

Excel 2016 programmieren

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

Bearbeitet von
Ralf Nebelo, Michael Kofler

1. Auflage 2016. Buch. 880 S.
ISBN 978 3 446 44798 1
Format (B x L): 18,5 x 24,5 cm
Gewicht: 1696 g

[Weitere Fachgebiete > EDV, Informatik > Programmiersprachen: Methoden > Microsoft Programmierung](#)

schnell und portofrei erhältlich bei


DIE FACHBUCHHANDLUNG

Die Online-Fachbuchhandlung beck-shop.de ist spezialisiert auf Fachbücher, insbesondere Recht, Steuern und Wirtschaft. Im Sortiment finden Sie alle Medien (Bücher, Zeitschriften, CDs, eBooks, etc.) aller Verlage. Ergänzt wird das Programm durch Services wie Neuerscheinungsdienst oder Zusammenstellungen von Büchern zu Sonderpreisen. Der Shop führt mehr als 8 Millionen Produkte.



Leseprobe

Michael Kofler, Ralf Nebelo

Excel 2016 programmieren

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

ISBN (Buch): 978-3-446-44798-1

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

Weitere Informationen oder Bestellungen unter

<http://www.hanser-fachbuch.de/978-3-446-44798-1>

sowie im Buchhandel.

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 Beispielprogramms	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 Programmiererweiterungen 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 entwicklungs-technisch 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 Web-Techniken 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 Mausclicks lösen, für das Sie ansonsten womöglich ein Programm geschrieben hätten ... ;-)

Jenseits von VBA

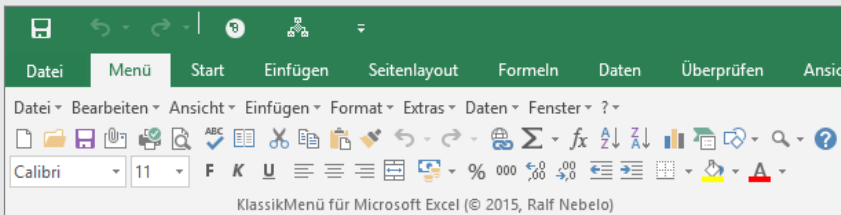
Obwohl VBA immer noch das wichtigste Werkzeug für die Entwicklung von Excel-Lösungen ist (und daher im Mittelpunkt dieses Buchs steht), stellen sich zunehmend mehr Aufgaben, die mit der „eingebauten“ Programmiersprache des Kalkulationsprogramms nur noch teilweise oder gar nicht mehr zu lösen sind. Beispiele 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.kofier.info	Autorenseite von Michael Kofier
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/assstvid.aspx	Interaktive Befehlsreferenz Excel 2007

Die Internetadressen in der Datei *Hyperlinks.pdf* können Sie direkt per Mausclick 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 Doppelgleichheiten der Form *der/die Anwender/in* oder kurz *AnwenderIn* aber sprachlich als nicht schön – und zwar sowohl beim Schreiben als auch beim Lesen.

VBA-Elemente für die Programmierung von SmartArt-Diagrammen	
<i>Description</i>	Beschreibung eines SmartArt-Layouts
<i>ID</i>	Kennung eines SmartArt-Layouts
<i>Larger</i>	vergrößert den Diagrammknoten
<i>msoSmartArt</i>	Konstante für die Kennzeichnung eines SmartArt-Diagramms
<i>Name</i>	Name eines SmartArt-Layouts
<i>Nodes</i>	Auflistung aller Knoten in einem SmartArt-Diagramm oder übergeordneten Knoten
<i>Promote</i>	stuft den Diagrammknoten herauf
<i>ReorderDown</i>	verschiebt den Diagrammknoten nach unten
<i>ReorderUp</i>	verschiebt den Diagrammknoten nach oben
<i>SmartArt</i>	verweist auf das SmartArt-Diagramm in einem Shape-Objekt
<i>SmartArtLayout</i>	verweist auf ein einzelnes SmartArt-Layout
<i>SmartArtLayouts</i>	Auflistung aller Layouts für SmartArt-Diagramme
<i>SmartArtNode</i>	verweist auf einen einzelnen Knoten innerhalb der Nodes-Auflistung
<i>TextFrame2.TextRange.Text</i>	enthält die Beschriftung eines Diagrammknotens

■ 10.8 Neue Diagrammtypen in Excel 2016

In Excel 2016 stehen dem Anwender nun die sechs neuen Diagrammtypen *Wasserfall*, *Histogramm*, *Pareto*, *Kastengrafik*, *Treemap* und *Sunburst* zur Verfügung. 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.

10.8.1 Programmierung von Wasserfall-Diagrammen

Ein Wasserfall-Diagramm verschafft dem Anwender ein klares Bild darüber, wie sich einzelne Faktoren auf ihre Nachfolger in einer Zahlenreihe auswirken. So kann man mit seiner Hilfe beispielsweise sehr gut darstellen, wie viel vom Umsatz eines Unternehmens nach Abzug diverser Kosten noch als Gewinn übrigbleibt. Bei einem solchen Einsatzszenario ergibt sich eine Reihe von treppenartig abfallenden Balken, die – mit etwas Fantasie – das Bild eines Wasserfalls ergeben.

Eine idealtypische Zahlentabelle für den Einsatz eines Wasserfall-Diagramms könnte wie folgt aussehen:

Bilanz	
Umsatz	61.310,00 €
Lohnkosten	- 15.056,00 €
Materialkosten	- 8.567,00 €
Transportkosten	- 5.824,45 €
Brutto	31.862,55 €

Sie finden diese Zahlentabelle in der Beispieldatei *10\Diagrammtypen2016.xlsm* auf einem Arbeitsblatt, das Sie mit einem Klick auf den Menü-Button „Wasserfall“ aktivieren können. Die Zahlentabelle ist mit dem Namen „rngDataWasserfall“ hinterlegt.

Das programmierte Anlegen eines Wasserfall-Diagramms beginnt nun damit, dass der benannte Arbeitsblattbereich mit zwei Befehlszeilen selektiert wird:

```
Set rngData = Range("rngDataWasserfall")
rngData.Select
```

Anschließend genügt die folgende Anweisung, um ein neues (eingebettetes) Diagramm vom Typ Wasserfall (*xlWaterfall*) zu generieren und ihm den selektierten Arbeitsblattbereich als Datenquelle zuzuweisen:

```
Set shpDiagram = ActiveSheet.Shapes.AddChart2(-1, xlWaterfall)
```



Hinweis

Die Programmzeile generiert das Objekt *shpDiagram*, das wie jedes eingebettete (d. h. in ein Arbeitsblatt eingefügte) Diagramm vom Typ *Shape* ist. Das eigentliche Diagramm darin ist über die *Chart*-Eigenschaft des Shape-Objekts zugänglich. Hinweis im Hinweis: Alle Shape-Objekte eines Arbeitsblatts sind über dessen *Shapes*-Auflistung abrufbar.

Das neu angelegte Diagramm hat jedoch noch wenig Ähnlichkeit mit einem Wasserfall, da sein letzter Balken – er stellt den errechneten „Brutto“-Wert am Ende der Zahlentabelle dar – auf der gleichen Höhe wie der Balken „Transportkosten“ beginnt und damit „nach oben“ statt „nach unten“ geht.

Um dieses Problem zu lösen, muss man Excel mitteilen, dass es sich bei diesem letzten Wert nicht um einen Einzelwert handelt, der von seinem Vorläufer abgezogen oder zu diesem hinzuaddiert wird, sondern um eine Summe. Händisch würde man das realisieren, indem man auf den letzten Diagrammbalken doppelklickt und dann das Kontrollkästchen „Als Summe festlegen“ am rechten Bildschirmrand einschaltet.

Die programmierte Variante der Lösung ist kaum schwieriger. Sie beginnt mit einem Verweis auf den letzten Datenpunkt des Diagramms, der sich wie folgt herstellen lässt:

```
Set objLastPoint =
shpDiagram.Chart.SeriesCollection(1).Points(
shpDiagram.Chart.SeriesCollection(1).Points.Count)
```

Danach muss man nur noch die *IsTotal*-Eigenschaft des *objLastPoint*-Objekts (das vom Typ *Point* ist) auf *True* setzen:

```
objLastPoint.IsTotal = True
```

Das Ergebnis dieser Aktion sollte dann so aussehen wie die nachfolgende Abbildung.

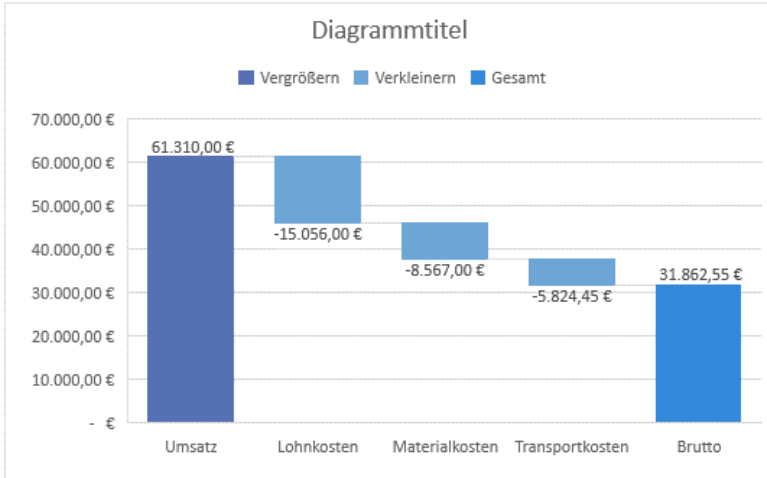


BILD 10.20: Das programmierte Wasserfall-Diagramm zeigt anschaulich, wie viel vom Umsatz nach Abzug diverser Kosten noch als (Brutto-)Gewinn übrigbleibt.

10.8.2 Programmierung von Histogrammen

Ein Histogramm ist eine grafische Darstellung der Häufigkeitsverteilung bestimmter Merkmale. Es erfordert die Einteilung der Daten in Klassen, die eine konstante oder variable Breite haben können. In Excel bestehen Histogramme aus nebeneinanderliegenden Balken, die jeweils einer Klasse entsprechen. Die Höhe eines jeden Balkens stellt die sogenannte Häufigkeitsdichte der betreffenden Klasse dar.

Was ziemlich kompliziert und wissenschaftlich klingt, lässt sich im Excel-Alltag aber sehr einfach und gewinnbringend anwenden. So könnte ein Lehrer beispielsweise mit Hilfe eines Histogramms sehr leicht feststellen, wie sich die Leistungen seiner Schüler verteilen. Als Datenbasis einer solchen Analyse könnte eine Zahlentabelle dienen, die in Spalte A die Namen der Schüler und dahinter in Spalte B deren Noten (oder Zensuren) enthält:

Schüler	Note
Schüler 1	3
Schüler 2	3
Schüler 3	6
Schüler 4	6
Schüler 5	5
Schüler 6	6

Die Abbildung oben zeigt nur einen Ausschnitt einer solchen Zahlentabelle; das vollständige Exemplar finden Sie in der Beispieldatei *10\Diagrammtypen2016.xlsm* auf einem Arbeitsblatt, das Sie mit einem Klick auf den Menü-Button „Histogramm“ aktivieren können.

Der Inhalt der Tabelle ist mit dem Namen „rngDataHistogramm“ hinterlegt, der im ersten Schritt der Programmierung wie folgt selektiert wird:

```
Set rngData = Range("rngDataHistogramm")
rngData.Select
```

Die Selektion stellt nun automatisch den Datenbereich des Histogramms dar, das mit der folgenden Programmzeile generiert wird:

```
Set shpDiagram = ActiveSheet.Shapes.AddChart2(-1, xlHistogram)
```

Das resultierende Diagramm (siehe nachfolgende Abbildung) stellt die Verteilung der Schulnoten in drei Klassen dar, die man mit „Gut“, „Mittel“ und „Schlecht“ überschreiben könnte. Der Lehrer sieht daran sofort, dass die Mehrzahl seiner Schüler der Klasse „Schlecht“ angehören und deren Versetzung somit gefährdet ist.

Sollte sich der Lehrer eine feinere Klassenbildung wünschen – beispielsweise für jede Schulnote einen eigenen Balken – so kann er die Eigenschaften des Histogramms manuell ändern. Dazu doppelklickt er auf die untere Diagrammachse und ändert dann im Dialogfeld *Achsenoptionen* am rechten Bildschirmrand die Einstellung *Anzahl der Container* (das sind die Klassen des Histogramms) auf „6“ und die Einstellung *Containerbreite* auf „0,999“. Ein programmierter Zugriff auf diese Einstellungen ist leider nicht möglich, da diese offenbar keinen Eingang in das Objektmodell von Excel 2016 gefunden haben.

