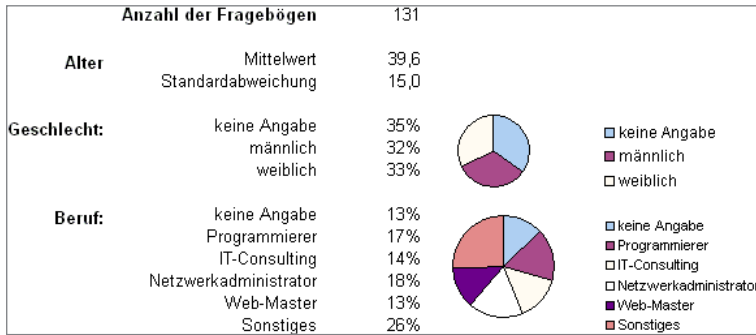


BILD 1.19: Auswertung von Fragebögen

Datenanalyse mit Excel (Pivot-Tabellen)

Excel ist ein hervorragendes Werkzeug zur Analyse von Daten – egal, ob sich diese in einer Excel-Datei oder in einer externen Datenbank befinden. 13\Pivot.xlsm gibt eine Menge Beispiele für Pivot-Tabellen und deren Programmierung.

		Qual.		
Kat.	Daten	I	II	Gesamtergebnis
a	Anzahl - Artikel	4	2	6
	Mittelwert - Preis	38,75	27,50	35,00
b	Anzahl - Artikel	4	2	6
	Mittelwert - Preis	83,75	60,00	75,83
c	Anzahl - Artikel	6	4	10
	Mittelwert - Preis	132,50	87,50	114,50
Gesamt: Anzahl - Artikel		14	8	22
Gesamt: Mittelwert - Preis		91,79	65,63	82,27

BILD 1.20:
Eine einfache Pivot-Tabelle

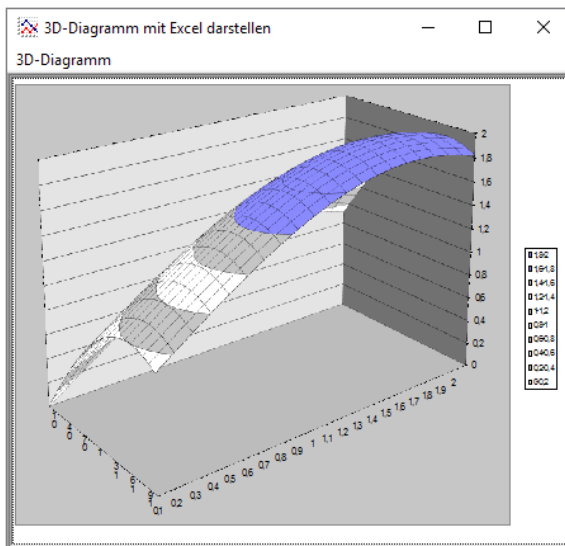
ActiveX-Automation

Das Beispiel 15\ActiveX-Access.xlsm zeigt, wie Access – ferngesteuert via ActiveX-Automation – einen Bericht mit der Produktaufstellung der Northwind-Datenbank ausdruckt. Voraussetzung ist allerdings, dass Sie Access besitzen.

Products by Category			
28. Jul. 10			
Category: Beverages		Category: Condiments	
Product Name:	Units In Stock:	Product Name:	Units In Stock:
Chai	39	Aniseed Syrup	13
Chang	17	Chef Anton's Cajun Seasoning	53
Chartreuse verte	69	Genen Shouyu	39
Côte de Blaye	17	Grandma's Boysenberry Spread	120
Ipoh Coffee	17	Gula Malacca	27
Lakkalikööri	57	Louisiana Fiery Hot Pepper Sauce	76
Laughing Lumberjack Lager	52	Louisiana Hot Spiced Okra	4
Outback Lager	15	Northwood's Cranberry Sauce	6
Rhönbräu Klosterbier	125	Original Frankfurter grüne Soße	32
Sasquatch Ale	111	Sirup d'érable	113
Steeleye Stout	20	Veggie-spread	24
Number of Products:	11	Number of Products:	11

BILD 1.21: Mit Excel externe Programme steuern

Nicht nur Excel kann via ActiveX-Automation fremde Programme steuern – auch der umgekehrte Weg ist möglich. Das Visual-Basic-Programm `15\Vb6\Chart\ExcelChart.exe` nutzt die Diagrammfunktionen von Excel, um eigene Daten in einem Excel-Diagramm anzuzeigen.



2

Neuerungen in Excel 2007 bis 2016

Das augenfälligste Merkmal von Excel 2007 war eine radikal neue Benutzeroberfläche. Da sich die Anwender inzwischen wohl mehrheitlich an die *Multifunktionsleiste* gewöhnt haben, hat Microsoft sie auch in die Nachfolgeversionen 2010 und 2016 übernommen. Der neue Name „Menüband“ signalisiert allerdings schon diverse Änderungen. Die wichtigste: Der Anwender kann das Befehlsarsenal des Menübands verändern, was vorher nur Entwicklern möglich war. Darüber hinaus hat Microsoft das Menüband aufgeräumt und alle Funktionen, die nicht der Bearbeitung von Dokumentinhalten dienen, in die sogenannte *Backstage-Ansicht* ausgelagert.

Ein weiteres Highlight von Excel 2007 war eine mächtig aufgebohrte *Diagramm-Engine*, die erstmals 3D- und Rendering-Effekte in „TV-Qualität“ zu bieten hatte. Zudem erschienen mit den *SmartArt*-Diagrammen und den *Zelldiagrammen der Bedingten Formatierung* zwei völlig neue Diagrammtypen am Excel-Horizont. Mit den *Sparklines*-Diagrammen kam in Excel 2010 ein weiterer Diagrammtyp hinzu, der die Visualisierung von Trends auf buchstäblich engstem Raum – innerhalb einer Zelle nämlich! – erlaubt. Excel 2016 erweitert den Diagrammfundus nun um die sechs Diagrammtypen *Wasserfall*, *Histogramm*, *Pareto*, *Kastengrafik*, *Treemap* und *Sunburst*. Die eignen sich insbesondere für die Visualisierung von finanziellen, statistischen oder hierarchisch strukturierten Daten und sind durchweg sehr ansprechend und professionell gestaltet.

Aber auch unter der Oberfläche hat sich seit der Ablösung der Excel-Version 2003 – der letzten, die man wegen ihrer altbekannten Menüs und Symbolleisten noch als „Microsoft-kompatibel“ bezeichnen konnte – einiges getan. So bescherte Excel 2007 den Anwendern u. a. ein neues, *XML-basiertes Dateiformat* und Arbeitsblätter mit drastisch erhöhter Kapazität.

Seit der Version 2010 gibt es Excel auch in einer *64-Bit-Version*. Die kann zwar den vollen Arbeitsspeicher eines Windows-Rechners nutzen (was noch größere Arbeitsblätter erlaubt), verweigert einer großen Zahl von vorhandenen VBA-Lösungen aber die Zusammenarbeit.

Mit den *Apps für Office* schließlich stellte Excel 2013 ein radikal neues und besonders anwendungsfreundliches Erweiterungsmodell vor, das integrierte Webanwendungen an die Stelle von klassischen VBA-Makros und Add-ins setzen will. Die Ablösung ist zwar bislang nicht vollzogen (und wird es so bald wohl auch nicht sein), das Konzept ist trotzdem vielversprechend und sollte von keinem Office-Entwickler ignoriert werden. Excel 2016 gibt dem Kind nun allerdings einen neuen Namen, nämlich *Office-Add-ins*.

■ 2.1 Die Benutzeroberfläche RibbonX

Die „RibbonX“ oder auch „Fluent“ genannte Oberflächenarchitektur der Excel-Versionen ab 2007 präsentiert die vielen Programmfunktionen des Kalkulationsprogramms deutlich übersichtlicher und in aufgabenorientierter Form, was insbesondere Einsteigern zugutekommt. Umsteiger von älteren Excel-Versionen müssen dagegen mit einer wesentlich längeren Eingewöhnungszeit rechnen, weil sie zuerst einmal vieles vergessen müssen, was sie über die bisherige Excel-Bedienung wussten. Die Mühe lohnt sich allerdings, weil es sich mit RibbonX tatsächlich effektiver arbeiten lässt.

Das Menüband

Die RibbonX-Oberfläche schickt Menüs und Symbolleisten in den Ruhestand. An deren Stelle tritt nun das sogenannte *Menüband*, das einen Satz von aufgabenorientierten *Befehlsregisterkarten* enthält. Aufgabenorientiert bedeutet, dass jede Registerkarte nur die Befehle bereitstellt, die für die Erledigung einer bestimmten Aufgabe nützlich sind. So verfügt Excel unter anderem über Registerkarten für die Gestaltung von Arbeitsblättern (SEITENLAYOUT) oder den Umgang mit Rechenfunktionen (FORMELN).



BILD 2.1: Neben neun ständig sichtbaren Registerkarten enthält das Menüband auch solche, die nur bei Bedarf (kontextabhängig) angezeigt werden.

Neben einem festen Inventar an Registerkarten zeigt das Menüband auch kontextbezogene Exemplare an. Die erscheinen ganz automatisch, wenn der Anwender bestimmte Objekte – ein Diagramm oder eine Grafik etwa – bearbeitet.

Die Befehle auf den Registerkarten bilden thematisch geordnete *Befehlsgruppen* und stehen zumeist in Form von unterschiedlich großen Schaltflächen oder Listefeldern zur Verfügung. Das erfordert zwar deutlich mehr Platz als eine schlanke Menüzelle, bringt aber wesentlich mehr Übersicht und spürbar verkürzte Mauswege mit sich, da es nur eine „Klickebene“ gibt, in der stets alle Befehle sichtbar sind. Das bislang übliche „Abtauchen“ in tief verschachtelte Menüstrukturen entfällt also.

Sollte sich das Platzangebot auf dem Bildschirm durch eine Änderung der Bildschirmauflösung oder durch das Verkleinern des Programmfensters verringern, zeigt Excel dennoch weiterhin alle Befehle der jeweiligen Registerkarte an. Das gelingt dem Programm durch intelligente „Sparmaßnahmen“, die von einer schrittweisen Reduzierung der Schaltflächengrößen über das Weglassen von Beschriftungen bis hin zum Zusammenfassen von Befehlen zu Listefeldern reichen. Bis zur Programmversion 2007 war dieses Feature nur Excel-eigenen Registerkarten vorbehalten. Seit Excel 2010 können nun auch benutzerdefinierte Registerkarten, die ihr „virtuelles Leben“ Programmcode oder dem neuen Oberflächeneditor verdanken, an der *Automatischen Skalierung* teilnehmen.

Seit Excel 2010 ist es ebenso möglich, Registerkarten programmgesteuert zu aktivieren. Makros oder Add-ins können also jederzeit auf ihre eigene Registerkarte „umschalten“, um dem Anwender die eigenen Funktionen bedarfsgerecht zu präsentieren.

Kataloge und Livevorschau

Neben Schaltflächen, Listefeldern und anderen Steuerelementen (*RibbonX-Controls*) findet man in den Registerkarten des Menübands auch sogenannte *Kataloge*. Damit kann der Anwender viel Zeit bei der Bewältigung von Auswahl- und Formatierungsaufgaben sparen, weil er sich nicht mehr im Versuch-und-Irrtum-Prinzip mit Dialogfeldern herumschlagen muss, um ein bestimmtes Resultat zu erzielen. Stattdessen zeigt ihm das Katalog-Control ein Vorschaubild der zu erwartenden Ergebnisse an, sodass er sich nur noch das gewünschte aussuchen muss.

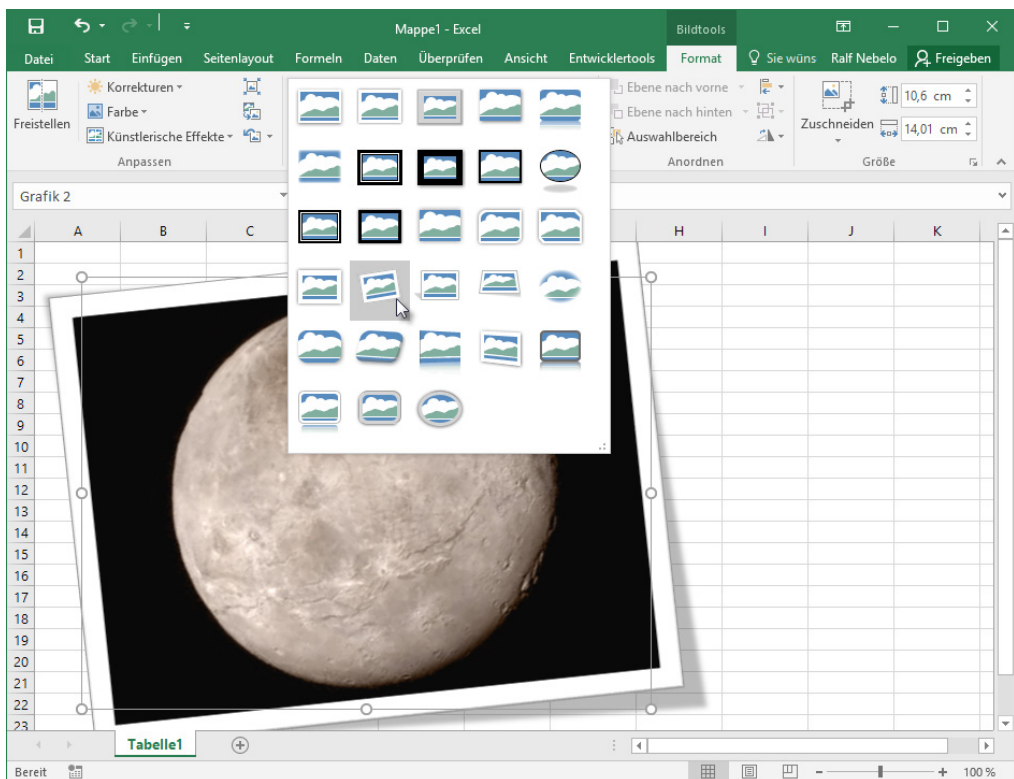


BILD 2.2: Kataloge zeigen per Vorschaubild und zudem „live“ im Dokument an, wie sich ein Befehl auswirken wird.

Weil die angezeigte Miniaturvorschau unter Umständen nicht ausreicht für eine Beurteilung, überträgt Excel die Einstellungen, die dem aktuell gewählten Katalogelement entsprechen, dynamisch auf das Arbeitsblatt. Mit dieser *Livevorschau* kann man die Auswirkungen einer möglichen Formatierungsänderung also schon vorab und im Kontext des Dokuments bewerten.

Neue und alte Symbolleisten

Die *Minisymbolleiste* hilft ebenfalls bei der Vermeidung unnötiger Dialogfeldeingaben und Mauswege. Klickt man einen Zellwert mit der rechten Maustaste an, so erscheint darüber ein halbtransparentes Tool, das einem die wichtigsten Formatierungswerkzeuge zur Verfügung stellt. Damit kann man dann „schnell mal eben“ Schriftart und -größe verändern, Text- und Hintergrundfarben ändern, Zahlenformate einstellen und dergleichen mehr.



BILD 2.3:

Die Minisymbolleiste fasst die wichtigsten Formatierungsoptionen an einem Ort zusammen.

Trotz ihres Namens ist die Minisymbolleiste keine Symbolleiste im herkömmlichen Sinn, ihr Befehlsvorrat lässt sich also nicht den Wünschen des Anwenders entsprechend verändern. Das funktioniert nur bei der *Symbolleiste für den Schnellzugriff*, die oberhalb des Menübands zu finden ist und in die man mithilfe des neuen Oberflächeneditors (DATEI | OPTIONEN | SYMBOLLEISTE FÜR DEN SCHNELLZUGRIFF) jeden verfügbaren Excel-Befehl und auch jedes Makro integrieren kann.

Die „klassischen“ Symbolleisten (und Menüs) sind ausnahmslos aus Excel 2007 und seinen Nachfolgern entfernt worden und lassen sich auch nicht wieder zum Vorschein bringen. Das einzige bedienungstechnische Zugeständnis an alte Excel-Hasen ist die Tatsache, dass die meisten Tastenkürzel weiterhin funktionieren. Das gilt allerdings im vollen Umfang nur für Funktionstasten sowie Kombinationen mit der Strg-Taste. Die menüorientierten Tastenkürzel, bei denen zuerst die Alt-Taste gedrückt werden muss, sind zum größten Teil unwirksam.



Tipp

Auch wenn Menüs und Symbolleisten aus Excel entfernt wurden, so kann man sie doch mit etwas Handarbeit und XML-Code nachbilden. Wie das im Detail funktioniert, wird ausführlich im Abschnitt 8.2.5 beschrieben. Wer sich die Arbeit sparen möchte, verwendet einfach unser Add-in *KlassikMenü.xlam*, das im Unterordner 8 der Beispieldateien zu finden ist.

Die Backstage-Ansicht

Die seinerzeit völlig neu gestaltete Oberfläche von Excel 2007 wies mindestens einen größeren Konstruktionsfehler auf: das *Office-Menü*. Viele Anwender vermuteten hinter dem „bunten Blümchen“ links oben nur ein Schmuckelement und wären von sich aus kaum auf die Idee gekommen, darauf zu klicken. Und wer es dennoch getan hat, dem bot sich ein funktionaler Gemischtwarenladen, in dem man relativ wahllos alles hineingepackt zu haben schien, was bei der Bestückung des Menübands (das damals noch „Multifunktionsleiste“ hieß) übrig geblieben war.

Mit Excel 2010 trat dann die *Backstage-Ansicht* an die Stelle des Office-Menüs. Sie integriert sich optisch geschickt über den farbig markierten Registerreiter *DATEI* in das Menüband. Nach einem Klick auf den Registerreiter öffnet sich ein sehr spezieller Oberflächenbereich. Darin finden sich links einzelne Kommandos wie *NEU*, *SPEICHERN* oder *SCHLIESSEN*, die beim

Anklicken jeweils einen sogenannten *Tab* auf der rechten Seite zum Vorschein bringen. Tabs sind im Grunde auch nur Registerkarten, die aber im Unterschied zu ihren Menüband-Kollegen die gesamte Fläche des Excel-Fensters einnehmen können.

Inhaltlich präsentiert sich die Backstage-Ansicht wesentlich aufgeräumter als das Office-Menü. Es enthält ganz konsequent nur solche Befehle, die das Dokument als Ganzes betreffen oder darüber hinausgehen, während sich sämtliche Funktionen für die Bearbeitung von Dokumentinhalten im Menüband konzentrieren.

Diese konsequente Aufgabenteilung sollten auch Entwickler beachten, wenn sie eigene Makrolösungen in die Backstage-Ansicht integrieren. Anwender können den Inhalt der Backstage-Ansicht *nicht* verändern.

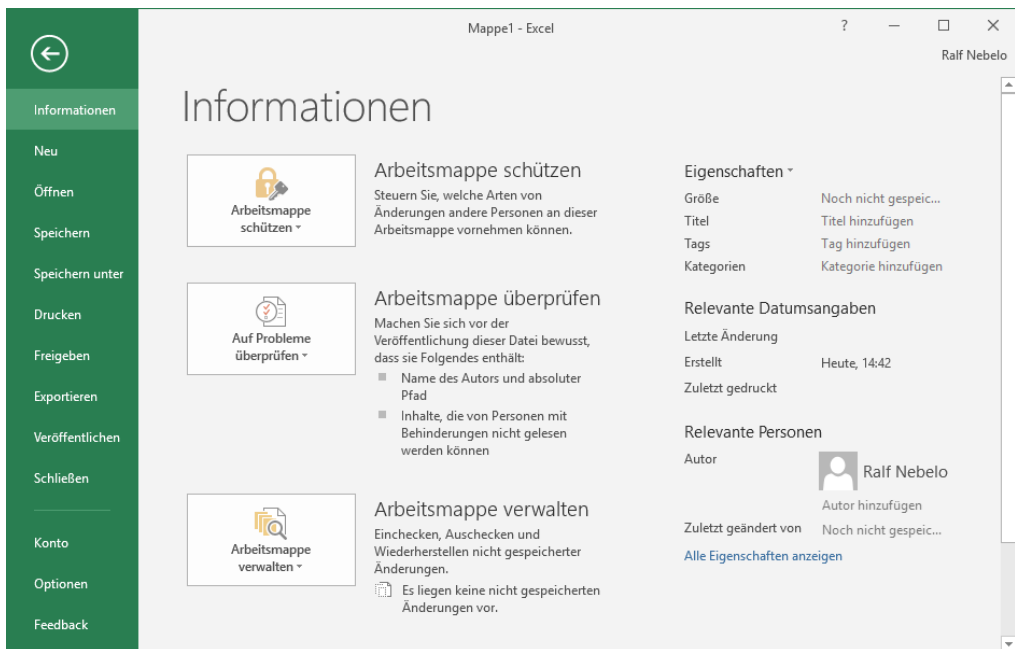


BILD 2.4: Die mit Excel 2010 eingeführte Backstage-Ansicht enthält ausschließlich Kommandos, die das Dokument als Ganzes betreffen.

■ 2.2 Neue Programmfunktionen

Neue Diagramm-Engine

Seit der Version 2007 verwendet Excel eine seinerzeit völlig neu konstruierte Diagramm-Engine, die auch in den übrigen Office-Anwendungen wie Word oder PowerPoint zum Einsatz kommt. Damit lassen sich nüchterne Zahlen in informative Grafiken überführen. Das wichtigste Merkmal der neuen Diagrammfunktion ist eine deutlich verbesserte Ausgabequalität. Das zeigt sich insbesondere bei den dreidimensionalen Diagrammtypen.

Über die kontextsensitiven Befehlsregister ENTWURF, LAYOUT und FORMAT lassen sich aber auch einfache Kreis- und Säulendiagramme beispielsweise mit echten 3D-Effekten, sauber gerenderten Schattenverläufen, Transparenzeffekten oder geglätteten Kanten ausstatten. Bei Bedarf ist sogar ein Griff in die Trickkiste erlaubt, wo „glühende Farben“ und andere TV-verdächtige Spezialeffekte auf ihren Einsatz warten.