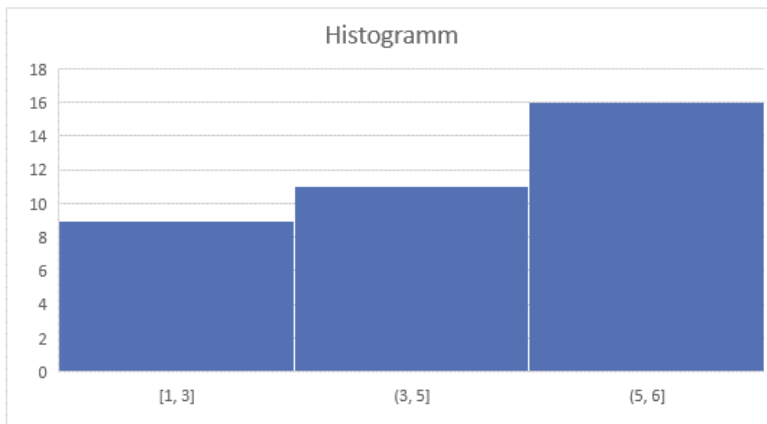


BILD 10.21: Ein Histogramm fasst Merkmale (hier Schulnoten) zu Klassen zusammen und stellt dann deren Häufigkeiten als Balken dar.

10.8.3 Programmierung von Pareto-Diagrammen

Ein Pareto-Diagramm ist die Sonderform eines Säulendiagramms, bei dem die einzelnen Werte der Größe nach geordnet angezeigt werden. Dabei befindet sich der größte Wert ganz links, der kleinste ganz rechts. Neben dem Säulendiagramm enthält ein Pareto-Diagramm aber auch noch ein Liniendiagramm, das die prozentualen Anteile der Einzelwerte am Gesamtergebnis in akkumulierter Form anzeigt.

Auch das klingt komplizierter als es ist. In der Praxis benutzt man ein Pareto-Diagramm immer dann, wenn man die wichtigsten Größen für das Zustandekommen eines Ergebnisses erkennen und sie von den weniger wichtigen Größen unterscheiden möchte. Der Chef eines Versicherungsunternehmens könnte ein Pareto-Diagramm beispielsweise dazu verwenden, die Top-Drei seiner Außendienstmitarbeiter zu ermitteln. Dazu schreibt er die Namen seiner Mitarbeiter in die erste Spalte einer Zahlentabelle, die Umsätze in die zweite. Das Ergebnis könnte dann wie folgt aussehen:

Vertreter	Umsatz
Vertreter 1	11.445,00 €
Vertreter 2	11.829,00 €
Vertreter 3	14.547,00 €
Vertreter 4	14.702,00 €
Vertreter 5	10.203,00 €
Vertreter 6	10.669,00 €
Vertreter 7	10.107,00 €
Vertreter 8	31.467,00 €
Vertreter 9	35.150,00 €
Vertreter 10	30.064,00 €

Sie finden diese Tabelle, wenn Sie die Beispieldatei *10\Diagrammtypen2016.xlsm* öffnen und auf den Menü-Button „Pareto“ klicken.

Das programmierte Anlegen des Pareto-Diagramms beginnt nun mit zwei Programmzeilen, die den Inhalt der Zahlentabelle selektieren. Dazu wurde dieser zuvor mit dem Namen „rngDataPareto“ gekennzeichnet:

```
Set rngData = Range("rngDataPareto")
rngData.Select
```

Die Zeilen machen die selektierten Zellen zum Datenbereich des neuen Diagramms, das dann mit dem folgenden Einzeiler generiert wird:

```
Set shpDiagram = ActiveSheet.Shapes.AddChart2(-1, xlPareto)
```

Über die *Chart*-Eigenschaft des Objekts *shpDiagram* können Sie anschließend auf alle Standardeigenschaften des Diagramms zugreifen, beispielsweise um den Titel des Diagramms nach Ihren Wünschen zu ändern:

```
shpDiagram.Chart.ChartTitle.Text = "Mein Pareto-Diagramm"
```

Spezielle Eigenschaften eines Pareto-Diagramms haben leider keinen Einzug in Excels Objektmodell gefunden.

Das Resultat der *AddChart2*-Anweisung oben sieht dann so aus wie in der nachfolgenden Abbildung. Das Diagramm verrät dem Chef mit einem Blick, dass seine drei besten Mitarbeiter – sie haben die (Personal-)Nummern 9, 8 und 10 – mehr als die Hälfte des Umsatzes generieren. Zeit für eine Lohnerhöhung!

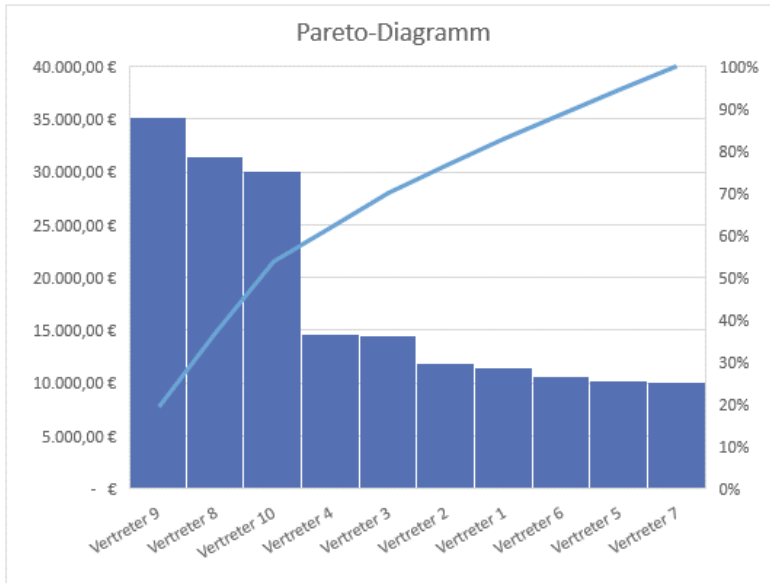


BILD 10.22: Wer wissen möchte, welche Faktoren den größten Anteil an einem bestimmten Ergebnis haben, verwendet ein Pareto-Diagramm. Hier sieht man, dass drei Mitarbeiter mehr als die Hälfte des Umsatzes generieren.

10.8.4 Programmierung von Kastengrafik-Diagrammen

Bei einem Kastengrafik-Diagramm, das im Englischen auch Boxplot oder Box-Whisker-Plot genannt wird, steht ähnlich wie bei einem Histogramm (siehe Abschnitt 10.8.2) die Verteilung von Daten im Mittelpunkt. Allerdings geht das Kastengrafik-Diagramm weit über die Anzeige schlichter Häufigkeiten hinaus, indem es für jede Datenreihe eine komplexe Grafik generiert, die verschiedene Streuungs- und Lagemaße in einer Darstellung kombiniert. Zentrales (und namensgebendes) Element einer Kastengrafik ist ein Rechteck. Es umfasst den Zahlenbereich, in dem die mittleren 50 Prozent der Daten liegen. Die beiden Linien ober- und unterhalb des Rechtecks werden Antennen genannt. Sie repräsentieren die Spannweite der oberen und unteren 25 Prozent der Daten und markieren an ihrem Ende den größten und kleinsten Wert der Datenreihe. Innerhalb des Rechtecks zeigt Excel den Median – das ist der mittlere Wert in der sortierten Datenreihe – mit einer Querlinie an; ein Kreuz markiert das arithmetische Mittel.

Mit Hilfe eines Kastengrafik-Diagramms könnte sich ein Buchhändler beispielsweise einen detaillierten Einblick verschaffen, wie sich die Umsätze verschiedener Buchgenres gestalten. Dazu legt er zunächst eine Zahlentabelle an, die in Spalte 1 den Titel, in Spalte 2 das Genre und in der letzten Spalte schließlich den Preis des jeweiligen Buchs verzeichnet. Die

nachfolgende Abbildung zeigt nur einen Ausschnitt; die vollständige Tabelle finden Sie, wenn Sie die Beispieldatei `10\Diagrammtypen2016.xlsm` öffnen und auf den Menü-Button „Kastengrafik“ klicken.

Buch	Genre	Preis
Buch 1	Fantasy	24,00 €
Buch 2	Fantasy	16,00 €
Buch 3	Satire	24,00 €
Buch 4	Satire	28,00 €
Buch 5	Fantasy	13,00 €
Buch 6	Satire	21,00 €
Buch 7	Belletristik	42,00 €

Die letzten beiden Spalten der Tabelle bilden den Datenbereich des anzulegenden Diagramms, wozu sie mit dem Namen „rngDataKasten“ hinterlegt wurden. Dieser benannte Bereich wird nun im ersten Schritt der Programmierung selektiert:

```
Set rngData = Range("rngDataKasten")
rngData.Select
```

Die folgende Anweisung erstellt nun auf der Grundlage der Selektion ein neues Diagramm vom Typ Kastengrafik:

```
Set shpDiagram = ActiveSheet.Shapes.AddChart2(-1, xlBoxwhisker)
```

Das Ergebnis dieser Aktion sollte der nachfolgenden Abbildung entsprechen. Über die *Chart*-Eigenschaft des Objekts *shpDiagram* kann der Entwickler wieder auf alle Standardeigenschaften des Diagramms zugreifen. Wer die spezifischen Eigenschaften eines Kastengrafik-Diagramms ändern möchte, muss das manuell über die Formatierungsdialoge von Excel erledigen. In Excels Objektmodell finden sich keine entsprechenden Erweiterungen.

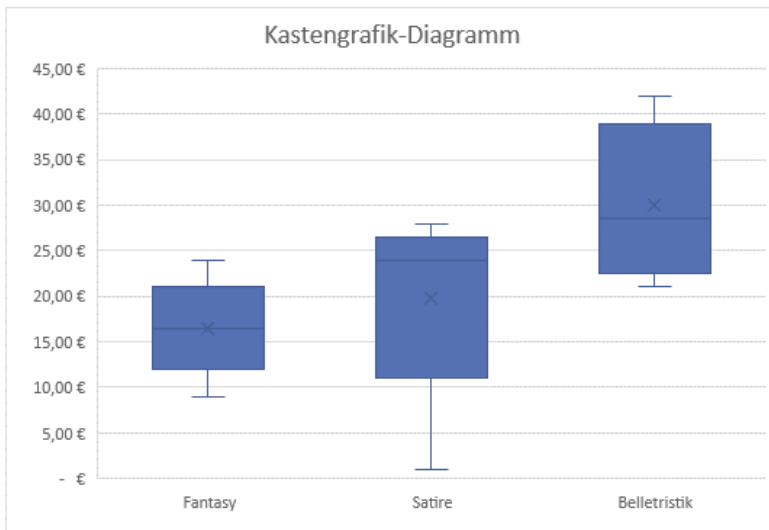


BILD 10.23: Kastengrafik-Diagramme enthalten diverse statistische Informationen darüber, wie sich die Daten einer Reihe – hier geht es um die Umsatzzahlen innerhalb von Buchgenres – verteilen.

10.8.5 Programmierung von Treemap-Diagrammen

Ein Treemap-Diagramm (auch Baumkarte genannt) visualisiert die Größenverhältnisse innerhalb von hierarchisch strukturierten Daten. Dazu stellt es Rechtecke innerhalb von Rechtecken dar. Die Flächen dieser Rechtecke sind dabei stets proportional zur Größe der darzustellenden Dateneinheit.

Ein typisches Einsatzszenario für ein Treemap-Diagramm könnte so aussehen: Ein Restaurantbesitzer möchte einen vollständigen Überblick über die Umsatzverteilung seines Unternehmens. Dazu möchte er nicht nur wissen, wie sich die Umsätze über die drei Mahlzeitenangebote Frühstück, Mittag- und Abendessen verteilen, sondern auch innerhalb der einzelnen Mahlzeitenangebote über die angebotenen Speisen und Getränke. Zu diesem Zweck erstellt er die folgende Zahlentabelle, die Sie in der Beispieldatei *10\Diagrammtypen2016.xlsm* nach einem Klick auf den „Treemap“-Button finden:

Mahlzeit	Speiseart	Name	Umsatz
Frühstück	Getränk	Milch	6,00 €
		Kaffee	9,00 €
		Orangensaft	7,00 €
	Speise	Brötchen	32,00 €
		Marmelade	11,00 €
Mittagessen	Getränk	Milchshake	15,00 €
		Smoothy	17,00 €
		Speise	Schnitzel
Abendessen	Getränk	Kartoffeln	35,00 €
		Tee	9,00 €
		Wein	44,00 €
		Speise	Steak
		Fisch	43,00 €
		Auflauf	33,00 €

Diese Zahlentabelle ist mit dem Namen „rngDataTreemap“ hinterlegt. Die folgenden Anweisungen

```
Set rngData = Range("rngDataTreemap")
rngData.Select
```

selektieren den benannten Bereich und machen ihn automatisch zur Datenbasis des neuen Treemap-Diagramms, das nach Ausführung der Code-Zeile

```
Set shpDiagram = ActiveSheet.Shapes.AddChart2(-1, xlTreemap)
```

entsteht. Das neu angelegte Diagramm entspricht jedoch der Standard-Diagrammvorlage, die weniger übersichtlich gestaltet ist als eine andere Vorlage, auf die man mit folgender Anweisung umschalten kann:

```
shpDiagram.Chart.ChartStyle = 413
```

Jetzt erscheinen die Namen der Mahlzeitenangebote als graue Balken direkt oberhalb der Rechtecke, die die Umsätze der zugehörigen Speisen und Getränke visualisieren. Da die Diagrammlegende jetzt überflüssig ist, schalten wir sie mit dieser Code-Zeile ab:

```
shpDiagram.Chart.HasLegend = False
```

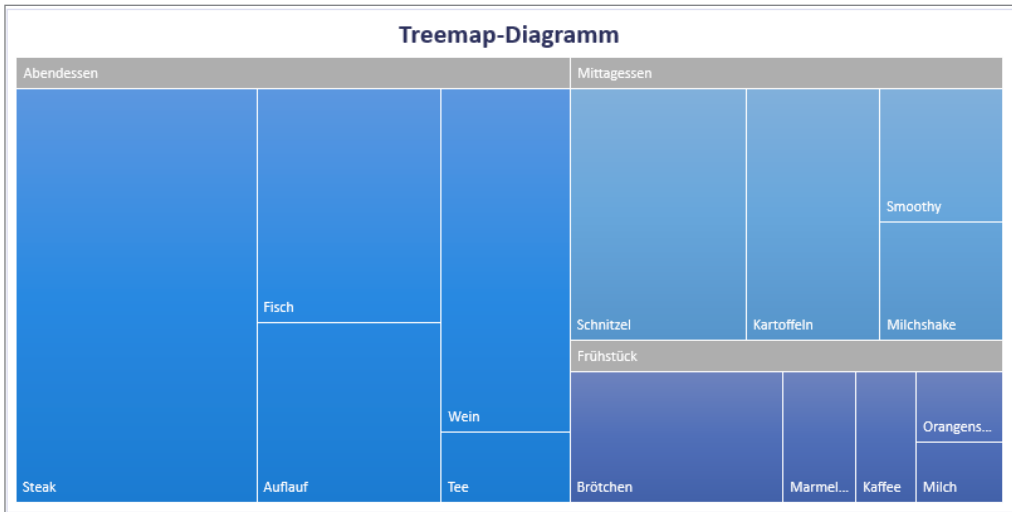


BILD 10.24: Dieses Treemap-Diagramm verrät dem Restaurantbesitzer, wie sich die Umsätze seines Unternehmens über die verschiedenen Mahlzeiten und die jeweils dabei angebotenen Speisen und Getränke verteilen.