SmartArt-Diagramme

Die *Schematischen Darstellungen* von Excel 2003 gibt es nicht mehr. Stattdessen kann der Anwender nun *SmartArt*-Diagramme verwenden, um Prozesse, Abläufe oder Beziehungen grafisch darzustellen. Im Gegensatz zum erwähnten Vorgänger stehen jetzt wesentlich mehr und wesentlich aufwendiger gestaltete Diagrammtypen zur Auswahl, die sich sehr unkompliziert beschriften, konfigurieren und über das kontextsensitive Befehlsregister ENTWURF nachbearbeiten lassen.

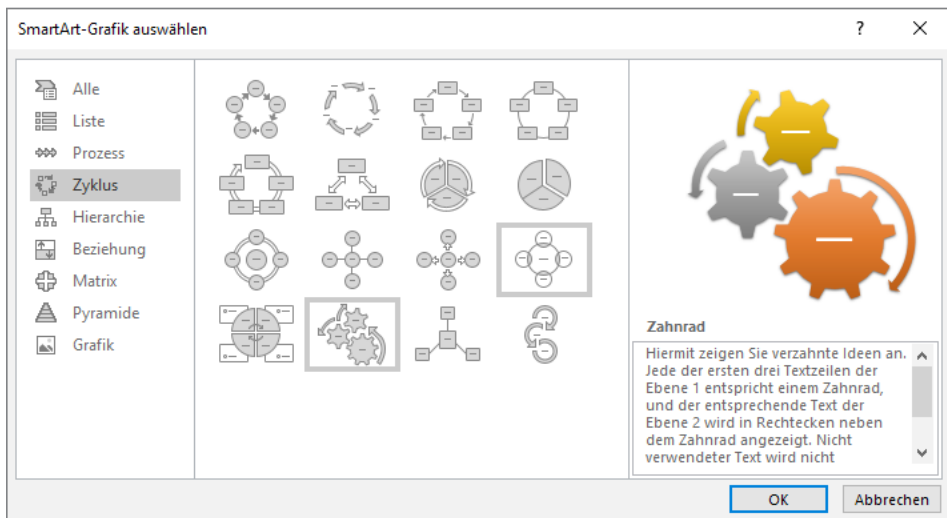
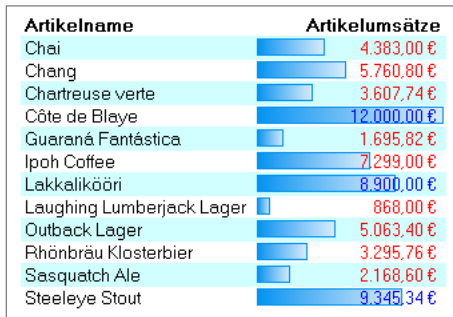


BILD 2.5: SmartArt-Diagramme eignen sich insbesondere für die Visualisierung von Prozessen und Hierarchien.

Da die SmartArt-Ergebnisse professionellen Standards entsprechen, können sich viele Excel-Eigner die Anschaffung von separaten Diagramm-Tools wie Visio beispielsweise sparen. Seit der Excel-Version 2010 lassen sich die „smarten Diagramme“ programmatisch erstellen oder verändern.

Zelldiagramme der Bedingten Formatierung

Mit der stark erweiterten *Bedingten Formatierung* kann man nicht nur wertabhängige Formatierungen durchführen, die grafische Darstellung von Zahlen unmittelbar im Arbeitsblatt ist ebenfalls machbar. Der Trick besteht darin, die Zellen einer markierten Zahlenreihe mit halbtransparenten, farbigen Balken zu hinterlegen. Dabei entspricht die Länge jedes einzelnen Balkens dem prozentualen Wert der zugehörigen Zelle im Vergleich zum größten Wert innerhalb der Markierung. Dadurch kann man Trends, Muster oder abweichende Einzelwerte auch ohne Erstellung eines Diagramms sofort erkennen.

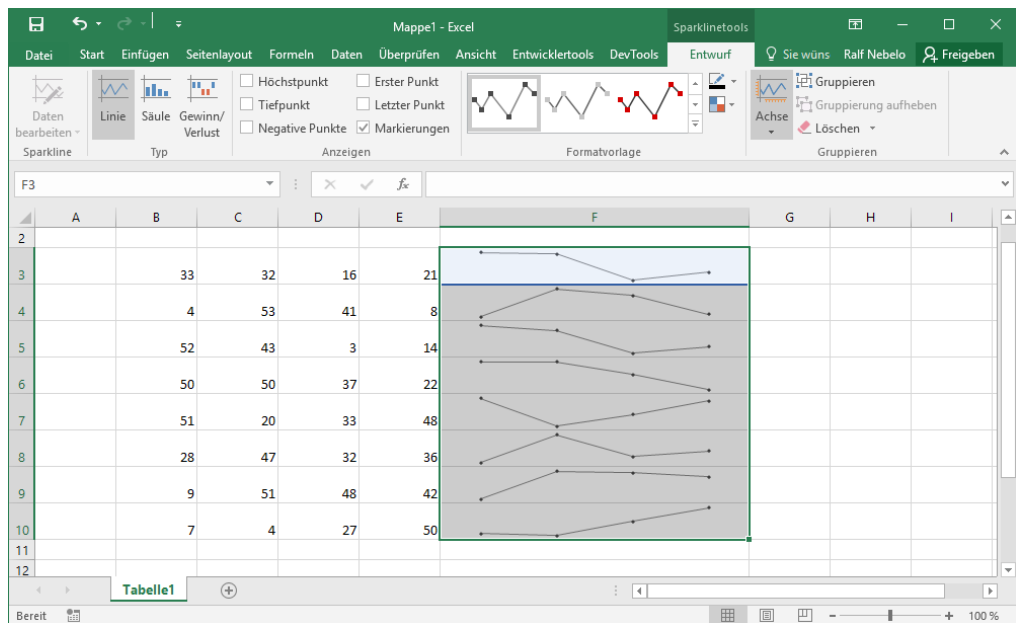
**BILD 2.6:**

Datenbalken-Diagramme veranschaulichen den relativen Wert einer Zahl direkt in der Tabelle.

Anstelle solcher *Datenbalken* kann man auch *Farbskalen* oder *Symbolsätze* verwenden. Im letzteren Fall verdeutlicht Excel die relativen Zellwertigkeiten nicht mit Balken, sondern mit vorangestellten Pfeilsymbolen oder Ampellichtern beispielsweise. Das erlaubt zwar nur wenige Abstufungen (maximal fünf), lässt den Anwender aber Entwicklungen besonders schnell erkennen.

Sparklines-Diagramme

Sparklines sind Linien- oder Balkendiagramme, die aber im Unterschied zu den „ausgewachsenen“ Excel-Diagrammen vollständig in eine einzige Zelle passen. Dadurch lassen sich die Minidiagramme in unmittelbarer Nähe zu ihren Quelldaten platzieren, was sie zum optimalen Mittel für die Visualisierung von Trends macht – von Wirtschaftszyklen etwa, Ausgabenentwicklungen oder saisonalen Auf- und Abschwüngen.

**BILD 2.7:** Sparklines-Diagramme eignen sich ideal für die datennahe Visualisierung von Trends.

Aufgrund der geringen Größe sind die Gestaltungsmöglichkeiten von Sparklines-Diagrammen natürlich etwas eingeschränkt. Der Anwender hat die Wahl zwischen den drei Typen „Linie“, „Säule“ und „Gewinn/Verlust“. Alle Diagrammtypen bieten die Möglichkeit, Maximal- und Minimalwerte sowie negative Zahlen farbig hervorzuheben. Dabei lässt sich die Lage des Nullpunkts durch eine horizontale Achse kennzeichnen. Beim Diagrammtyp *Linie* besteht darüber hinaus die exklusive Möglichkeit, farbige Markierungen für alle Datenpunkte sichtbar zu machen.

Wasserfall & Co – neue Diagrammtypen in Excel 2016

Nach Zell-, Sparkline- und SmartArt-Diagrammen gibt es mit Excel 2016 nun auch wieder Neuerungen im Bereich der regulären Diagramme zu vermelden. Dabei handelt es sich um die sechs neuen Diagrammtypen *Wasserfall*, *Histogramm*, *Pareto*, *Kastengrafik*, *Treemap* und *Sunburst*, die durchweg sehr ansprechend und modern gestaltet sind.

Während die ersten vier sich insbesondere für die Visualisierung (und Analyse) von finanziellen und statistischen Daten eignen, stehen mit Treemap und Sunburst wahre Spezialisten für die übersichtliche Darstellung von hierarchisch strukturierten Zahlentabellen zur Verfügung.

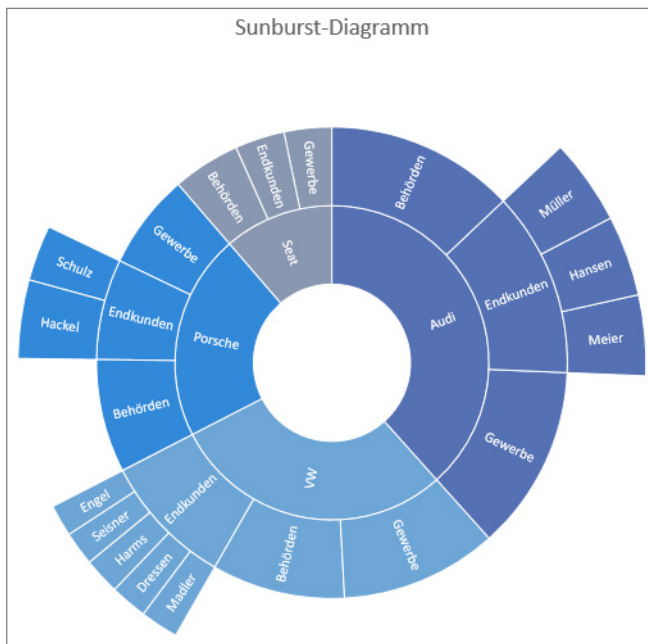


BILD 2.8:

Die sechs neuen Diagrammtypen von Excel 2016 sind wahre Spezialisten für die Visualisierung von finanziellen, statistischen oder hierarchisch strukturierten Daten.

Größere Arbeitsblätter und Dokumente

Seit Excel 2007 beträgt die Maximalgröße für Arbeitsblätter eine Million Zeilen und 16 000 Spalten (gegenüber 65 536 Zeilen und 256 Spalten in Excel 2003). Die Größe einer Arbeitsmappe darf zwei GByte betragen. In der 64-Bit-Version von Excel, die mit der Versionsnummer 2010 eingeführt wurde, ist die Größe einer Arbeitsmappe sogar nur noch durch den vorhandenen Arbeitsspeicher begrenzt.

Fragt sich jedoch, wem derart „unendliche Speicherwelten“ nützen. Klar ist: Wer in einem Excel-Dokument ähnlich viele Zahlen und Daten bunkert, wie sie für die HD-Wiedergabe eines abendfüllenden Spielfilms nötig sind, macht irgendetwas ganz furchtbar verkehrt (und sollte vielleicht dringend über die Anschaffung einer richtigen Datenbank-Software nachdenken).

Das Dateiformat Open XML

Seit der Version 2007 werden Arbeitsmappen und andere Excel-Dokumente standardmäßig nicht mehr binär gespeichert, sondern in einem Klartextformat, das auf der Auszeichnungssprache XML (steht für „Extensible Markup Language“) beruht. Dieses *Open XML* genannte Format unterscheidet sich aber grundlegend von dem XML-Format, das schon in Excel 2003 zur Auswahl stand.

Für den Anwender zeigt sich die Existenz des neuen Dateiformats eigentlich nur in den neuen Dateiendungen, die sich um ein angehängtes „X“ von den bisherigen unterscheiden. So werden nun Arbeitsmappen beispielsweise nicht mehr in XLS-Dateien gespeichert, sondern in XLSX-Dateien. Vorlagen sichert man anstatt in XLT- nun in XLTX-Dateien.

Das gilt allerdings nur, solange Arbeitsmappen und Vorlagen *keine* VBA-Makros enthalten. Ist das der Fall, tritt ein „M“ an die Stelle des angehängten „X“. Für Arbeitsmappen mit Makros lautet die korrekte Dateiendung dann XLSM, für Vorlagen mit Makros XLTM. Excel-Add-ins enthalten ja grundsätzlich Makros, weshalb es hier nur eine Dateiendung gibt, die XLAM lautet.

Beim Speichern von Excel-Dokumenten stehen die alten Dateiendungen aus Kompatibilitätsgründen weiterhin zur Wahl. Dann erfolgt die Speicherung allerdings im alten Binärformat und unter Verzicht auf Dokumentelemente, die mit Excel 2007 oder danach erst eingeführt wurden. So lassen sich SmartArt- oder Sparklines-Diagramme beispielsweise nur im Open-XML-Format konservieren.



Hinweis

Open-XML-Dateien sind keine Dateien im eigentlichen Sinn, sondern „getarnte“ ZIP-Archive. Wer sich den (aus vielen XML-Dateien bestehenden) Inhalt eines solchen Archivs anschauen möchte, kann die Endung „.zip“ an den Dateinamen anhängen (via *Umbenennen*-Befehl) und das Archiv dann per Doppelklick im Windows-Explorer öffnen.

■ 2.3 Office-Add-ins

Wer die Oberfläche von Excel 2016 mit früheren Versionen vergleicht, dürfte ziemlich schnell über den neuen Befehl **MEINE ADD-INS** im Menüband **EINFÜGEN** stolpern.

Dahinter verbirgt sich ein mit Excel 2013 eingeführtes Erweiterungskonzept, das ähnlich radikal ist wie der mit Excel 2007 vollzogene Wechsel von der Befehlsleistenoberfläche zum

Menüband. Die Idee dahinter: Klassische VBA-Makros und Add-ins sollen langfristig abgelöst und durch trendige „Office-Add-ins“ (ehemals „Office-Apps“) ersetzt werden.

Diese Minianwendungen klinken sich direkt in die Benutzeroberflächen von Excel, Word und diversen anderen Office-2016-Anwendungen (und deren Web-App-Versionen) ein. Sie erscheinen wahlweise im Aufgabenbereich („Task Pane“), also am rechten Rand des Programmfensters, oder als abgegrenzter Bereich innerhalb des Dokuments. Von dort aus verfolgt das typische Office-Add-in alles, was der User gerade eintippt oder markiert, und beschafft ihm dazu passende Infos, Übersetzungen, Karten oder Grafiken aus der vielzitierten Cloud oder dem Internet.

Das klingt erst mal nicht sonderlich spannend, da VBA-Makros und vor allem Add-ins, die mit den *Visual Studio Tools for Office* (siehe Abschnitt 15.8) programmiert wurden, so etwas auch können. Den Office-Add-ins fällt die Webrecherche allerdings ungleich leichter, da sie selbst hundertprozentige Webanwendungen sind, die mit einschlägigen Standardtechnologien wie HTML, XML oder JavaScript realisiert werden.

Vor allem aber muss man sich Office-Add-ins nicht erst umständlich beschaffen und anschließend installieren. Man findet sie stets in einem Online-Shop namens „Office Store“, den man im Unterschied zu den klassischen Vertriebskanälen direkt aus Word & Co heraus erreichen kann – Mausklick respektive Finger-Touch genügt. Der Office Store bündelt das weltweite (!) Angebot an Office-Add-ins und macht es transparent. Hat man das passende Add-in darin gefunden, bringt man dieses mit ein paar weiteren Mausklicks an den Start.

Wie einfach die Beschaffung eines Office-Add-ins ist und was man damit anstellen kann, davon sollte sich jeder Excel-Anwender selbst überzeugen. Als nützliches und noch dazu kostenloses Anschauungsobjekt empfehlen wir „Bing Maps“, ein Office-Add-in, das Städte- und Ländernamen innerhalb eines Arbeitsblatts erkennt und dazu passende Karten liefert. Zwecks Einrichtung des Add-ins ...

- ... wechseln Sie in das EINFÜGEN-Register und wählen dort den Befehl MEINE ADD-INS. Sollte „Bing Maps“ nicht bereits im *Office-Add-ins*-Dialog angezeigt werden, klicken Sie auf WEITERE ADD-INS IM OFFICE STORE SUCHEN am unteren Rand des Dialogfelds.
- Geben Sie „Bing Maps“ in das Suchfeld ein, und drücken Sie die Eingabetaste. Erscheint das Add-in schließlich in den Store-Ergebnissen, klicken Sie darauf.
- Wenn die Detailseite des Add-ins erscheint, klicken Sie auf HINZUFÜGEN.

Sofern Sie bereits mit Ihrem Microsoft-Konto angemeldet waren, wird das Add-in nun umgehend geladen und eingerichtet. Zum Testen tippen Sie die folgenden Inhalte in vier untereinanderliegende Zellen des Arbeitsblatts (die erste Zelle muss stets eine Überschrift enthalten):

Städte
Hamburg
Köln
München

Anschließend markieren Sie die Zellen und klicken auf das ORTE ANZEIGEN-Symbol in Bing Maps. Die angezeigte Karte können Sie nach Belieben vergrößern, verkleinern, verschieben oder darin zoomen. Nicht schlecht, oder?

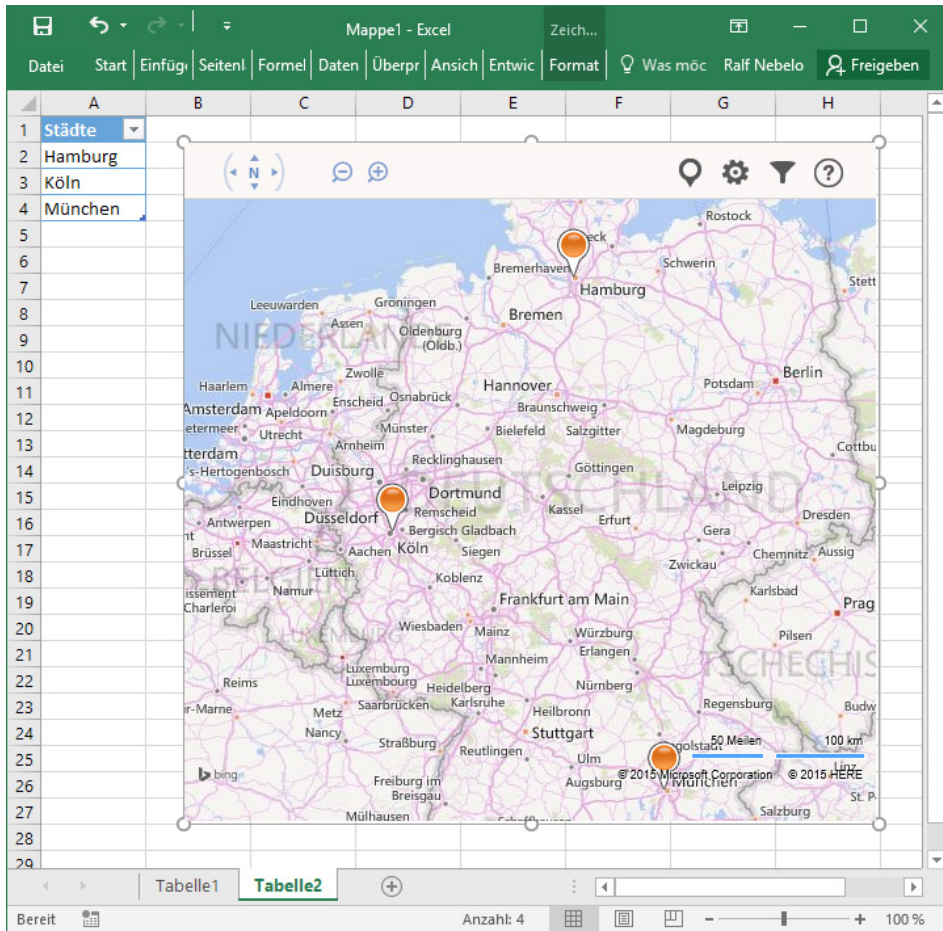


BILD 2.9: Das Office-Add-in „Bing Maps“ erkennt Inhalte von Arbeitsblättern und liefert dem Anwender passende Karten dazu.

■ 2.4 Neues in Sachen Programmierung

Angesichts des mit Excel 2007 völlig neu gestalteten Erscheinungsbilds könnte man meinen, Microsoft hätte das Kalkulationsprogramm von Grund auf neu programmiert. Was aber nicht stimmt. Tatsächlich basieren auch Excel 2007 und seine Nachfolger noch auf dem *Component Object Model* oder kurz: COM. Das bedeutet grob verkürzt nichts anderes, als dass man auch die neuesten Excel-Versionen immer noch über COM-fähige Entwicklungstools programmieren kann. Und da steht das „eingebaute“ und somit kostenlose VBA immer noch leistungsmäßig in der ersten Reihe – was die meisten Office-Entwickler wohl freuen dürfte.

Es gibt aber zunehmend mehr (und zumeist anspruchsvollere) Programmieraufgaben, die sich mit VBA entweder gar nicht oder nur noch teilweise lösen lassen. Für einen Teil dieser Fälle benötigt man nur einen Texteditor und ein paar grundlegende XML- respektive JavaScript-Kenntnisse – was das Kapitel 15.9 zum neuen und sehr interessanten Thema *Office-Add-ins* belegt. Andere Problemfälle verlangen nach einem ebenso mächtigen wie leider auch teuren Entwicklungswerkzeug, das sich *Visual Studio Tools for Office* (siehe Abschnitt 15.8) nennt.

2.4.1 Kompatibilitätskrücke Add-ins-Register

Das Festhalten an der Programmierschnittstelle COM bedeutet auch, dass zahllose Excel-Lösungen, die für frühere Versionen des Kalkulationsprogramms entwickelt wurden und in Gestalt von „makrohaltigen“ Arbeitsmappen sowie VBA- und COM-Add-ins vorliegen, weiterhin funktionieren. Das gilt sogar für Lösungen, die sich über Menüerweiterungen oder eigene Symbolleisten in die Oberfläche früherer Excel-Versionen eingeklinkt hatten – obwohl es diese Oberfläche gar nicht mehr gibt! Wie ist das möglich?