10.8.6 DirectoryMap – Inhaltsverzeichnisse visualisieren

Irgendwann kommt jede Festplatte an ihre Kapazitätsgrenzen. Wer dann effektiv Platz für neue Inhalte schaffen will, sollte beim Aufräumen mit den größten Platzverschwendern beginnen. Die aufzuspüren, ist jedoch nicht immer leicht, da der Windows-Explorer nur über den Umweg des EIGENSCHAFTEN-Dialogs Auskunft über Verzeichnisgrößen liefert. Wenige Zeilen VBA und der neue Diagrammtyp Treemap (siehe Abschnitt 10.8.5) genügen allerdings, um das Manko zu beheben.

Inside DirectoryMap.xlsm

Die Beispieldatei *10\DirectoryMap.xlsm* enthält nur ein einziges Arbeitsblatt namens „Directory“. An dessen Spitze befindet sich ein Shape mit Button-Optik, das beim Anklicken das Makro *ShowDirectoryMap* aufruft. Das bringt zunächst das typische Dialogfeld zur Auswahl eines Ordners auf den Bildschirm. Dazu bedient sich das Makro im Rahmen der ActiveX-Automation (siehe Abschnitt 15.6) des Windows-eigenen *Shell*-Objekts, das es mit der Zeile

```
Set objShell = CreateObject("Shell.Application")
```

einbindet. Die Darstellung des Dialogs erledigt dann die *BrowseForFolder*-Methode des Shell-Objekts, das den ausgewählten Ordner als *Folder*-Objekt zurückgibt:

```
Set objFolder = objShell.BrowseForFolder(0, strPrompt, &H8, 0)
```

Das aufrufende Makro liest den Pfad des gewählten Ordners aus *objFolder.Items.Item.Path* und speichert ihn in der Variablen *strPath*.

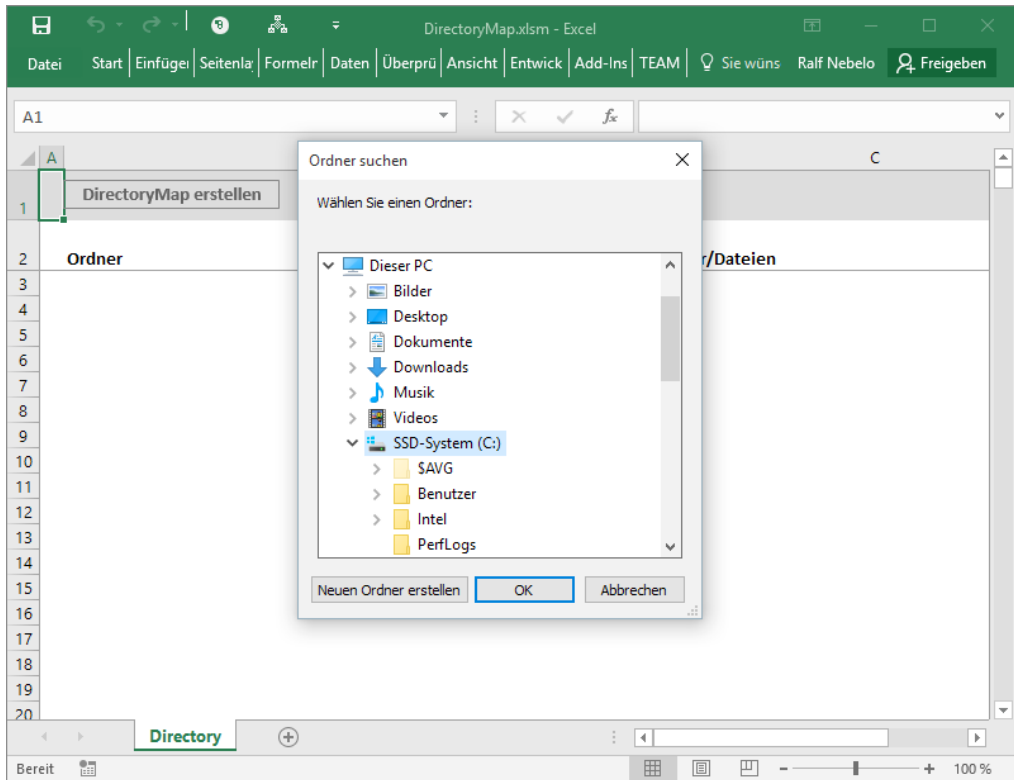


BILD 10.25: Ein Klick auf den (Shape-)Button „DirectoryMap erstellen“ bringt zunächst ein Dialogfeld hervor, das die Auswahl des gewünschten Ordners oder Laufwerks erlaubt.

Da das Makro mit sehr ausführlichen Kommentaren versehen ist, können wir uns im Folgenden auf die wichtigsten Aspekte der Programmierung beschränken. Auf die *CreateDirectory*-Funktion beispielsweise, die das Inhaltsverzeichnis des ausgewählten Ordners generiert und es als Datenbasis für das spätere Treemap-Diagramm in das Arbeitsblatt „Directory“ schreibt.

Inhaltsverzeichnis generieren

Da VBA nur über unzureichende Bordmittel für den Zugriff auf das Dateisystem verfügt, greift es diesbezüglich – und wiederum im Rahmen der ActiveX-Automation – auf die Dienste der *File System Objects* (siehe Abschnitt 5.6.1) zurück, die es wie folgt einbindet:

```
Set objFSO = CreateObject("Scripting.FileSystemObject")
```

Mit Hilfe der *GetFolder*-Methode der File System Objects verschafft sich das Makro einen Verweis auf den ausgewählten Ordner, den es in der Variablen *objDir* entgegennimmt:

```
Set objDir = objFS0.GetFolder(strPath)
```

Anschließend durchläuft es per *For-Each-Next*-Schleife alle Unterordner des Ordners und schreibt deren Namen in die erste Spalte der Inhaltsverzeichnis-Tabelle:

```
For Each objSubDir In objDir.SubFolders
    wksSheet.Cells(lngRow, lngFirstColumn).Value = objSubDir.Name
    ...
Next
```

Innerhalb dieser Schleife gibt es zwei weitere (eingeschlossene) Schleifen, die die Inhalte des jeweiligen Unterordners durchlaufen und die Namen und Größen (in Bytes) der dabei gefundenen (Unter-Unter-)Ordner und Dateien in die zweite und dritte Spalte der Inhaltsverzeichnis-Tabelle ausgeben:

```
For Each objSubDir2 In objSubDir.SubFolders
    wksSheet.Cells(lngRow, lngFirstColumn + 1).Value = _
        objSubDir2.Name
    wksSheet.Cells(lngRow, lngFirstColumn + 2).Value = _
        objSubDir2.Size
    lngRow = lngRow + 1
Next

For Each objFile In objSubDir.Files
    wksSheet.Cells(lngRow, lngFirstColumn + 1).Value = _
        objFile.Name
    wksSheet.Cells(lngRow, lngFirstColumn + 2).Value = _
        objFile.Size
    lngRow = lngRow + 1
Next
```

Zum Abschluss ihrer Tätigkeit merkt sich die *CreateDirectory*-Funktion den Bereich (Range) der soeben erstellten Inhaltsverzeichnis-Tabelle ...

```
Set rngData = Range(wksSheet.Cells(lngFirstRow, _
    lngFirstColumn), wksSheet.Cells(lngRow - 1, lngFirstColumn + 2))
```

... und gibt ihn mit der folgenden Zeile an das aufrufende Makro zurück:

```
Set CreateDirectory = rngData
```

Ordner	Unterordner/Dateien	Bytes
Program Files (x86)	Acronis	167.254.577,00
	Adobe	184.751.387,00
	AppInsights	1.503.778,00
	Apple Software Update	2.428.606,00
	Application Verifier	311.234,00
	ASUS	72.908.783,00
	Audible	507.990,00
	AVG	157.130.562,00
	BriefBlitz	4.223.722,00
	BriefBlitz.com	42.087.148,00
	Canon	72.220.989,00
	CD-LabelPrint	11.649.311,00
	Common Files	1.212.259.400,00
	Component Factory	118.617.825,00
	Corel	781.651.546,00
	CustomUIEditor	418.688,00
	D-Link	1.497.392,00
	DVDFab 8 Qt	55.754.445,00
	HTML Help Workshop	217.744,00
	IcoFX 1.6	3.839.735,00
	IIS	1.163.091,00
	IIS Express	17.563.722,00
	InstallShield	144.109.170,00
	InstallShield Installation Information	15.170.809,00
	Intel	17.790.859,00
	Internet Explorer	2.404.083,00
	Java	144.380.940,00
	LogMeIn Hamachi	11.084.171,00
	MediathekView	66.995.695,00
	Microsoft	4.202.368,00

BILD 10.26: Das Makro generiert eine Verzeichnis-Tabelle, die alle Ordner des gewählten Verzeichnisses und deren jeweilige Unterordner und Dateien auflistet.

Verzeichnisinhalte per Treemap darstellen

Nachdem dem Makro nun der Bereich des generierten Inhaltsverzeichnis bekannt ist, muss es diesen nur noch zur Datenbasis eines neuen Treemap-Diagramms machen. Das Anlegen des Diagramms beginnt mit der Auswahl der Inhaltsverzeichnis-Tabelle:

```
rngData.Select
```

Die folgende Zeile erstellt dann ein neues Treemap-Diagramm und weist diesem die Diagrammvorlage mit der internen Nummer 413 zu, welche die Ordnernamen zwecks besserer Übersicht als graue Balken über den jeweiligen Ordnerinhalten erscheinen lässt:

```
Set shpDiagram = wksDirectory.Shapes.AddChart2(413, xlTreemap)
```

Jetzt müssen in dem neuen Diagramm nur noch die überflüssige Legende abgeschaltet und die Überschrift so geändert werden, dass diese den Pfad des ausgewählten Ordners zeigt:

```
With shpDiagram.Chart
    .HasLegend = False
    .ChartTitle.Text = "Inhalt von " & strPath
End With
```

Die finale Anweisung ...

```
shpDiagram.Chart.Location xlLocationAsNewSheet
```

... verschiebt das Diagramm(-Shape) schließlich aus dem Arbeitsblatt in ein eigenständiges Diagrammblatt, wo es mehr Platz für eine detaillierte Darstellung der Verzeichnisinhalte bekommt.

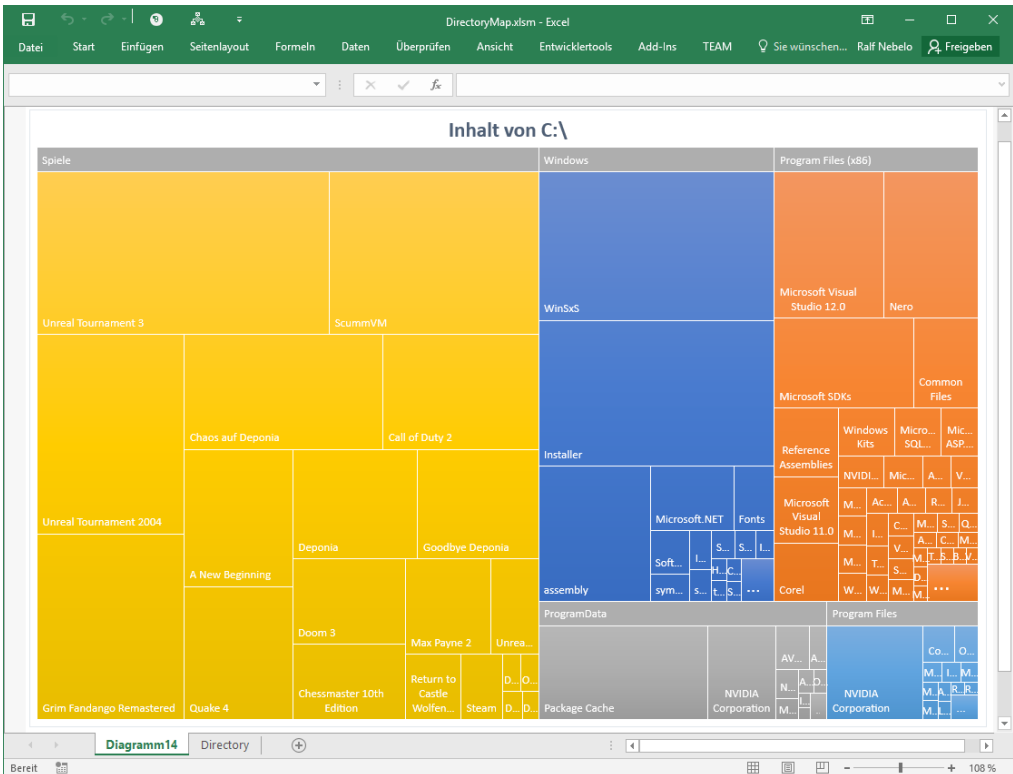


BILD 10.27: Das Treemap-Diagramm stellt die Ordnergrößen auf der Festplatte (und die offensichtliche Vorliebe des Autors für Computerspiele ;-) eindrucklich dar.

10.8.7 Programmierung von Sunburst-Diagrammen

Ähnlich dem Treemap-Diagramm (siehe Abschnitt 10.8.5) stellt auch das Sunburst-Diagramm die Größenverhältnisse innerhalb von hierarchisch strukturierten Daten dar. Allerdings greift es dazu nicht auf verschachtelte Rechtecke zurück, sondern auf ebenso verschachtelte Ringe oder Kreise, welche die einzelnen Hierarchiestufen symbolisieren. Dabei stellt der innerste Kreis die höchste Hierarchiestufe dar, der äußerste die tiefste. Jeder Kreis ist in Segmente unterteilt, deren Flächen proportional den Größen der dargestellten Daten entsprechen.

Mit Hilfe eines Sunburst-Diagramms könnte ein großes Autohaus beispielsweise die Umsatzverteilung über drei Hierarchiestufen hinweg – nämlich Automarken, Kundentypen und Verkäufer – detailliert darstellen. Dazu braucht es zunächst eine Zahlentabelle wie die folgende, die in der Beispieldatei *10\Diagrammtypen2016.xlsm* nach einem Klick auf den „Sunburst“-Button zu finden ist:

Marke	Kundentyp	Verkäufer	Umsatz
Audi	Gewerbe		93
	Endkunden	Müller	32
		Meier	30
		Hansen	30
	Behörden		95
VW	Gewerbe		79
	Endkunden	Harms	14
		Seisner	13
		Engel	12
		Dressen	14
		Madler	14
		Behörden	
Porsche	Gewerbe		48
	Endkunden	Schulz	21
		Hackel	30
		Behörden	
Seat	Gewerbe		24
	Endkunden		25
	Behörden		34

Die Tabelle ist mit dem Bereichsnamen „rngDataSunburst“ hinterlegt. Die Code-Zeilen

```
Set rngData = Range("rngDataSunburst")
rngData.Select
```

selektieren den benannten Bereich und machen ihn damit automatisch zum Datenbereich des anzulegenden Sunburst-Diagramms (siehe nachfolgende Abbildung), das mit der Anweisung

```
Set shpDiagram = ActiveSheet.Shapes.AddChart2(-1, xlSunburst)
```

generiert wird. Wie die meisten neuen Excel-2016-Diagramme verfügt das Sunburst-Diagramm über eine Reihe von spezifischen Eigenschaften, die der Anwender mangels Integration in das Objektmodell von Excel leider nur manuell mit Hilfe der einschlägigen Formatierungsdialoge verändern kann.

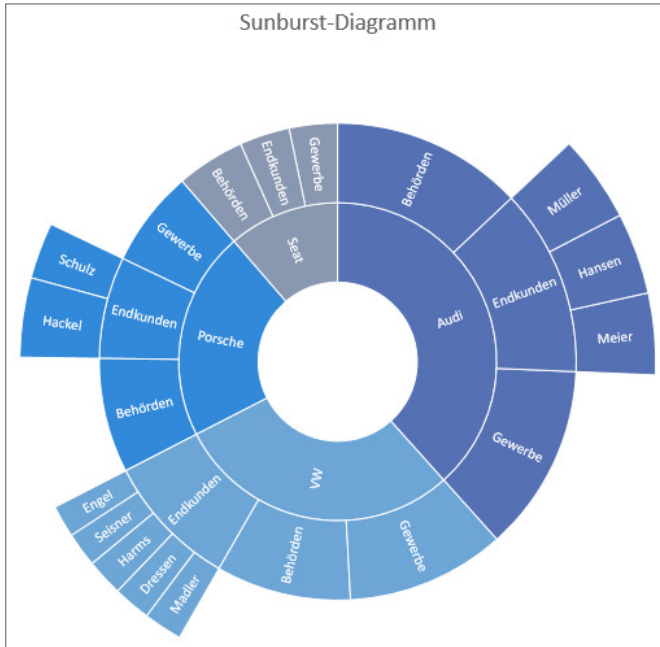


BILD 10.28:
Ein Sunburst-Diagramm stellt für jede Hierarchie-stufe einen Kreis bereit. Dessen Segmente visualisieren die Größenverhältnisse der zugehörigen Daten.

■ 10.9 Zeichnungsobjekte (Shapes)

Überblick

Das *Shape*-Objekt dient primär zur Darstellung von AutoFormen (Linien, Rechtecke, Pfeile, Sterne etc. – siehe EINFÜGEN | FORMEN). Es löst damit die diversen Zeichnungsobjekte aus Excel 5/7 ab. Verwirrung stiftet allerdings die große Anzahl verwandter Objekte.

Hierarchie der Shape-Objekte

Worksheet/Chart	
└ Shapes	alle <i>Shape</i> -Objekte innerhalb des Blatts
└└ Shape	ein <i>Shape</i> -Objekt
└└└ ConnectorFormat	Verbindung zu anderen Objekten
└└└ ControlFormat	zusätzliche Eigenschaften für Steuerelemente
└└└ FillFormat	Hintergrundmuster (via <i>Fill</i> -Eigenschaft)
└└└ GroupShapes	Einzelobjekte (via <i>GroupItems</i> , wenn <i>Type=msoGroup</i>)
└└└└ Shape	
└└└ HyperLink	Querverweise und Internetlinks
└└└ LineFormat	Linieneigenschaften (via <i>Line</i>)

OLE, ActiveX-Automation	
<i>oleob.Delete</i>	löscht OLE-Objekt
<i>oleob.Object</i>	Verweis für ActiveX-Automation
<i>obj = GetObject(„“, „ole-bezeichn“)</i>	Verweis für ActiveX-Automation

■ 15.7 64-Bit-Programmierung

Seit der Versionsnummer 2010 wird Excel sowohl in einer 32- als auch einer 64-Bit-Version angeboten. Letztere bietet den Vorteil, den gesamten Arbeitsspeicher eines 64-Bit-Windows-Systems nutzen zu können. Davon profitiert das Programm insbesondere beim Umgang mit sehr großen Arbeitsmappen, die mehr als 2 GByte RAM für sich beanspruchen dürfen. Ein weiterer 64-Bit-Vorteil ist die sogenannte hardware-gestützte Datenausführungsverhinderung. Die soll Sicherheitslücken im Zusammenhang mit Pufferüberlaufen schließen und damit die Angriffsfläche für Viren und Würmer deutlich verringern.

15.7.1 Kompatibilitätsprobleme

Den genannten Vorteilen stehen allerdings diverse Nachteile gegenüber, die insbesondere die Kompatibilität mit vorhandenen Makros, Add-ins, Datenbanken und sonstigen Excel-Erweiterungen betreffen. So verwendet die 64-Bit-Version von Excel beispielsweise eine ebenso breite Variante des Windows-eigenen *Graphics Device Interface* (GDI), um ihre Diagramme und sonstigen Hochglanzgrafiken zu rendern. Das 64-Bit-GDI unterstützt aber keine MMX-Befehle mehr, die eine besonders schnelle, weil parallele Verarbeitung von Grafik- und Videodaten auf Intel-Prozessoren ermöglichen. Das stellt allen Excel-Erweiterungen mit MMX-gestützten Multimedia-Ambitionen den Stuhl vor die 64-Bit-Tür. Add-ins dieser Art dürfte es allerdings nicht allzu viele geben.

Da wiegt der Ausschluss aller kompilierten Datenbankdateien (*.mde oder *.accde), die je mit einer 32-Bit-Ausgabe von Microsoft Access erstellt wurden, deutlich schwerer. Eine Neukompilierung mit der jüngsten 64-Bit-Variante des Datenbankprogramms kann dieses Problem zwar lösen, erfordert allerdings Zugriff auf die originären ACCDB- oder MDB-Dateien. Die jedoch rücken die Entwickler nur selten heraus, da die verwendeten Programmcodes darin für jedermann einsehbar sind.

Problemfall ActiveX

Noch schwerwiegender dürfte die Verweigerungshaltung von Excel 64 Bit in Bezug auf *ActiveX-Steuerelemente* und *COM-Add-Ins* (siehe Abschnitt 15.1) sein. Dabei handelt es sich durchweg um 32-Bit-Binärdateien, die ein 64-Bit-Prozess grundsätzlich nicht laden kann. Und das ist ein wirkliches Problem, da nahezu jede programmierte Lösung, die die funktionalen Grenzen von Excel wirksam erweitert, COM- und ActiveX-Elemente verwendet: als Userform-Control (Steuerelement) für besondere Aufgaben, als Funktionsbibliothek oder Fernsteuerung für beliebige (COM-fähige) Anwendungen beispielsweise.

Zwar lässt sich auch dieses Problem grundsätzlich durch Neukompilierung beseitigen. Dazu braucht es allerdings einen geeigneten 64-Bit-Compiler, sämtliche Quellcodes sowie ein nicht unerhebliches Know-how. Als Endanwender ohne Programmiererfahrung wird man daher in der Regel warten müssen, bis der Software-Hersteller eine 64-Bit-Version seines ActiveX-Controls beziehungsweise COM-Add-Ins herausgibt.

Problemfall Windows-API

Wenn ein Excel-Entwickler die VBA-Grenzen sprengen will, verwendet er ebenfalls sehr häufig das *Application Programming Interface* (API) von Windows, das seine zahllosen Funktionen in Form von Dynamic Link Libraries (siehe Abschnitt 15.5) zur Verfügung stellt. Für die Verwaltung von Fenster-Handles und Adresszeigern verwenden API-Funktionen standardmäßig den 32-Bit-Datentyp *Long* – was in einem 64 Bit breiten Adressraum schwere Komplikationen bis hin zum Programmabsturz verursachen kann.

Probleme dieser Art kann der Entwickler allerdings selbst lösen – mit den einschlägigen Werkzeugen der neuen VBA-Version 7.0, die wir Ihnen im Folgenden anhand eines praktischen Beispiels vorstellen möchten.



Hinweis

Microsoft bietet ein nützliches Tool an, mit dem Sie die 64-Bit-Verträglichkeit vorhandener Excel-Erweiterungen überprüfen können. Sie finden den *Microsoft Office Code Compatibility Inspector* unter der folgenden Adresse [Link 35]:

<http://www.microsoft.com/downloads/details.aspx?FamilyID=23C8A7F6-88B3-48EF-9710-9742340562C0>

15.7.2 Ein problematisches (32-Bit-)Beispiel

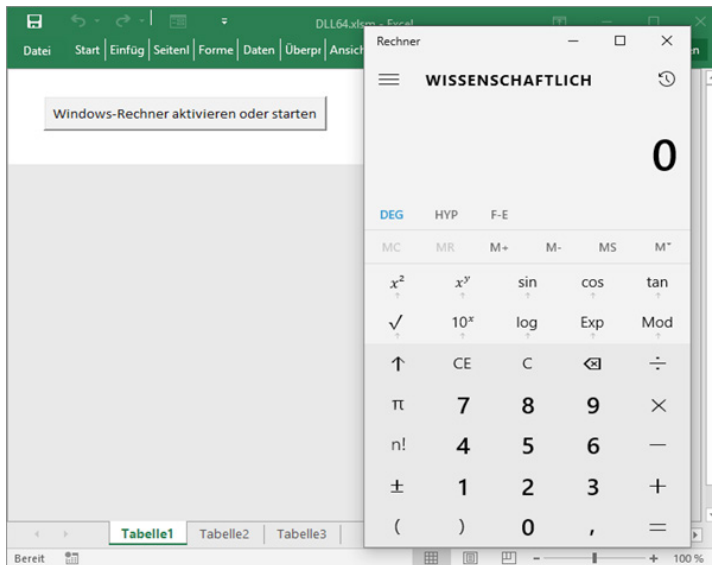


BILD 15.13: Das Beispielprogramm nutzt diverse API-Funktionen, die den Windows-Rechner je nach Zustand starten oder aktivieren.