Antwort: weil die Excel-Versionen ab 2007 zwar weiterhin die *CommandBars*-Objekte (siehe Abschnitt 8.1.3) unterstützen, entsprechende Programmieranweisungen aber nicht mehr in *echte* Menüs und Symbolleisten umsetzen, sondern in Nachbildungen, die zu hundert Prozent aus den Elementen der neuen RibbonX-Oberfläche bestehen. Und diese „simulierten“ Befehlsleisten – so der Oberbegriff von Menü- und Symbolleisten – werden auch nicht mehr an den ursprünglich vorgesehenen Bildschirmpositionen angezeigt, sondern in einer einzigen Befehlsregisterkarte namens *ADD-INS*. Die bildet somit eine Art Kompatibilitätskrücke, welche die Bedienbarkeit älterer Excel-Lösungen grundsätzlich sicherstellt.



BILD 2.10: Menüs und Symbolleisten zeigt Excel seit der Version 2007 nur noch im Add-ins-Register an.

Dass es sich beim *ADD-INS*-Register in der Tat nur um einen Behelf handelt, zeigt sich an deutlichen Einschränkungen in Sachen Handling. So lassen sich die hier angezeigten Befehlsleistenattrappen beispielsweise nicht mehr verschieben, andocken oder manuell anpassen, und zugeordnete Tastenkürzel für den schnellen Aufruf von selbst gebauten Menükommandos funktionieren auch nicht mehr. Angesichts des radikal erneuerten Oberflächenkonzepts erscheint das *ADD-INS*-Konstrukt aber dennoch als guter Kompromiss zwischen Fortschritt und Kompatibilität mit vorhandenen Excel-Lösungen.

2.4.2 Zu- und Abgänge im Objektmodell

Viele neue Excel-Features spiegeln sich auch im Objektmodell des Kalkulationsprogramms wider. Und zwar in Form von neuen oder erweiterten Objekten, die sich mit VBA nutzen lassen.

Bei den Neuzugängen ist unter anderem das *Databar*-Objekt erwähnenswert, das die Programmierung der oben vorgestellten Zelldiagramme der Bedingten Formatierung ermöglicht (siehe Abschnitt 10.5). Die Darstellungstaleute der erweiterten Diagramm-Engine lassen sich unter anderem über die neuen Chart-Methoden *ChartStyle* und *ApplyLayout* (siehe Abschnitt 10.2.2) wecken. Auch SmartArt-Diagramme kann man jetzt per VBA-Code erzeugen und verändern – die *AddSmartArt*-Methode der *Shapes*-Auflistung und das Objekt *SmartArt* machen's möglich (siehe Abschnitt 10.7.1). Die *SparklineGroups*-Auflistung von *Range*-Objekten und das *SparklineGroup*-Objekt erlauben es, Sparklines-Diagramme programmgesteuert zu erschaffen (siehe Abschnitt 10.6.1).

Neue *Shape*- und *Chart*-Objekte machen den programmierten Zugriff auf die sechs mit Excel 2016 eingeführten Diagrammtypen *Wasserfall*, *Histogramm*, *Pareto*, *Kastengrafik*, *Treemap* und *Sunburst* möglich (siehe Kapitel 10.8). Über das neue *WorkbookQuery*-Objekt schließlich kann der Entwickler alle Möglichkeiten der jetzt voll integrierten Power-Query-Technologie für jede Art von Datenbankabfrage nutzen (Kapitel 12.2.1).

Den Erweiterungen der Objektbibliothek stehen aber auch einige „ausgeblendete“, das heißt nicht mehr unterstützte Objekte, Methoden und Eigenschaften gegenüber. Zu den prominentesten und schmerzlichsten Abgängen gehört nach wie vor das *FileSearch*-Objekt, das noch in Excel 2003 die vergleichsweise simple Programmierung einer Dateisuchfunktion ermöglichte und in zahlreichen VBA-Lösungen zum Einsatz kam. Viele davon lassen sich nun wieder flottmachen, indem man sie auf eine selbst gebaute und dennoch weitgehend „syntaxkompatible“ Ersatzklasse namens *SearchFile* (siehe Abschnitt 4.5.5) umstellt.

Die folgende Internetseite [Link 3] liefert Ihnen eine vollständige Auflistung aller Objektmodelländerungen in Excel 2007:

<http://msdn2.microsoft.com/en-us/library/bb149067.aspx>

Für Excel 2010 finden Sie die gleichen Informationen hier [Link 4]:

<http://msdn.microsoft.com/en-us/library/ee836187.aspx>

Für Excel 2013 gibt es diese Info-Adresse [Link 47]:

<http://msdn.microsoft.com/library/office/ff837594.aspx>

Für Excel 2016 schließlich bietet die MSDN-Library bislang keine vergleichbare Dokumentation aller Zu- und Abgänge in Excels Objektmodell. Wer sich ohnehin mehr für die Programmierung von Office-Add-ins interessiert, dem bietet der folgende Link eine gute Übersicht der diesbezüglichen Neuerungen [Link 51]:

<https://blogs.office.com/2015/09/28/whats-new-in-office-2016-for-developers>

2.4.3 Anpassen der Benutzeroberfläche

Dank der *CommandBars*-Objekte lassen sich – wie schon erwähnt – auch in Excel 2016 noch Menüs und Symbolleisten anlegen, die dann jedoch ausschließlich in der Befehlsregisterkarte *ADD-INS* erscheinen. Wer aber in die Zukunft denkt, sollte sich von *CommandBars* und dem *ADD-INS*-Register verabschieden und die Funktionen seiner Lösung lieber gleich in die neuen Elemente der Excel-Oberfläche integrieren. Bei diesem Vorhaben kommt man mit VBA allein aber nicht mehr zurecht.

Das liegt daran, dass diese Elemente – Menüband, Symbolleiste für den Schnellzugriff und Backstage-Ansicht – nicht mehr im Objektmodell von Excel abgebildet werden. Folglich hat man aus VBA heraus auch (fast) keine Möglichkeit, das Aussehen dieser Oberflächenelemente zu verändern. Das lässt sich jetzt nur noch über XML-Code erledigen, der in die Dokumentdatei der Excel-Lösung eingebettet ist und ausschließlich beim Öffnen ausgeführt wird.

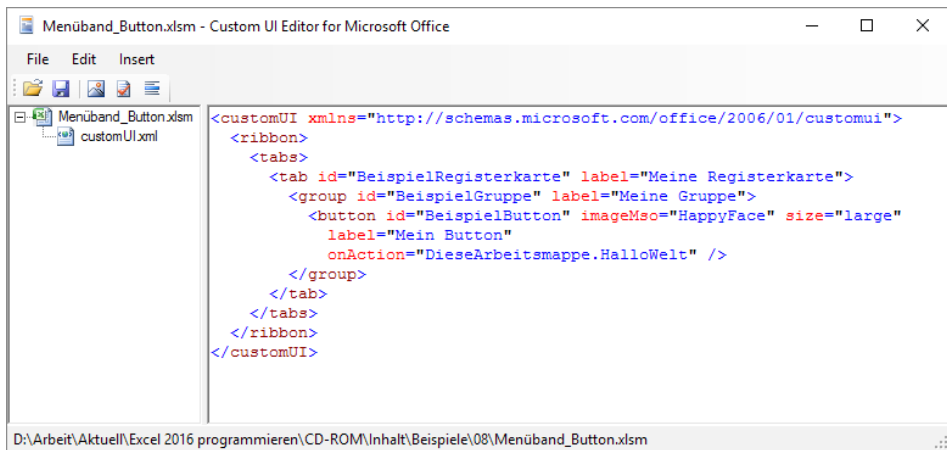


BILD 2.11: XML-Code bestimmt jetzt das Design von Oberflächenanpassungen; für die Funktion sind VBA-Makros zuständig.

Die XML-Anweisungen bestimmen aber nur das „Design“ der gewünschten Oberflächenanpassung, indem sie dem Menüband beispielsweise eine neue Befehlsregisterkarte hinzufügen und darin eine neue Schaltfläche anlegen. Für die Funktion – also das, was beim Anklicken dieser Schaltfläche geschieht – ist nicht mehr der XML-Code, sondern eine sogenannte *CallBack*-Routine zuständig. Und dabei handelt es sich um eine spezielle Form von VBA-Makro. Die Einzelheiten über das (gar nicht so schwierige) Anpassen von Menüband, Symbolleiste für den Schnellzugriff und Backstage-Ansicht sind ausführlich in den Abschnitten 8.2, 8.3 und 8.5 beschrieben.

Das „duale Konzept“ der Oberflächenprogrammierung – XML-Code für das Design, *CallBack*-Makros für die Funktion – gilt übrigens jetzt auch für die zahlreichen *Kontextmenüs*, die ja immer schon einen wesentlichen Bestandteil der Benutzeroberfläche bildeten. Diesbezügliche Details finden Sie im Abschnitt 8.4.

2.4.4 Die Grenzen von VBA

Neben der großen Zahl von Aufgaben, die sich mit VBA immer schon bewältigen ließen, erlaubt Excels bordeigene Programmiersprache nun also die programmierte Nutzung vieler neuer Funktionen, die mit den Excel-Versionen seit 2007 Einzug gehalten haben. Darüber hinaus kann man mit VBA den Funktionscode für eine individuelle Anpassung *aller* Oberflächenelemente (Menüband, Symbolleiste für den Schnellzugriff, Backstage-Ansicht und Kontextmenüs) schreiben. Trotz dieses ansehnlichen Leistungsspektrums aber gibt es Aufgaben, denen VBA nicht mehr gewachsen ist.

Individuelle Aufgabenbereiche und Smart Tags

Dazu gehört etwa das Anlegen von individuell gestalteten *Aufgabenbereichen*, die im Microsoft-Jargon – je nachdem, ob sie anwendungsweit oder nur auf Dokumentenebene verfügbar sind – als „Custom Task Panes“ oder „Action Panes“ bezeichnet werden. Die Entwicklung von maßgeschneiderten *Smart Tags* – jenen Minimenüs, die dem Anwender zu bestimmten Dokumentinhalten passende Aktionen zur Verfügung stellen – gelingt ebenfalls nicht mit Bordmitteln.

Wer sich nicht mehr mit diesen (und diversen anderen) Beschränkungen von VBA abfinden will (oder kann), hat derzeit nur eine ernsthafte Alternative und die heißt *Visual Studio Tools for Office* oder kurz VSTO. Was es damit auf sich hat, wird ausführlich in Abschnitt 15.8 beschrieben.