Wenn ein VBA-Makro eine externe Anwendung wie den Windows-Rechner beispielsweise aufrufen soll, muss es diese natürlich starten können, was mit der *Shell*-Anweisung von VBA (siehe Abschnitt 15.6.6) auch keinerlei Probleme bereitet. Um mehrfache Starts und die damit verbundene Verschwendung von Rechnerressourcen zu vermeiden, sollte das Makro allerdings vorher prüfen, ob die Anwendung nicht bereits läuft. Mit VBA geht das nicht, das Windows-API hält dagegen gleich mehrere Lösungen bereit.

Die meistgenutzte Lösung dürfte der Einsatz der Funktion *FindWindow* sein, die man nach Auskunft der üblichen API-Dokumentationen wie folgt deklarieren muss, um sie in einem Makro nutzen zu können:

```
Declare Function FindWindow Lib "user32" Alias "FindWindowA" _
    (ByVal lpClassName As String, ByVal lpWindowName As String) _
    As Long
```

Wie man sieht, besitzt die *FindWindow*-Funktion zwei Argumente, die der zweifelsfreien Identifikation der gewünschten Anwendung dienen. Das erste Argument *lpClassName* benennt den sogenannten Klassennamen, eine vom jeweiligen Programmierer festgelegte Bezeichnung der zentralen Anwendungskomponente, die beim Windows-Rechner „Calc-Frame“ lautet (der Klassename von Excel ist „XLMAIN“, der von Word „OpusApp“). Kennt man den Klassennamen nicht, gibt sich *FindWindow* auch mit dem Titel des Anwendungsfensters zufrieden, den man im zweiten Argument *lpWindowName* übergibt. Im Fall des Windows-Rechners lautet dieser schlicht „Rechner“.

Findet *FindWindow* ein Anwendungsfenster, das den angegebenen Argumenten entspricht, dann liefert die API-Funktion im Gegenzug dessen „Handle“ zurück. Das ist eine eindeutige Fensternummer, die das aufrufende Makro – ebenfalls laut API-Dokumentation – in einer Variablen vom Typ *Long* entgegenzunehmen hat. Das könnte ungefähr so aussehen:

```
Dim lngWindowHandle As Long
lngWindowHandle = FindWindow(vbNullString, „Rechner“)
```

Mit einer simplen *If-Then-Else*-Abfrage wie der folgenden kann das Makro dann schnell die passenden Schlüsse ziehen und angemessen reagieren:

```
If lngWindowHandle = 0 Then
    Shell "calc.exe"
Else
    Dim lngResult As Long
    lngResult = ShowWindow(lngWindowHandle, SW_RESTORE)
    lngResult = SetForegroundWindow(lngWindowHandle)
End If
```

Die *If*-Anweisung prüft den Wert des zurückgelieferten Handles. Ist der gleich null, läuft der Windows-Rechner offensichtlich nicht, da er ja kein (nummeriertes) Anwendungsfenster besitzt. In dem Fall startet das Makro den Rechner per *Shell*-Anweisung. Liegt der Handle-Wert aber über null, gibt es bereits ein Anwendungsfenster, das nur noch aktiviert werden muss. Das erledigt das Makro im *Else*-Abschnitt mithilfe von zwei weiteren API-Funktionen: Die *ShowWindow*-Funktion bringt das Anwendungsfenster zunächst in seine normale Größe (es könnte ja zum Symbol verkleinert sein), so dass es die *SetForegroundWindow*-Funktion in den Vordergrund holen kann (es könnte ja von anderen Fenstern verdeckt sein).

Der Datentyp LongPtr

Wenn Sie das obige Beispiel mit der 32-Bit-Version von Excel ausführen, werden Sie keinerlei Probleme bemerken. Die ergeben sich erst mit der 64-Bit-Version des Kalkulationsprogramms. Auslöser ist *lngWindowHandle*, eine Variable vom Datentyp *Long*, die das von *FindWindow* bereitgestellte Handle des Anwendungsfensters aufnehmen soll. Und das funktioniert in einer 64-Bit-Umgebung nicht, da Handles hier volle 64 Bit beanspruchen, von denen der stets nur 32 Bit „breite“ Datentyp *Long* dann nur noch die Hälfte speichern kann. Das gleiche Problem tritt übrigens auch bei „Pointern“ auf, das sind vom Windows-API gelieferte Variablen, die auf bestimmte Speicheradressen verweisen.

Damit die Zuweisung von Handles und Pointern nun auch ohne kapitalen Absturz in einem 64-Bit-Excel gelingt, hat Microsoft den Sprachumfang von VBA erstmals seit vielen Jahren wieder (und ausschließlich zu diesem Zweck) erweitert. Wichtigste diesbezügliche Errungenschaft der VBA-Version 7.0 ist der „intelligente“ Datentyp *LongPtr*. Der passt seine Bit-Breite automatisch an die der verwendeten Excel-Version an: Im Fall eines 32-Bit-Excels speichert er also 32-Bit-Werte, bei einem 64-Bit-Excel nimmt er entsprechend 64-Bit-Werte auf.

Somit eignet sich der Datentyp *LongPtr* ideal für die Aufnahme von Handles und Pointern, die als Argument oder Rückgabewert von API-Funktionen in Erscheinung treten. Wurden die zugehörigen Variablen bislang „As Long“ definiert, so müssen diese Definitionen nun also einfach in „As LongPtr“ geändert werden, um eine vorhandene Excel-Lösung 64-Bit-tauglich zu machen.

Der *FindWindow*-Aufruf aus unserem Beispiel würde dann so aussehen:

```
' Beispiel 15\DLL64.xlsm
Dim lngWindowHandle As LongPtr
lngWindowHandle = FindWindow(vbNullString, „Rechner“)
```



Hinweis

Die vermeintliche „Intelligenz“ des Datentyps *LongPtr* ist nichts weiter als ein cleverer Trick. Der besteht darin, sämtliche *LongPtr*-Variablen je nach Bit-Breite der Excel-Version in den real existierenden Datentyp *Long* (32 Bit) beziehungsweise dessen neuesten Kollegen *LongLong* (64 Bit) umzuwandeln.

Das Schlüsselwort PtrSafe

Mit der makrointernen Umstellung der Variablen *lngWindowHandle* auf den Datentyp *LongPtr* ist es aber nicht getan. Schließlich „weiß“ Excel ja noch gar nicht, dass es sich beim Rückgabewert von *FindWindow* um ein Handle mit variabler Bitbreite handelt. Damit der ausführende VBA-Interpreter von diesem Umstand Kenntnis erhält, muss man die Deklarationszeile der API-Funktion ebenfalls wie folgt ändern:

```
' Beispiel 15\DLL64.xlsm
Declare PtrSafe Function FindWindow Lib "user32" Alias _
    "FindWindowA" (ByVal lpClassName As String, ByVal _
        lpWindowName As String) As LongPtr
```

Die Unterschiede zum Original liegen nicht nur am Ende der Deklarationszeile, wo der Datentyp des Rückgabewerts von *Long* in *LongPtr* geändert wurde, sondern auch in der zusätzlichen Verwendung des Schlüsselworts *PtrSafe*. Es informiert den VBA-Interpreter darüber, dass die *Declare*-Anweisung für die 64-Bit-Version von Excel geschrieben wurde. Ohne dieses Schlüsselwort tritt bei Verwendung der *Declare*-Anweisung auf einem 64-Bit-System ein Kompilierungsfehler auf. Bei der 32-Bit-Version von Excel ist das *PtrSafe*-Schlüsselwort optional. Auf diese Weise wird die Funktion vorhandener *Declare*-Anweisungen nicht beeinträchtigt.

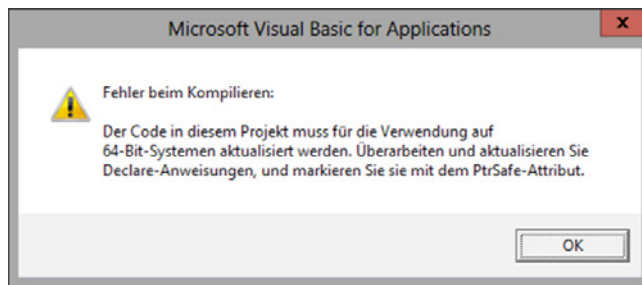


BILD 15.14:

In der 64-Bit-Version von Excel müssen *Declare*-Anweisungen zwingend das Schlüsselwort *PtrSafe* enthalten, sonst gibt es Gemecker in Form dieser Fehlermeldung.

Während die Kennzeichnung durch das Schlüsselwort *PtrSafe* also zwingend ist für den 64-Bit-Einsatz, sollte man Datentypänderungen in *Declare*-Anweisungen nur sehr gezielt vornehmen. Die Online-Hilfe von VBA und viele Internetquellen erwecken zwar den Eindruck, als müsste man den Datentyp *sämtlicher* Argumente oder Rückgabewerte von *Long* auf *LongPtr* umstellen. Tatsächlich ist das aber nur bei solchen Argumenten oder Rückgabewerten erforderlich, die für die bilaterale Weitergabe eines Handles oder Pointers verantwortlich sind. Alle anderen Argumente oder Rückgabewerte dürfen ihrem Datentyp *Long* unverändert treu bleiben.

Bei der Deklaration der API-Funktionen *ShowWindow* und *SetForegroundWindow* ist eine Datentypänderung somit nur für das Argument *hwnd* notwendig, das in beiden Fällen das von *FindWindow* gelieferte Handle des Anwendungsfensters übergibt. Als Rückgabewert geben beide Funktionen einen numerischen (Fehler-)Code zurück, der in jedem Fall in einer *Long*-Variablen Platz findet. Die 64-Bit-tauglichen *Declare*-Anweisungen sehen folglich so aus:

```
' Beispiel 15\DLL64.xlsm
Declare PtrSafe Function ShowWindow Lib "user32" (ByVal _
    hwnd As LongPtr, ByVal nCmdShow As Long) As Long
Declare PtrSafe Function SetForegroundWindow Lib "user32" (ByVal _
    hwnd As LongPtr) As Long
```

Da die beiden API-Routinen somit auch in einer 64-Bit-Umgebung „nur“ einen *Long*-Wert zurückgeben, genügt für dessen Aufnahme innerhalb des Makros selbstverständlich auch weiterhin eine Variable desselben Typs. Der entsprechende Aufrufcode in der *If-Then-Else*-Abfrage unseres Beispiels erfordert daher keinerlei Änderungen:

```
' Beispiel 15\DLL64.xlsm
Dim lngResult As Long
lngResult = ShowWindow(lngWindowHandle, SW_RESTORE)
lngResult = SetForegroundWindow(lngWindowHandle)
```

Bedingte Kompilierung

Der Datentyp *LongPtr*, das Schlüsselwort *PtrSafe* und einige andere Elemente, die vollständig in der nachfolgenden Syntaxzusammenfassung dokumentiert sind, sind Neuerungen der VBA-Version 7.0 und stehen damit – vom Datentyp *LongLong* einmal abgesehen – sowohl in der 64- als auch der 32-Bit-Version von Excel zur Verfügung. Somit dürfte jeder VBA-Code, der ursprünglich für die 64-Bit-Fassung von Excel entwickelt wurde und die neuen Elemente für den Umgang mit API-Funktionen nutzt, auch in der 32-Bit-Version funktionieren. Die Codeverträglichkeit endet allerdings schon bei der Excel-Version 2007, die es ja ausschließlich in einer 32-Bit-Fassung gibt und deren VBA-Version 6.5 Elemente wie *LongPtr*, *PtrSafe* & Co natürlich völlig unbekannt sind.

Man kann trotzdem „allgemeingültigen“ VBA-Code schreiben, der API-Funktionen nutzt und dennoch mit allen Bitbreiten und VBA-Varianten bis hinunter zur Version 6.0 (die mit Excel 2000 eingeführt wurde) zurecht kommt. Das Mittel der Wahl heißt „Bedingte Kompilierung“ und wurde in Excel 2010 um eine neue Konstante namens *VBA7* erweitert. Die ermöglicht eine saubere Codetrennung für die jüngste VBA-Version und alle VBA-Versionen davor. Das Grundmuster sieht so aus:

```
#If VBA7 Then
  'Anweisungen für VBA 7.0
#Else
  'Anweisungen für frühere VBA-Versionen
#End If
```

Wird der obige Code in Excel (egal, ob 32 oder 64 Bit) ausgeführt, pickt sich der VBA-Interpreter exklusiv die Anweisungen heraus, die im *#If*-Abschnitt durch die Kompilierungskonstante *VBA7* gekennzeichnet sind. Älteren VBA-Interpretern ist die Konstante unbekannt; sie verzweigen daher automatisch in den *#Else*-Abschnitt des Codeblocks.