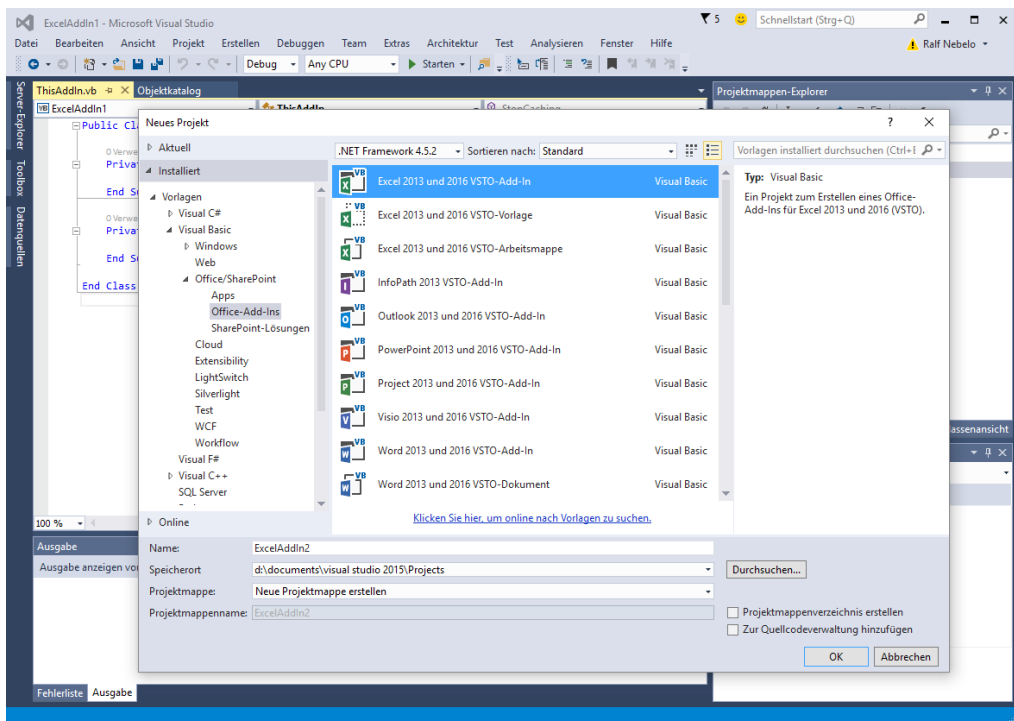


BILD 2.12: Die in Visual Studio integrierten Visual Studio Tools for Office sind eine professionelle, aber auch teure Alternative zu VBA.

64-Bit-Programmierung

Seit der Version 2010 wird Microsofts Kalkulationsprogramm sowohl in einer 32- als auch einer 64-Bit-Version angeboten. Letztere bietet nicht nur den eingangs erwähnten Vorteil, den gesamten Arbeitsspeicher eines 64-Bit-Windows-Systems nutzen zu können, sie soll auch Sicherheitslücken im Zusammenhang mit Pufferüberlaufen schließen und damit die Angriffsfläche für Viren und Würmer verringern können.

Den genannten Vorteilen stehen allerdings diverse Nachteile gegenüber, die die Lauffähigkeit vorhandener Excel-Erweiterungen betreffen. Welche das sind und wie Sie diese überwinden, erfahren Sie in Abschnitt 15.7.

Office-Add-ins (ehemals Apps für Office)

Die in Abschnitt 2.3 beschriebenen Office-Add-ins entziehen sich ebenfalls dem Einfluss von VBA. Wer sie entwickeln will, benötigt zwar nur einen besseren Texteditor, sollte aber grundlegende Erfahrungen in der Programmierung von Webanwendungen – denn darum handelt es sich im Grunde – mitbringen.

Kapitel 15.9 liefert Ihnen alle Infos, die Sie für den Einstieg in die Programmierung von Office-Add-ins benötigen.

■ 2.5 Probleme und Inkompatibilitäten



Vorsicht

Auch in Excel 2016 kann es beim Bearbeiten beziehungsweise Testen von VBA-Code zu Abstürzen kommen. Speichern Sie Ihre Arbeit daher regelmäßig! Beachten Sie auch, dass Excel nicht immer vollständig abstürzt. Manchmal kommt zwar die obligatorische Systemfehlermeldung, Excel bleibt jedoch im Speicher und blockiert weiter alle zuletzt geöffneten Dateien (ohne dass aber eine Möglichkeit besteht, diese noch zu speichern). Damit Sie wieder richtig weiterarbeiten können, müssen Sie Excel ganz beenden. Dazu verwenden Sie den Task-Manager von Windows, den Sie mit Strg+Alt+Entf zum Vorschein bringen. Es erscheint eine Task-Liste, aus der Sie Excel auswählen und gewaltsam beenden können.

Allgemeine Probleme

- VBA kennt noch immer keine Optimierungen bei der Auswertung von Bedingungen: Eine Abfrage in der Form $\text{If } x >= 0 \text{ And } \text{Sqr}(x) < 3$ führt bei negativen Zahlen in x zu einem Fehler. (Dieses Problem besteht in Visual Basic schon seit der ersten Version, d. h., wir haben die Hoffnung auf Besserung in diesem Punkt aufgegeben.)
- Bei allen Excel-Versionen gibt es Probleme mit dem Operator Is zum Objektvergleich. Dieser Operator sollte feststellen, ob zwei Variablen auf dasselbe Objekt verweisen. Leider funktioniert das nicht immer.

Probleme mit MS-Forms-Dialogen und -Steuerelementen

Excel und die meisten VBA-Kommandos sind blockiert, solange sich der Eingabecursor in einem MS-Forms-Steuerelement befindet, das in das Arbeitsblatt eingefügt wurde. Nur bei Buttons kann die Blockade durch *TakeFocusOnClick=False* verhindert werden. (Die Default-Einstellung lautet allerdings *True* und ist der Grund, weswegen es mit Buttons in Arbeitsblättern oft Probleme gibt. Die auftretenden Fehlermeldungen sind ohne jede Aussagekraft.)

Wenn im Arbeitsblatt auch andere Steuerelemente verwendet werden, muss der Eingabecursor per Programmcode (etwa durch *Worksheets(n).[A1].Activate*) in eine Zelle gesetzt werden, um damit sicherzustellen, dass er nicht auf ein Steuerelement gerichtet ist.

Generell bereitet die Verwendung von Steuerelementen in Arbeitsblättern Probleme und löst mitunter nicht nachvollziehbare Fehler und zum Teil sogar Excel-Abstürze aus. Betroffenen von diesen Problemen ist insbesondere die Beispieldatei *07\Userform.xlsm*, aus der bei einer zurückliegenden Neuauflage dieses Buchs einige Beispiele entfernt werden mussten; diese Beispiele funktionierten unter Excel 2000 noch problemlos, verursachten unter späteren Excel-Versionen aber Abstürze.

Inkompatibilitäten gegenüber Excel 2003

Der Wechsel von der letzten „klassischen“ Excel-Version 2003 auf die mit einem Menüband („Ribbon“) ausgestatteten Nachfolgeversionen (2007 und neuer) bringt zwar grundlegende Änderungen bei der Bedienung des Kalkulationsprogramms mit sich, aber nur wenige Unterschiede in Sachen VBA. Und so konnten wir die meisten Beispieldateien ohne Probleme aus der 2003er-Auflage dieses Buchs übernehmen – nach der notwendigen Konvertierung in die neuen Open-XLM-Dateiformate und einigen inhaltlichen Aktualisierungen, versteht sich.

Bei der Beispieldatei *Chart.xlsm* aus Kapitel 10 funktionierte das allerdings nicht so einfach. Die weigerte sich beharrlich, Diagramme für Tagesprotokolle zu erzeugen. Da der Code, der unter Excel 2003 problemlos funktioniert, nicht einmal eine Fehlermeldung produzierte, blieb die Ursache der Verweigerungshaltung längere Zeit im Dunkeln. Dann kam uns die Idee, die Tagesdaten aus der externen Protokollarbeitsmappe zunächst in die Beispieldatei zu kopieren, um sie dann erst zur Grundlage der neuen Diagramme zu machen – und siehe da: es funktionierte! Aus irgendeinem Grund sträuben sich die jüngeren Excel-Versionen, VBA-generierte Diagramme mit externen Daten zu verknüpfen. Ein offensichtlicher Bug, der hoffentlich bald behoben wird.

Auch das Web-Services-Beispiel aus Kapitel 15.4 bereitete uns Probleme. Es entlockte dem Server trotz korrekter Programmierung mitunter keine Ergebnisse und brachte Excel in seltenen Fällen sogar zum Absturz. Microsoft betrachtet das verwendete Web Services Toolkit zwar als ein „not supported product“, sollte aber dennoch bald eine auf aktuelle Excel-Versionen zugeschnittene Version auf den Markt bringen, bei der das Problem hoffentlich nicht mehr auftaucht. Wer Web Services für die Realisierung einer Excel-Lösung benötigt, sollte einstweilen die *Visual Studio Tools for Office* (siehe Abschnitt 15.8) zur Programmierung einsetzen.

TEIL II

Grundlagen

Dieses Kapitel beschreibt detailliert die Bedienung der VBA-Entwicklungsumgebung. Diese Entwicklungsumgebung, auch Visual-Basic-Editor genannt, steht seit Excel 97 zur Verfügung und wurde seither nur mehr in wenigen Details verändert. Sie erscheint in einem getrennten Fenster mit eigenen Menüs, Symbolleisten etc. Die Entwicklungsumgebung ermöglicht die Eingabe von Programmcode und die Definition neuer Formulare, hilft bei der Fehlersuche, enthält eine Objektreferenz (Objektkatalog) mit Querverweisen zur Hilfe, einen Direktbereich zum Test einzelner Anweisungen etc.

■ 3.1 Komponenten von VBA-Programmen

Ein VBA-Programm ist immer Teil einer Excel-Arbeitsmappe – es ist also unmöglich, Excel-Programme außerhalb einer normalen Excel-Datei zu speichern, zu editieren oder auszuführen. Wenn hier von den Komponenten eines VBA-Programms die Rede ist, sind also die VBA-Komponenten einer Excel-Datei gemeint, die in der VBA-Entwicklungsumgebung – auch „VBA-Editor“ genannt – angezeigt werden.



Anmerkung

Es gibt zwei Sonderfälle zur Speicherung von VBA-Code: Zum einen können Sie den Code eines Moduls oder Dialogs als ASCII-Datei exportieren – Sie können diese Dateien aber nicht ausführen. Zum anderen können Sie eine Excel-Datei als Add-in speichern – dann steht der darin enthaltene VBA-Code allen Dokumenten zur Verfügung, und die Tabellenblätter werden unsichtbar. Obwohl ein Add-in rein optisch wenig mit einer Excel-Arbeitsmappe gemeinsam hat und eine ganz andere Aufgabe erfüllen soll (siehe Abschnitt 15.1), handelt es sich nichtsdestoweniger nur um einen Sonderfall einer normalen Excel-Datei.

Eine Excel-Anwendung kann folgende VBA-Komponenten umfassen:

- Normalen Programmcode (Module): Programmcode mit der Definition von Variablen, Prozeduren und Funktionen wird in sogenannten Modulen gespeichert. Ein Modul ist also

eine Gruppe eigens programmierter Prozeduren (Unterprogramme), die in Excel genutzt werden können. In der Entwicklungsumgebung wird ein Modul in einem Textfenster für den Code angezeigt.

- Programmcode zur Definition von neuen Objektklassen (Klassenmodule): Rein optisch sieht ein Klassenmodul wie ein normales Modul aus – es handelt sich also ebenfalls um ein Textfenster mit Programmcode. Der Unterschied besteht darin, dass Klassenmodule zur Definition neuer Objekte dienen. Eine Einführung in die Programmierung von Klassenmodulen gibt Abschnitt 4.5.3.
- Programmcode mit Ereignisprozeduren zu Excel-Objekten: Jedes Excel-Blatt (Tabelle, Diagramm) sowie die gesamte Excel-Arbeitsmappe kennt Ereignisse – etwa den Wechsel von einem Blatt zum anderen, das Speichern oder Drucken der Arbeitsmappe etc. Wenn ein solches vordefiniertes Ereignis eintritt, kann automatisch eine sogenannte Ereignisprozedur ausgelöst werden. Der Programmcode für diese Ereignisprozeduren befindet sich in eigenen Modulen, die jeweils dem entsprechenden Excel-Objekt zugeordnet sind. Detaillierte Informationen zu Ereignisprozeduren gibt Abschnitt 4.4.1.
- Dialoge (*UserForm*): Dialoge bestehen seit Excel 97 aus zwei zusammengehörigen Teilen: dem eigentlichen Dialog mit seinen Steuerelementen und dem Programmcode mit den Ereignisprozeduren zu den Steuerelementen. (Diese Ereignisprozeduren sind zur Verwaltung des Dialogs erforderlich.) Der Entwurf und die Verwaltung von Dialogen sind Thema von Kapitel 7.
- Verweise: Solange Sie nur die Excel-Standardobjekte verwenden, brauchen Sie sich um Verweise nicht zu kümmern. Sobald Sie aber Objekte verwenden möchten, die in externen Objektbibliotheken definiert sind (etwa in der ADO-Bibliothek zur Datenbankprogrammierung), müssen Sie diese mit EXTRAS | VERWEISE aktivieren. Die Verweise auf die genutzten Objektbibliotheken werden in der Excel-Datei gespeichert.

Die ersten vier Punkte dieser Aufzählung haben eine Gemeinsamkeit: Der VBA-Code wird in immer gleich aussehenden Codefenstern angezeigt. Die Werkzeuge zur Codeeingabe und zur Fehlersuche sind also in jedem Fall dieselben.

■ 3.2 Komponenten der Entwicklungsumgebung

Primäre Aufgabe der Entwicklungsumgebung ist es, die Eingabe von Programmcode und die Definition von Formularen zu ermöglichen. Seit Excel 97 ist die VBA-Entwicklungsumgebung nicht mehr in Excel integriert, sondern verhält sich beinahe wie ein eigenständiges Programm. Die Entwicklungsumgebung wird in Excel durch ENTWICKLERTOOLS | VISUAL BASIC aufgerufen und erscheint dann in einem eigenen Fenster.



Hinweis

Falls Sie die standardmäßig ausgeblendete Befehlsregisterkarte ENTWICKLER-TOOLS noch nicht sichtbar gemacht haben, holen Sie das wie folgt nach: Öffnen Sie die Registerkarte DATEI, und wählen Sie OPTIONEN. Klicken Sie links im Dialogfeld auf MENÜBAND ANPASSEN, schalten Sie im rechten Listenfeld das Kontrollkästchen vor ENTWICKLERTOOLS ein, und schließen Sie das Dialogfeld mit OK.

Alternativ können Sie die Entwicklungsumgebung auch mit der Tastenkombination Alt+F11 aktivieren.

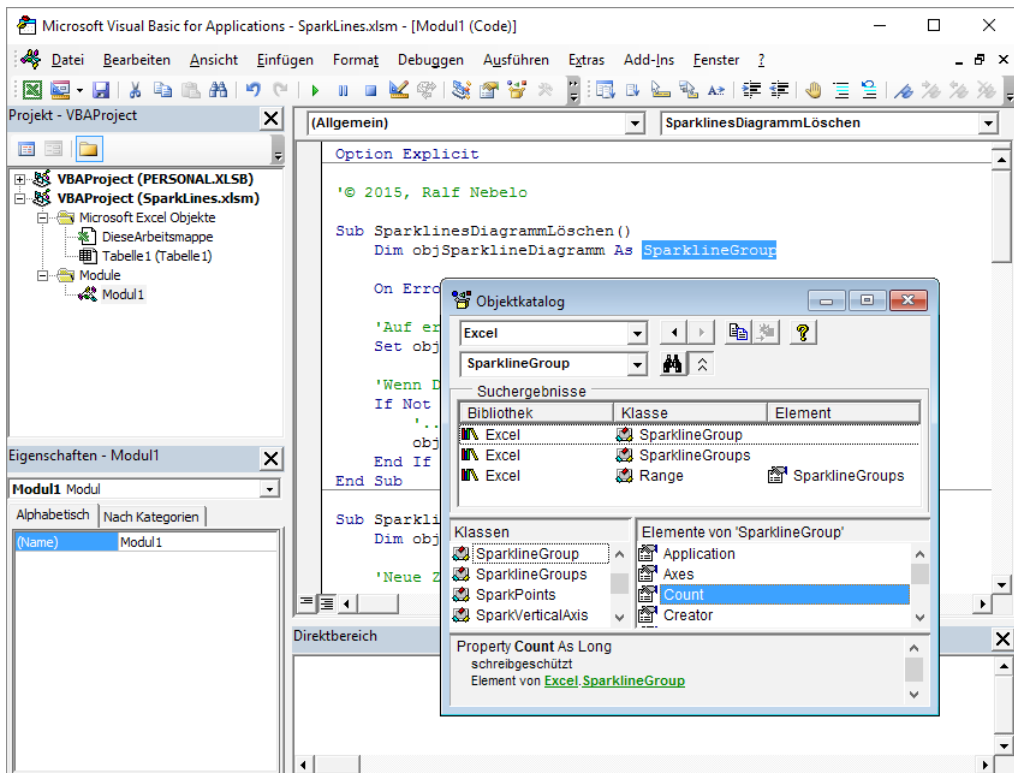


BILD 3.1: Die VBA-Entwicklungsumgebung

Statt mit ENTWICKLERTOOLS | VISUAL BASIC bzw. mit Alt+F11 kann der Wechsel von Excel in die Entwicklungsumgebung auch durch ein Symbol in der *Symbolleiste für den Schnellzugriff* erfolgen: Führen Sie in Excel DATEI | OPTIONEN | SYMBOLLEISTE FÜR DEN SCHNELLZUGRIFF aus, stellen Sie das linke Listenfeld auf „Alle Befehle“ ein, markieren Sie in der Liste darunter den Befehl „Visual Basic“, und klicken Sie auf HINZUFÜGEN. Schließen Sie das Dialogfeld mit OK. Künftig reicht ein einfacher Mausklick aus, um in das Fenster der Entwicklungsumgebung zu springen.

Zu beinahe allen Komponenten der Entwicklungsumgebung sind Kontextmenüs definiert, die mit der rechten Maustaste aufgerufen werden und eine effiziente Ausführung der wichtigsten Kommandos ermöglichen. Probieren Sie es aus!



Hinweis

Der Wechsel zwischen Excel und der Entwicklungsumgebung funktioniert nur, wenn in der gerade aktiven Komponente kein Dialog geöffnet ist. In Excel darf keine Zelle bzw. kein Objekt bearbeitet werden. Ansonsten wird der Blattwechsel ohne Fehlermeldung verweigert.



Verweis

Nicht alle Schritte zur Programmentwicklung werden tatsächlich in der VBA-Entwicklungsumgebung durchgeführt. Beispielsweise wird die Makroaufzeichnung direkt in Excel gesteuert. Aus diesem Grund, und um thematisch zusammengehörige Themen gemeinsam zu behandeln, finden Sie weitere Informationen zur Entwicklungsumgebung in anderen Kapiteln:

- Objektkatalog, Bibliotheksverweise: Abschnitt 4.3,
- Hilfsmittel zur Fehlersuche: Abschnitt 6.1,
- Dialogeditor: Abschnitt 7.3.

Projektfenster

Das Projektfenster (ANSICHT | PROJEKT-EXPLORER oder Strg+R) dient zur Orientierung in Excel-Programmen. Zu jeder geladenen Excel-Datei wird im Projektfenster eine Gruppe mit allen dazugehörigen Modulen und Dialogen angezeigt. Durch einen Doppelklick auf den jeweiligen Eintrag werden die Komponenten in einem Fenster angezeigt und können bearbeitet werden.

Die einzelnen Komponenten eines Projekts können wahlweise alphabetisch geordnet oder wie in Bild 3.1 thematisch gruppiert werden. Die Umschaltung erfolgt durch das dritte Symbol im Projektfenster (mit der irreführenden Bezeichnung `ORDNER WECHSELN`).



Tipp

Wenn Sie mehrere Excel-Dateien gleichzeitig bearbeiten, können Sie einzelne Dateien („Projekte“) im Projektfenster zusammenklappen (Klick auf das Symbol + oder –). Alle Fenster dieses Projekts werden damit ausgeblendet. Das kann die Orientierung in der Entwicklungsumgebung erheblich erleichtern.

**Tipp**

In der Default-Einstellung werden das Projektfenster und die meisten anderen Komponenten nicht als frei verschiebbare Fenster angezeigt, sondern sind am Randbereich der Entwicklungsumgebung fixiert. Das ist nur dann praktisch, wenn Sie mit einem großen Monitor arbeiten. Andernfalls wird der zur Verfügung stehende Platz schlecht genutzt. Um die Komponenten der Entwicklungsumgebung frei platzieren zu können, führen Sie **EXTRAS | OPTIONEN | VERANKERN** aus und deaktivieren sämtliche Optionen dieses Dialogblatts.

Eigenschaftfenster

Im Eigenschaftfenster (**ANSICHT | EIGENSCHAFTENFENSTER** oder **F4**) können diverse Merkmale des gerade aktuellen Objekts eingestellt werden. Als „Objekte“ gelten Module ebenso wie Dialoge und die darin enthaltenen Steuerelemente. Die größte Rolle spielt das Eigenschaftfenster beim Entwurf neuer Dialoge: Jedes Element eines solchen Dialogs kennt Dutzende von Eigenschaften. Bei normalen Codemodulen kann dagegen nur der Name des Moduls eingestellt werden. Dieser Name darf keine Leerzeichen enthalten. Bei Objektmodulen weicht der VBA-Name im Regelfall vom Excel-Blattnamen ab.

Wie im Projektfenster können auch im Eigenschaftfenster die Einträge alphabetisch oder nach Gruppen geordnet werden. Die Umschaltung erfolgt hier allerdings mit Blattregistern und ist nur dann sinnvoll, wenn Objekte sehr viele Eigenschaften unterstützen.

Falls Sie Steuerelemente direkt in Tabellenblätter einbetten, können Sie das Eigenschaftfenster auch in Excel benutzen. Der Aufruf in Excel erfolgt allerdings nicht mit **F4**, sondern über den Kontextmenüeintrag **EIGENSCHAFTEN** bzw. über den **EIGENSCHAFTEN**-Befehl der Symbolleiste **ENTWICKLERTOOLS**.

Objektkatalog

Die Programmierung in Excel basiert auf mehreren Objektbibliotheken, deren wichtigste die *Excel*-Bibliothek ist. Jede Bibliothek ist mit zahlreichen Objekten ausgestattet, die Objekte wiederum mit vielen Eigenschaften, Methoden und Ereignissen. Die einzige Chance, in dieser Fülle von Schlüsselwörtern den Überblick zu bewahren, bietet der Objektkatalog, der mit **ANSICHT | OBJEKTKATALOG** bzw. **F2** angezeigt wird.