Im Fall unseres Beispiels würde man die bedingte Kompilierung unter anderem für eine versionsgerechte Deklaration der beteiligten API-Funktionen einsetzen, und zwar so:

```
' Beispiel 15\DLL64.xlsm
#If VBA7 Then
  Declare PtrSafe Function FindWindow Lib "user32" Alias _
    "FindWindowA" (ByVal lpClassName As String, ByVal _
    lpWindowName As String) As LongPtr
  Declare PtrSafe Function ShowWindow Lib "user32" _
    (ByVal hwnd As LongPtr, ByVal nCmdShow As Long) As Long
  Declare PtrSafe Function SetForegroundWindow Lib _
    "user32" (ByVal hwnd As LongPtr) As Long
#Else
  Declare Function FindWindow Lib "user32" Alias _
    "FindWindowA" (ByVal lpClassName As String, ByVal _
    lpWindowName As String) As Long
  Declare Function ShowWindow Lib "user32" (ByVal hwnd As _
    Long, ByVal nCmdShow As Long) As Long
  Declare Function SetForegroundWindow Lib "user32" (ByVal _
    hwnd As Long) As Long
#End If
```

Darüber hinaus käme die bedingte Kompilierung auch innerhalb des Makros für die Aktivierung des Windows-Rechners zum Einsatz, und zwar bei der Deklaration der Variablen *lngWindowHandle*:

```
' Beispiel 15\DLL64.xlsm
Public Sub RechnerAktivierenOderStarten()
  #If VBA7 Then
    Dim lngWindowHandle As LongPtr
  #Else
    Dim lngWindowHandle As Long
  #End If
  lngWindowHandle = FindWindow(vbNullString, "Rechner")
  If lngWindowHandle = 0 Then
    Shell "calc.exe"
  Else
    Dim lngResult As Long
    lngResult = ShowWindow(lngWindowHandle, SW_RESTORE)
    lngResult = SetForegroundWindow(lngWindowHandle)
  End If
End Sub
```



Hinweis

Das vollständige Beispiel finden Sie in der Datei *DLL64.xlsm* im Unterordner 15 der Beispieldateien.

15.7.3 Syntaxzusammenfassung

Neue Elemente von VBA 7.0 für die 64-Bit-Programmierung	
<i>CLngLng</i>	konvertiert einen Wert in den Datentyp LongLong
<i>CLngPtr</i>	konvertiert einen Wert in den Datentyp LongPtr
<i>LongLong</i>	8-Byte-Datentyp, der nur in 64-Bit-Versionen von Excel 2010 (und neuer) zur Verfügung steht
<i>LongPtr</i>	variabler Datentyp, der auf 32-Bit-Versionen von Excel 2010 (und neuer) als 4-Byte-Datentyp (Long) und auf 64-Bit-Versionen als 8-Byte-Datentyp (LongLong) ausgelegt ist
<i>ObjPtr</i>	Objektkonverter; gibt auf 64-Bit-Versionen LongPtr und auf 32-Bit-Versionen Long zurück
<i>PtrSafe</i>	gibt an, dass die Declare-Anweisung mit 64-Bit-Systemen kompatibel ist
<i>StrPtr</i>	Zeichenfolgenkonverter; gibt auf 64-Bit-Versionen LongPtr und auf 32-Bit-Versionen Long zurück
<i>VarPtr</i>	Variantenkonverter; gibt auf 64-Bit-Versionen LongPtr und auf 32-Bit-Versionen Long zurück
<i>VBA7</i>	Konstante, die die bedingte Kompilierung von Anweisungsblöcken für VBA 7 und ältere VBA-Versionen ermöglicht

Mit einem Druck auf F5 führen Sie das Projekt aus. Die Aktion startet Excel mit einer neuen Arbeitsmappe und dem neuen Aufgabenbereich am rechten Rand des Programmfensters. Zum Ausprobieren tippen Sie Ihre ganz persönliche Bankleitzahl (ohne Leerzeichen!) in eine beliebige Zelle, markieren diese anschließend und klicken dann auf die Schaltfläche BANKINFOS ABRUFEN.

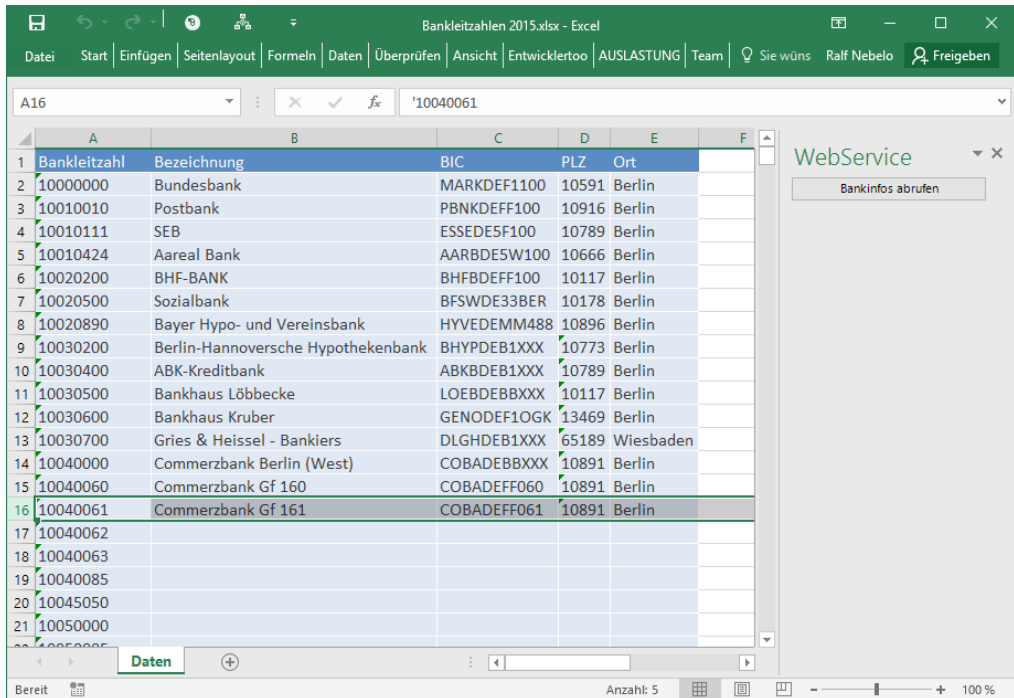


BILD 15.25: Das fertige VSTO-Projekt demonstriert, wie leicht sich der BIC und andere Bankdaten mit der Hilfe eines Webdienstes abrufen lassen. Dazu muss man diesem nur eine gültige Bankleitzahl übergeben.

15.9 Office-Add-ins

Hinter dem neuen Befehl MEINE ADD-INS im Menüband EINFÜGEN von Excel 2016 verbirgt sich ein mit Office 2013 eingeführtes und damals radikal neues Erweiterungskonzept, das sogenannte „Office-Add-ins“ (die in Office 2013 noch „Office-Apps“ hießen) an die Stelle von VBA-Makros setzen und Webtechniken und Cloud-Dienste direkt in die Bedienoberflächen aller Office-Anwendungen integrieren will.

Obwohl sich Office-Add-ins auch lokal bereitstellen lassen – was wir in diesem Kapitel anhand von zwei Beispielen demonstrieren werden –, erhält man sie vorzugsweise im *Office Store*. Das ist ein Online-Shop, den man direkt aus den Office-Anwendungen heraus erreicht.

Der Office Store präsentiert das bislang immer noch vorwiegend englischsprachige Angebot an Office-Add-ins in übersichtlicher Form und reduziert die Installation des gewählten Helferleins auf wenige Mausklicks. Die schnelle Verfügbarkeit macht Office-Add-ins für Anwender und damit natürlich auch für Entwickler attraktiv. Grund genug also, sich mit den Grundlagen der neuen Office-Add-ins vertraut zu machen.

15.9.1 Bestandteile eines Office-Add-ins

Ein Office-Add-in ist im Grunde eine normale Webanwendung, die aber nicht auf einem Server im Internet, sondern in Office „gehostet“ und auch angezeigt wird. Das typische Office-Add-in besteht aus den folgenden Komponenten:

- **Webseite:** Dabei handelt es sich um eine (halbwegs) normale HTML-Datei, die man wie jede andere Webseite auch mit Steuerelementen aus dem HTML-, ASP.NET-, Silverlight- oder Flash-Fundus bestücken kann. Das Aussehen der Webseite kann der Entwickler über standardmäßige Webtechniken wie HTML5, XML und CSS3 bestimmen. Die Webseite bildet die Bedienoberfläche des Office-Add-ins.
- **Skriptdatei:** Das ist zumeist eine JavaScript-Datei, die den Programmcode des Add-ins enthält und damit für die „Action“ zuständig ist. Wie bei regulären Webanwendungen wird die Verbindung zwischen Webseite und Skriptdatei häufig über das *onclick*-Attribut der Steuerelemente hergestellt. Es weist dem jeweiligen Steuerelement eine Ereignisroutine in der Skriptdatei zu. Der Code darin bestimmt, was beim Anklicken des Steuerelements geschieht.

Grundsätzlich lässt sich die gewünschte Funktionalität eines Office-Add-ins aber nicht nur über eine externe Skriptdatei, sondern über *jede* Art der client- oder serverseitigen Programmierung bereitstellen. Das kann im einfachsten Fall ein in die Webseite eingebettetes Skript sein, in anspruchsvolleren Szenarien eine komplexe ASP.NET-Anwendung. Über REST-APIs kann ein Office-Add-in so gut wie jeden Web Service (siehe Abschnitte 15.4 und 15.8.4) zur Informationsbeschaffung anzapfen.

- **Icon-Datei (optional):** Diese BMP-, GIF-, EXIF-, JPG-, PNG- oder TIFF-Datei enthält das Icon des Office-Add-ins, das eine Größe von 32 mal 32 Pixel aufweisen muss. Fehlt die Icon-Datei oder weist sie eine andere Bildgröße auf, erhält die App ein monochromes Standard-Icon zugewiesen.
- **OfficeStyles.css (optional):** Diese Datei stellt das Stylesheet für das Office-Add-in bereit und weist allen Bestandteilen der Webseite die Office-typischen Schriftarten und Formatierungen zu.
- **Manifestdatei:** Diese XML-Datei ist das Bindeglied zwischen den einzelnen Add-in-Elementen. Die Manifestdatei verrät der Office-Anwendung unter anderem, wo die Webseiten- und die Icon-Datei (falls vorhanden) zu finden sind. Darüber hinaus definiert sie die Einstellungen des Add-ins, seine Fähigkeiten und Rechte.
- **JavaScript-API für Office:** Diese von Microsoft bereitgestellte und online verfügbare Bibliothek (die eine JavaScript-Datei ist) stellt die Verbindung zwischen der Office-Anwendung und der Add-in-Webseite her. Die Bibliothek sorgt dafür, dass das Add-in unter anderem auf Inhalte des Dokuments zugreifen oder mit der als Host fungierenden Office-Anwendung kommunizieren kann.

Die Risiken einer solchen Interaktion zwischen Anwendung und Add-in will Microsoft insbesondere durch eine sorgsame Trennung der beiden klein halten. Dazu sperrt der Hersteller die HTML-Datei in ein Webbrowser-Control und lässt dieses in einem von der Host-Anwendung unabhängigen Prozess ausführen. Zu den weiteren Sicherheitsmaßnahmen gehört eine restriktive Laufzeitumgebung. Die überwacht jede Interaktion über Prozessgrenzen hinweg und gestattet Zugriffe auf die Daten und die Bedienoberfläche der Office-Anwendung grundsätzlich nur über asynchrone Methoden, die den Office-Prozess weder ausbremsen noch zum Entgleisen bringen können.

Die Auflistung der Add-in-Elemente lässt bereits erahnen, dass die Entwicklung von Office-Add-ins vorzugsweise in die Zuständigkeit erfahrener Webentwickler fällt. Klassische Office-Entwickler, die sich „nur“ mit VBA und eventuell noch mit den Visual Studio Tools for Office (VSTO, siehe Abschnitt 15.8) auskennen, werden sich in einer neuen Programmierwelt zurechtfinden müssen.

Aus diesem Grund kann das vorliegende Kapitel auch kaum mehr als eine Einführung in das Thema sein. Für das Verständnis der folgenden Ausführungen und Codepassagen sind grundlegende HTML-, XML- und JavaScript-Kenntnisse erforderlich. Wer weiterführende Infos benötigt, findet diese auf Microsofts Portal für Office-Add-in-Entwickler [Link 44]:

<https://msdn.microsoft.com/de-de/library/office/jj220060.aspx>

15.9.2 Typen von Office-Add-ins

Es gibt drei Typen von Office-Add-ins, die man je nach ihrem „Einsatzort“ unterscheidet:

- **Aufgabenbereich-Add-in:** Dieses Office-Add-in zeigt sich im Aufgabenbereich der Office-Anwendung am rechten Rand des Dokumentfensters. Add-ins dieser Art eignen sich insbesondere dazu, dem Anwender kontextbezogene Informationen und Funktionen – für die Suche oder das Übersetzen von Inhalten beispielsweise – zu liefern.
- **Inhalts-Add-in:** Dieses Office-Add-in erscheint ähnlich wie ein Excel-Diagramm als abgegrenzter Bereich innerhalb des Dokuments. Inhalts-Add-ins eignen sich vor allem für die Visualisierung von Daten oder die Wiedergabe von YouTube-Videos und anderen Internetmedien.
- **Mail-Add-in:** Es klinkt sich in Outlook-Formulare ein und stellt dem Anwender dort maßgeschneiderte Infos und Funktionen bereit, die sich auf das jeweils angezeigte Outlook-Element – eine Nachricht, eine Besprechungsanfrage oder einen Termin etwa – beziehen. Mail-Add-ins funktionieren nur im Zusammenspiel mit Microsoft Exchange ab Version 2013, normale POP- und IMAP-Konten werden nicht unterstützt.

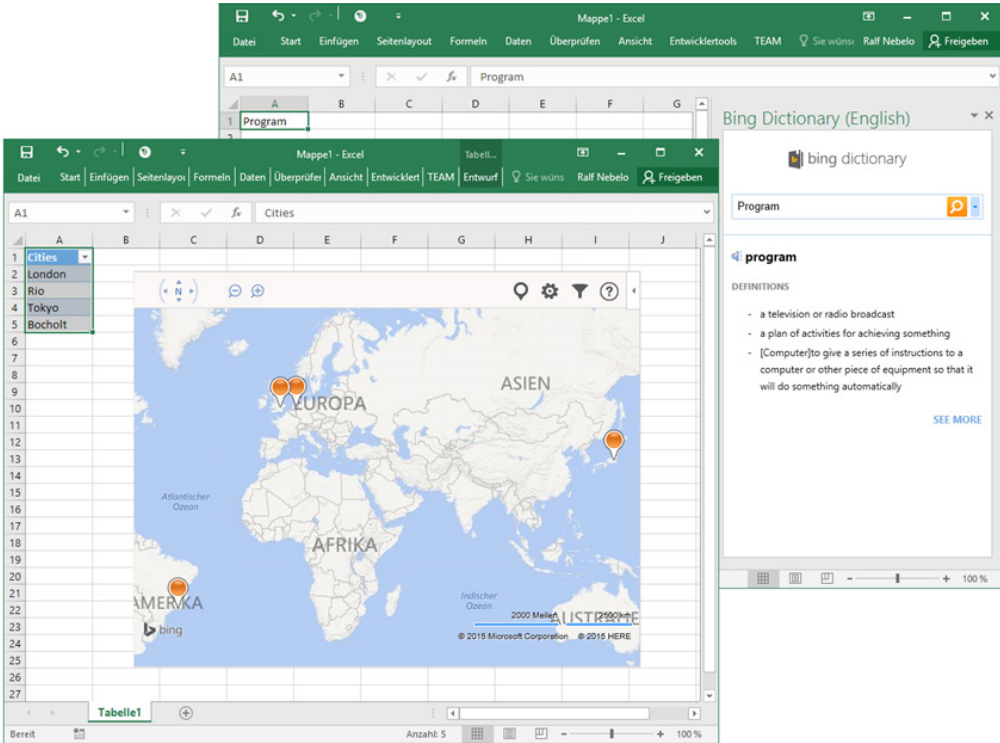


BILD 15.26: Microsofts Suchmaschine Bing stellt Excel-Anwendern unter anderem ein Aufgabenbereich-Add-in (hinten) sowie ein Inhalts-Add-in zur Verfügung.

Während Mail-Add-ins quasi naturgemäß auf Outlook festgelegt sind, sollten Inhalts- und Aufgabenbereich-Add-ins von der Konzeption her eigentlich in allen Office-Anwendungen nutzbar sein, die der Bearbeitung von Dokumenten dienen. Momentan unterstützen aber nur Excel und PowerPoint 2016 beide Add-in-Typen, während sich Project und Word nur mit Aufgabenbereich-Add-ins erweitern lassen. Die Tabelle zeigt den aktuellen Stand, der sich jedoch jederzeit ändern kann.

Office-Anwendung	Inhalts-Add-ins	Mail-Add-ins	Aufgabenbereich-Add-ins
Access 2016	-	-	-
Access-Web-App	+	-	-
Excel 2016	+	-	+
Excel-Web-App	+	-	+
Excel für iPad	+	-	+
Outlook 2016	-	+	-
Outlook für Mac	-	+	-
Outlook-Web-App	-	+	-
PowerPoint 2016	+	-	+

Office-Anwendung	Inhalts-Add-ins	Mail-Add-ins	Aufgabenbereich-Add-ins
PowerPoint-Web-App	+	-	+
Project 2016	-	-	+
Word 2016	-	-	+
Word-Web-App	-	-	+
Word für iPad	-	-	+

Anders als ein konventionelles (COM-)Add-in, das man für jede Applikation separat entwickeln muss, lässt sich ein Office-Add-ins aber sehr leicht so gestalten, dass es mit demselben Code in jeder Office-Anwendung funktioniert, die den jeweiligen Add-in-Typ unterstützt.

15.9.3 Werkzeuge für die Entwicklung von Office-Add-ins

Auch wenn die Entwicklung von Office-Add-ins einiges Wissen über Webtechniken erfordert, so benötigt man doch erfreulich wenig Handwerkszeug dafür: Ein schlichter Texteditor genügt für den Anfang (und die Realisierung des nachfolgenden Beispiel-Add-ins).

Allerdings sollte man es mit der Bescheidenheit auch nicht übertreiben, indem man sich mit dem Windows-eigenen Notepad begnügt. Wesentlich angenehmer lässt es sich beispielsweise mit *Notepad++* arbeiten, da es unter anderem mehrere Fenster für die Bearbeitung der verschiedenen Projektdateien unterstützt. Die Download-Adresse für das nützliche Open-Source-Werkzeug lautet zuletzt wie folgt [Link 40]:

<http://notepad-plus-plus.org>

Wer es komfortabler mag, verwendet die *Napa Office 365 Development Tools*. Dabei handelt es sich um eine webbasierte Entwicklungsumgebung, mit der Sie im Browser Projekte erstellen, Code schreiben und Ihre Apps ausführen können. Weitere Informationen finden Sie hier [Link 45]:

<http://msdn.microsoft.com/de-de/library/office/apps/jj220038>

Das mit Abstand komfortabelste, leistungsfähigste, aber auch teuerste Werkzeug zum Erstellen eines Office-Add-ins ist *Visual Studio (ab Version 2012)*. Die Entwicklungsumgebung verfügt über eine spezielle *App für Office*-Projektvorlage, die dem Entwickler viele Handgriffe, die er ansonsten manuell erledigen müsste, erspart. Darüber hinaus beschleunigt Visual Studio die Office-Add-in-Entwicklung mit einer Projektmappe, die eine vorkonfigurierte Manifestdatei, Skriptbibliotheken, Stylesheets sowie HTML- und JavaScript-Ausgangsdateien enthält.

15.9.4 Beispiel 1: SimpleApp

Die Arbeit an unserem ersten Beispiel-Add-in beginnt mit dem Anlegen einer Ordnerstruktur, die die diversen Projektdateien aufnimmt. Dazu benötigen Sie eine Netzwerkfreigabe, die auf einem Rechner Ihres Firmen- respektive Heimnetzes oder einer Netzwerkfestplatte

(NAS) liegen kann. Lokale Festplatten kommen nicht in Betracht, da Office für Speicherortangaben ausschließlich echte URLs (ohne „file:///“-Präfix) akzeptiert.

Fügen Sie der Netzwerkfreigabe zunächst ein neues Stammverzeichnis für die Add-in-Entwicklung hinzu, das Sie mit *OfficeApps* benennen. Dieses Stammverzeichnis dient unter anderem der Aufnahme der Manifestdatei(en) (und ist damit gleichzeitig auch der Manifestordner). Fügen Sie dem Stammverzeichnis einen Unterordner namens *SimpleApp* hinzu, der die Heimat des Beispiel-Add-ins bildet.

Die Webseite von SimpleApp

Zunächst erstellen Sie die Webseite des Beispiel-Add-ins, indem Sie eine HTML-Datei mit folgendem Inhalt anlegen (die erste Zeile bitte weglassen, sie verweist nur auf den Speicherort der Datei bei den Download-Dateien zum Buch). Speichern Sie die HTML-Datei anschließend unter dem Namen *SimpleApp.html* im Unterordner *SimpleApp*.

```
<! 15\OfficeApps\SimpleApp\SimpleApp.html >
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=Edge"/>
    <link rel="stylesheet" type="text/css"
      href="../OfficeStyles.css" />
    <script src="SimpleApp.js"></script>
  </head>
  <body>
    <input type="text" value=http://www.hanser-fachbuch.de
      id="TextBox1" style="margin-top: 10px; width: 280px" />
    <input type="button" value="Öffnen" id="Button1" onclick=
      "doAction()" style="margin-top: 10px; margin-right: 10px;
      padding: 0px; width: 100px;" />
  </body>
</html>
```

Wer sich mit HTML ein wenig auskennt, erkennt rasch, dass sich die Struktur dieser Add-in-Webseite kaum von jeder anderen Webseite unterscheidet. Erklärungsbedürftig ist allenfalls das zweite *meta*-Element innerhalb des HTML-Kopfs, das den Internet Explorer zur bevorzugten Rendering-Engine erhebt.

Das *link*-Element stellt einen Verweis auf das Stylesheet *OfficeStyles.css* im übergeordneten Verzeichnis (..) her. Da diese CSS-Datei ziemlich umfangreich ist, kopieren Sie sie der Einfachheit halber von den Download-Dateien zum Buch in Ihren neu angelegten *OfficeApps*-Ordner, wo sie künftig allen Apps zur Verfügung steht.

Das *script*-Element erhebt die Datei *SimpleApp.js* im gleichen Verzeichnis in den Rang der zuständigen Skriptdatei.

Die Anweisungen innerhalb des *body*-Blocks statten die Webseite mit zwei HTML-Controls aus, einer Textbox und einem Button. In der Textbox erscheint die Webadresse <http://www.hanser-fachbuch.de> als Vorgabewert, dem Button wird per *onclick*-Attribut eine Funktion namens „doAction“ als Ereignisroutine für das Anklicken zugewiesen.

Die Skriptdatei von SimpleApp

Die oben erwähnte *doAction*-Funktion bildet den einzigen Inhalt der nachfolgend abgedruckten Skriptdatei *SimpleApp.js*. Speichern Sie diese – genau wie die HTML-Datei zuvor – im Unterordner *SimpleApp*. Die erste Zeile können Sie beim Abtippen wieder weglassen.

```
/* 15\OfficeApps\SimpleApp\SimpleApp.js */
function doAction() {
    var url = document.getElementById("TextBox1").value
    window.location.href = url;
}
```

Der Inhalt der *doAction*-Funktion ist schnell erklärt. Die erste Anweisungszeile liest den aktuellen Inhalt der Textbox, bei dem es sich um eine gültige Webadresse handeln sollte, in die Variable *url* ein. Die zweite Zeile öffnet dann ein Browser-Fenster, das den Inhalt der entsprechenden Internetseite anzeigt.

Die Manifestdatei von SimpleApp

Die Manifestdatei unseres Beispiel-Add-ins hat folgenden Inhalt (erste Zeile bitte wieder weglassen):

```
<! 15\OfficeApps\SimpleApp.xml >
<?xml version="1.0" encoding="utf-8"?>
<OfficeApp
  xmlns="http://schemas.microsoft.com/office/appforoffice/1.1"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:type="TaskPaneApp">
  <Id>08afd7fe-1631-42f4-84f1-5ba51e242f11</Id>
  <Version>1.0</Version>
  <ProviderName>Ralf Nebel</ProviderName>
  <DefaultLocale>EN-US</DefaultLocale>
  <DisplayName DefaultValue="SimpleApp"/>
  <Description DefaultValue="Mein erstes Office-Add-in"/>
  <IconUrl DefaultValue=
    "\\NAS-Archive\Public\OfficeApps\SimpleApp\SimpleApp.png"/>
  <Hosts>
    <Host Name="Document"/>
    <Host Name="Workbook"/>
  </Hosts>
  <DefaultSettings>
    <SourceLocation DefaultValue=
      "\\NAS-Archive\Public\OfficeApps\SimpleApp\SimpleApp.html"/>
  </DefaultSettings>
  <Permissions>ReadWriteDocument</Permissions>
</OfficeApp>
```

Speichern Sie diese Manifestdatei unter dem Namen *SimpleApp.xml* im Stammverzeichnis (respektive Manifestordner) *OfficeApps*.

**Hinweis**

Im Normalfall werden Manifestdateien *nicht* lokal gespeichert. Das ist nur für die Dauer der Programmierung der Fall oder wenn das Office-Add-in ausschließlich für Teilnehmer des eigenen Heim- oder Firmennetzwerks gedacht ist. Möchte man ein Office-Add-in in Microsofts eingangs erwähntem Office Store veröffentlichen (und es damit weltweit für *alle* Office-Anwender verfügbar machen), so leitet man das Manifest im Rahmen eines offiziellen Anmelde- und Prüfverfahrens an Microsoft weiter. Informationen darüber finden Sie hier [Link 41]:

<https://msdn.microsoft.com/de-DE/library/office/fp123515>

Die HTML-Datei und alle weiteren Bestandteile der App gehören dann auf einen öffentlich erreichbaren Webserver, dessen Adresse im Manifest zu vermerken ist.

Der konstruktive Inhalt der abgedruckten Manifestdatei beginnt mit einem XML-Element namens *OfficeApp*, das neben dem verwendeten XML-Schema gleich auch den Add-in-Typus definiert. Letzteres fällt in die Zuständigkeit des Attributs *xsi:type*, das im Falle eines Aufgabenbereich-Add-ins den Wert „TaskPaneApp“ enthält. Bei einem Inhalts- oder Mail-Add-in würden die Attributwerte „ContentApp“ beziehungsweise „MailApp“ lauten.