**Tipp**

Wenn sich der Cursor gerade über einem Schlüsselwort im Codefenster befindet, wird danach mit **Shift+F2** automatisch im Objektkatalog gesucht.

Der Objektkatalog bietet in vielen Situationen auch den schnellsten Weg zur Hilfe.

Im Objektkatalog werden sowohl die Objekte aller aktivierten Bibliotheken als auch (in fetter Schrift) alle in Modulen selbst definierten Funktionen und Prozeduren angezeigt. Im Listenfeld links oben können Sie die Anzeige auf Objekte einer bestimmten Bibliothek einschränken. Das ist besonders praktisch, wenn die Suche nach einer Zeichenkette sehr viele

Ergebnisse liefert. Eine Suche führen Sie durch, indem Sie im zweiten Listenfeld eine Zeichenkette eingeben und Return drücken.

Normalerweise werden im Katalog nur „offiziell“ unterstützte Schlüsselwörter angezeigt. Daneben gibt es eine Menge verborgener Schlüsselwörter, die entweder intern verwendet werden oder aus Gründen der Kompatibilität zu früheren Versionen aufgenommen wurden. Mit dem Kontextmenükommando **VERBORGENE ELEMENTE ANZEIGEN** können Sie auch diese Schlüsselwörter in grauer Schrift anzeigen.

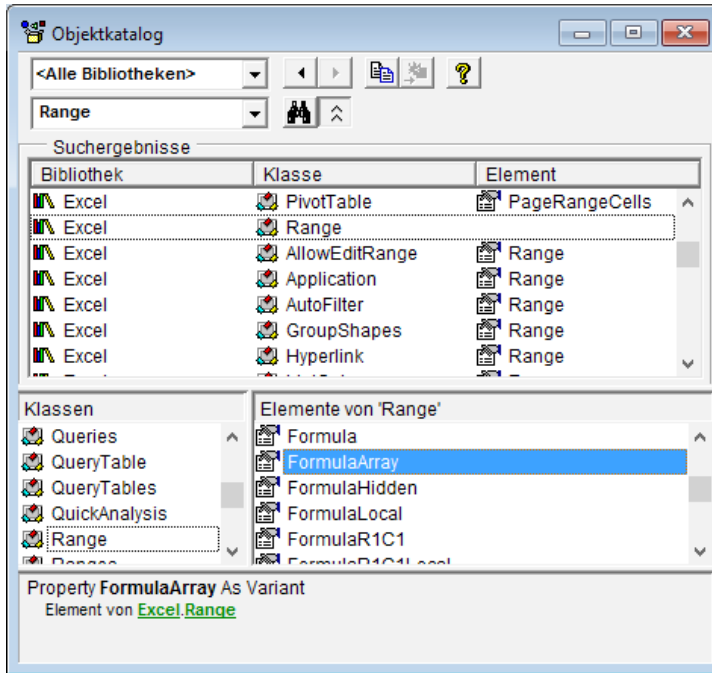


BILD 3.2:
Der Objektkatalog



Tipp

Normalerweise sind alle Schlüsselwörter alphabetisch geordnet. Durch das Kontextmenükommando **ELEMENTE GRUPPIEREN** erreichen Sie, dass die Einträge stattdessen in Gruppen geordnet werden, d. h. zuerst alle Eigenschaften, dann die Methoden und schließlich die Ereignisse. Im Regelfall ist diese Form der Anzeige übersichtlicher.

Editoroptionen

Durch **EXTRAS | OPTIONEN** wird ein vierblättriger Dialog für diverse Einstellungen der Entwicklungsumgebung angezeigt. Die meisten Einstellungen sind leicht verständlich; zu den anderen einige Anmerkungen:

- **AUTOMATISCHE SYNTAXÜBERPRÜFUNG (Blatt EDITOR):** Wenn diese Option aktiviert ist, wird nach der Eingabe einer fehlerhaften Zeile eine Fehlermeldung angezeigt. Während der

ersten VBA-Gehversuche ist das vielleicht ganz nützlich, nach ein paar Tagen werden die ständigen Fehlermeldungen aber lästig. Wenn Sie die Option deaktivieren, werden fehlerhafte Zeilen immer noch in roter Farbe angezeigt, was vollkommen ausreichend ist.

- **VARIABLENDEKLARATION ERFORDERLICH:** Wenn diese Option aktiviert ist, wird in jedes neue Modul die Zeile *Option Explicit* eingefügt. Das bedeutet, dass Sie nur Variablen verwenden können, die Sie mit *Dim* deklariert haben. Diese Option vermeidet Programmfehler aufgrund von Tippfehlern und führt zu einem korrekten Code. Unbedingt aktivieren!
- **ELEMENTE AUTOMATISCH AUFLISTEN, AUTOMATISCHE QUICKINFO, AUTOMATISCHE DATENTIPPS:** Diese drei Optionen geben an, ob im Codefenster automatisch Informationen über die erlaubten Methoden und Eigenschaften, den aktuellen Inhalt von Variablen und über die erlaubten Parameter eingeblendet werden. Lassen Sie die Optionen in der Default-Einstellung (also aktiviert) – die Informationen sind ausgesprochen nützlich.
- **STANDARDMÄSSIG GANZES MODUL ANZEIGEN:** Diese Option bewirkt, dass im Codefenster nicht nur eine einzelne Prozedur, sondern alle Prozeduren des gesamten Moduls angezeigt werden.
- **Blatt EDITORFORMAT:** Hier können Sie den gewünschten Zeichensatz sowie die Farben für verschiedene Syntaxelemente einstellen.

Allgemeine Optionen

- **AUSBLENDEN DES PROJEKTS SCHLIESST FENSTER (Blatt ALLGEMEIN):** Wenn diese Option aktiviert ist, werden alle Fenster eines Projekts geschlossen, sobald das Projekt im Projektfenster zusammengeklappt wird (Klick auf Minussymbol). Beim Aufklappen erscheinen die Fenster wieder. Die Option dient dazu, auch bei mehreren Projekten gleichzeitig eine gewisse Ordnung in der Entwicklungsumgebung zu halten.
- **BEARBEITEN UND FORTFAHREN:** Bei manchen Änderungen im Code – etwa bei der Deklaration neuer Variablen – müssen alle aktuellen Variableninhalte gelöscht werden. Wenn die Option **BENACHRICHTIGEN VOR ZUSTANDSÄNDERUNG** aktiviert ist, werden Sie vor der Durchführung solcher Änderungen gewarnt.
- **BEI JEDEM FEHLER UNTERBRECHEN:** Diese Option hebt die Wirkung von Fehlerbehandlungsroutinen auf. Trotz *On-Error*-Anweisungen wird das Programm unterbrochen. Die Option ist manchmal zur Fehlersuche sehr praktisch (siehe auch Kapitel 6).
- **IN KLASSENMODUL | BEI NICHT VERARBEITETEN FEHLERN UNTERBRECHEN:** Die beiden Optionen führen nur dann zu unterschiedlichen Resultaten, wenn Sie Klassenmodule entwickeln. Wenn in einem Klassenmodul ein Fehler auftritt, wird das Programm im ersten Fall im Klassenmodul und im zweiten Fall dort unterbrochen, wo die Methode oder Eigenschaft der Klasse aufgerufen wurde, die den Fehler verursacht hat (siehe auch Abschnitt 4.5).
- **KOMPILIEREN:** VBA-Programme werden automatisch zu einem Pseudocode kompiliert, der effizienter ausgeführt werden kann als der zugrunde liegende ASCII-Code. (Es handelt sich aber nicht um einen Maschinencode, wie er von C-Compilern erzeugt wird.) Die beiden **KOMPILIEREN**-Optionen steuern, wann kompiliert wird. Die Default-Einstellung (beide Optionen aktiviert) bedeutet, dass sofort mit der Programmausführung begonnen wird und nur jene Prozeduren kompiliert werden, die gerade benötigt werden. Der Vorteil: ein rascher Programmstart. Der Nachteil: Manche offensichtlichen Fehler werden erst spät gemeldet. Bei größeren Projekten ist es zumeist sinnvoll, die Optionen zu deaktivieren,

weil dann mögliche Syntaxfehler im Code sofort gemeldet werden (und nicht irgendwann später, wenn die Prozedur erstmalig benötigt wird).

- **Blatt VERANKERN:** Wie bereits oben in einem Tipp erwähnt, gelten in der Defaultkonfiguration die meisten Komponenten der Entwicklungsumgebung als sogenannte verankerte Fenster. Diese Fenster kleben gewissermaßen an einem Ende der Entwicklungsumgebung. Wenn Sie mit einem ausgesprochen kleinen Monitor arbeiten, können Sie alle Kontrollkästchen dieses Dialogblatts deaktivieren. Anschließend lassen sich alle Fenster ohne Einschränkungen verschieben und überlappend anordnen.

Projekteigenschaften

Mit EXTRAS | EIGENSCHAFTEN VON VBAPROJECT wird ein Dialog zur Einstellung der Eigenschaften des gerade aktuellen Projekts angezeigt. Im Blatt ALLGEMEIN können Sie eine Kurzbeschreibung des Projekts und den Dateinamen einer dazugehörigen Hilfedatei angeben. Im Blatt SCHUTZ können Sie den VBA-Teil einer Excel-Datei ausblenden und durch ein Passwort absichern.



Vorsicht

Welchen Stellenwert Microsoft dem Passwortschutz in Excel gibt, hat man beim Versionswechsel von Excel 7 auf Excel 97 gesehen. In Excel 7 ausgeblendete und per Passwort abgesicherte Module waren in Excel 97 ohne Weiteres jedermann zugänglich! Der Passwortschutz der Excel-Version 2003 war zwar deutlich besser, ließ sich aber ebenfalls knacken beziehungsweise umgehen. Trotz weiterer Verbesserungen gilt das leider auch noch für Excel 2016. Es gibt diverse kommerzielle Tools, die den vermeintlichen Schutz in wenigen Sekunden aushebeln.

Bedingte Kompilierung

Manchmal kommt es vor, dass Sie parallel zu einem Programm eine zweite Version verwalten möchten (etwa eine Demoversion mit eingeschränkten Merkmalen oder eine Debug-Version mit zusätzlichen Sicherheitsabfragen). Dazu können Sie in EXTRAS | EIGENSCHAFTEN VON VBAPROJECT | ALLGEMEIN im Textfeld ARGUMENTE FÜR BEDINGTE KOMPILIERUNG eine Konstante definieren, beispielsweise *demo=1*. Im Programmcode können Sie den Inhalt der Konstanten dann mit *#If*-Anweisungen auswerten.

Je nach Ergebnis der *#If*-Abfrage wird entweder der eine oder der andere Zweig ausgeführt. Im Unterschied zu normalen *If*-Abfragen erfolgt die Unterscheidung zwischen den beiden Varianten allerdings schon bei der Kompilierung. Das Kompilat enthält nur eine Variante und keine *#If*-Abfragen, es ergibt sich also kein Geschwindigkeitsnachteil. Die folgenden Zeilen zeigen, wie Programmcode mit *#If*-Anweisungen aussehen kann:

```
Sub Command1_Click()
    #If demo Then
        MsgBox "In der Demoversion kann nichts gespeichert werden"
    #Else
        ' ... Programmcode zum Speichern
    #End If
End Sub
```