Das *Id*-Element benennt die GUID der App, die man als eine Art Seriennummer bezeichnen könnte. Diese sollte für jede App neu generiert werden, was sich unter anderem mit dem *Online GUID Generator* (www.guidgenerator.com, [Link 42]) erledigen lässt. Im jeweiligen Verbreitungsraum der App (Office Store oder Netzwerk) muss die GUID auf jeden Fall eindeutig sein. Ist sie das nicht, taucht die App gar nicht erst im Auswahldialog der Office-Anwendung auf. Weniger kritisch ist der Inhalt des *Version*-Elements, das die Versionsnummer des Codebildes angibt.

Die XML-Elemente *DisplayName* und *Description* definieren den Namen der App sowie eine Beschreibung ihrer Funktion, der Name des Entwicklers lässt sich im Element *ProviderName* verewigen. *IconUrl* verweist auf den Speicherort der Icon-Datei. Dabei handelt es sich im konkreten Fall um die Projektdatei *SimpleApp.png*, die Sie von den Download-Dateien zum Buch in den Projektordner *SimpleApp* kopieren sollten.

**Hinweis**

Die Icon-Datei wird nur sichtbar, wenn Sie die hinter *DefaultValue* angegebene URL an Ihre konkreten Gegebenheiten anpassen. Dabei müssen Sie stets absolute Angaben machen; relative URLs wie in der Webseitendatei sind in der Manifestdatei nicht zulässig.

Im Abschnitt *Hosts* bestimmt das Manifest, mit welchen Office-Dokumenten die App zusammenarbeiten kann. Dabei steht „Document“ für ein Word-Dokument, „Workbook“ für eine Excel-Arbeitsmappe, „Project“ für ein Microsoft-Project-Projekt und „MailBox“ für ein Outlook-Postfach.

Das Element *Permissions* weist der App die Rechte für den Umgang mit den aufgeführten Office-Dokumenten zu. Der Wert *ReadWriteDocument* erlaubt hier Lese- und Schreibzugriffe und bildet das Maximum an zuweisbaren Rechten. Weitere Abstufungen wären *WriteDocument* (nur Schreiben), *ReadDocument* (nur Lesen) und *Restricted*, die dem Office-Add-in gar keinen Zugriff gewährt.

Das Element *SourceLocation* im Abschnitt *DefaultSettings* schließlich verweist auf den Speicherort der Add-in-eigenen Webseite. Auch hier müssen Sie die angegebene (absolute!) URL an Ihre Gegebenheiten anpassen.

Vertrauenswürdige Add-in-Kataloge

Die Office-Anwendungen laden lokal gespeicherte Office-Add-ins grundsätzlich nur aus einem „Vertrauenswürdigen Add-in-Katalog“, wobei es sich um den Ordner handelt, in dem die Manifestdateien gespeichert sind. Im Fall unseres Beispiel-Add-ins wäre das also der Ordner *OfficeApps*.

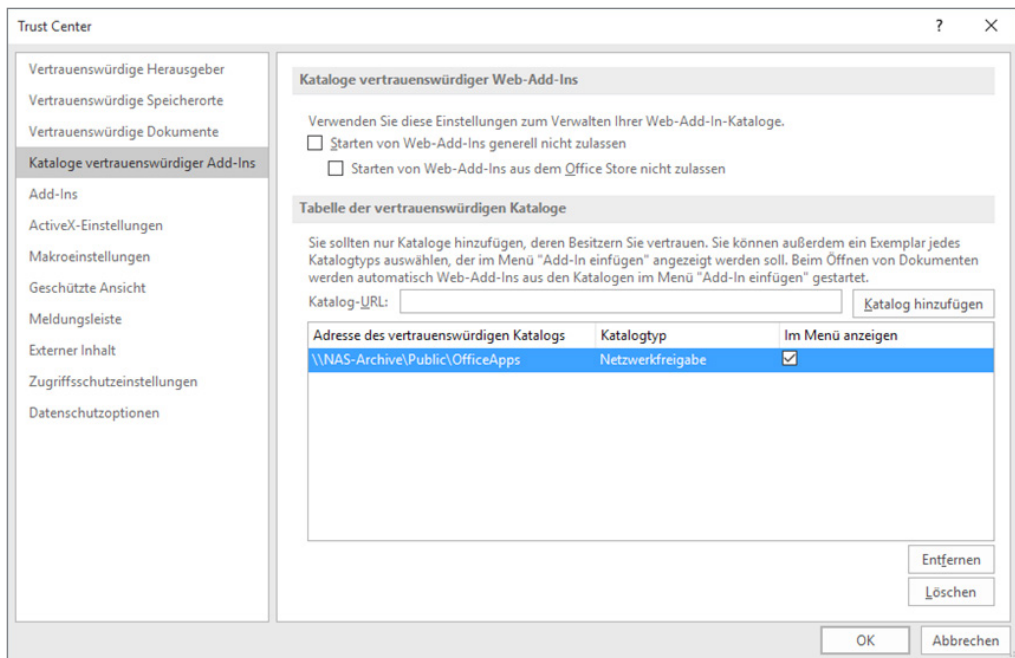


BILD 15.27: Lokal gespeicherte Apps stehen nur zur Auswahl, wenn man deren Manifestordner zum „Vertrauenswürdigen Katalog“ erklärt.

Um diesen Ordner in den Status der Vertrauenswürdigkeit zu erheben, starten Sie Excel 2016 und wählen DATEI | OPTIONEN. Klicken Sie links auf TRUST CENTER und danach auf die Schaltfläche EINSTELLUNGEN FÜR DAS TRUST CENTER. Markieren Sie links KATALOGE VERTRAUENSWÜRDIGER ADD-INS, und tragen Sie den UNC-Pfad des Ordners in das Textfeld „Katalog-URL“ ein – er sollte die Form „\\server\freigabe\manifestordner“ haben. Im Fall unseres Entwicklungsrechners lautet der Pfad:

\\NAS-Archive\Public\OfficeApps

Unnötig zu erwähnen, dass Sie auch diesen Pfad Ihren Gegebenheiten entsprechend ändern müssen. Wählen Sie dann **KATALOG HINZUFÜGEN**. Das Listenfeld enthält jetzt einen neuen Eintrag mit einem Kontrollkästchen dahinter („Im Menü anzeigen“). Das müssen Sie unbedingt einschalten, da Ihre Apps ansonsten nicht im Auswahldialog erscheinen. Anschließend schließen Sie alle Dialoge mit OK und starten Excel neu.

Add-in-Start

Das Beispiel-Add-in ist jetzt betriebsbereit und kann zum ersten Mal gestartet werden. Wählen Sie **EINFÜGEN | MEINE ADD-INS**, wobei Sie *nicht* auf das Icon, sondern auf dessen Pfeilsymbol klicken. Nach einem Klick auf **ALLE ANZEIGEN** und dann auf **FREIGELEGEBENE ORDNER** sollte das Dialogfeld einen Eintrag namens „SimpleApp“ enthalten, dessen Icon aus einem Smiley besteht.

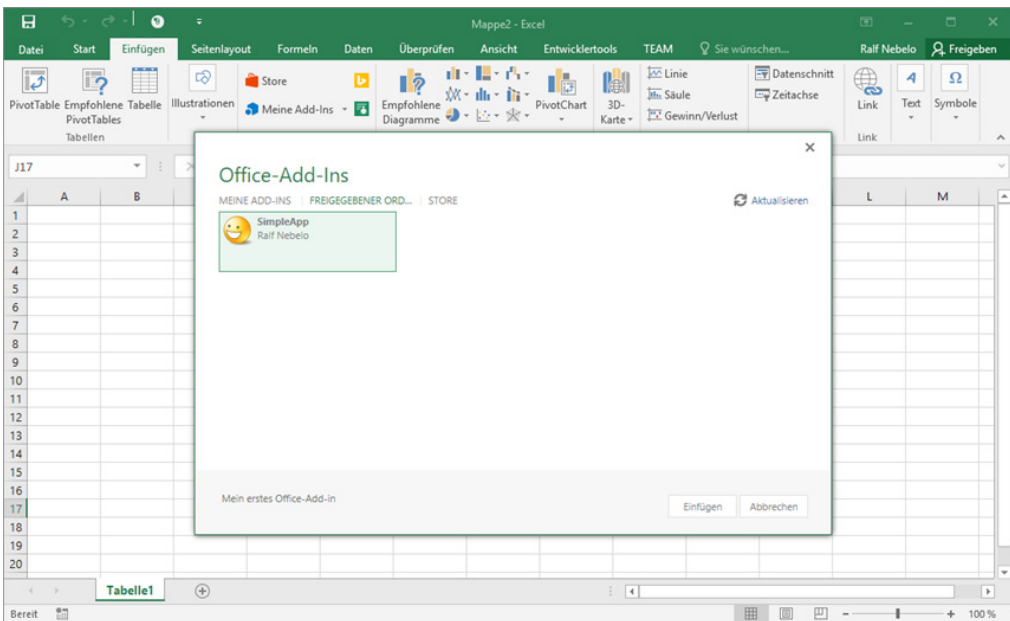


BILD 15.28: Wenn bis hierher alles geklappt hat, steht das Beispiel-Add-in nun im Auswahldialog bereit.

Fehlt der Eintrag, empfiehlt sich ein Klick auf die Schaltfläche **AKTUALISIEREN**, um die Anzeige auf den neuesten Stand zu bringen. Der Rest ist simpel: Markieren Sie „SimpleApp“, und wählen Sie **EINFÜGEN**.

Wenn alles geklappt hat, erscheint das Office-Add-in nun am rechten Rand des Excel-Fensters. Inhaltlich bietet es nicht viel: Sie können eine Webadresse eingeben und die entsprechende Seite dann mit einem Klick auf den Button in Ihrem Standard-Browser öffnen. SimpleApp halt!

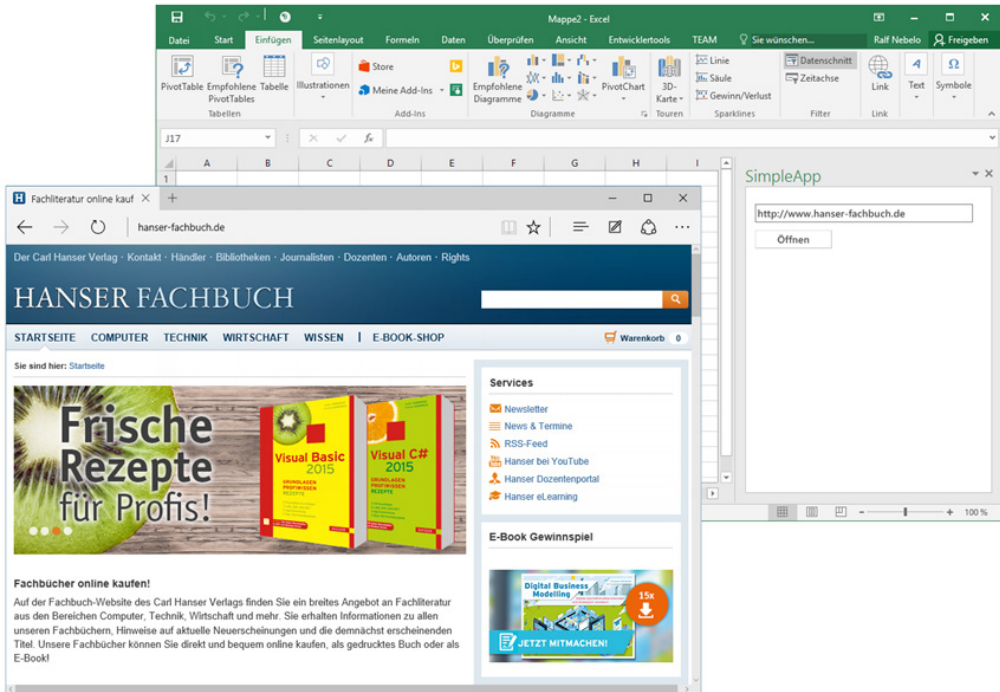
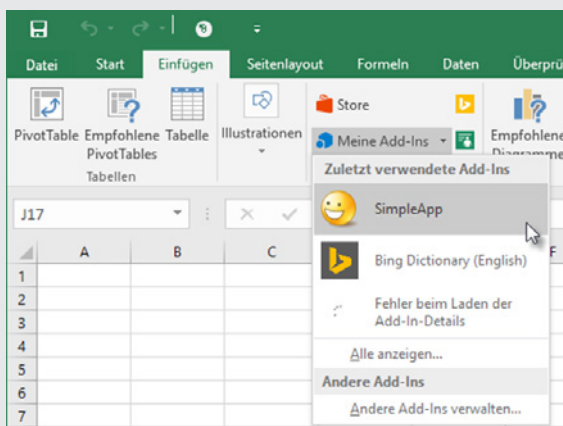


BILD 15.29: Das Beispielprojekt SimpleApp in Aktion



Hinweis

Wenn die App einmal erfolgreich gestartet wurde, erscheint sie künftig im Schnellstartmenü des MEINE ADD-INS-Symbols, das Sie mit einem Klick auf das Pfeilsymbol öffnen können.



15.9.5 Das JavaScript-API für Office

SimpleApp ist zwar schon ein „echtes“ Office-Add-in, hat mit Office-Programmierung aber noch nicht viel zu tun. Es ist absolut isoliert und kann anders als jedes VBA-Makro weder mit der Host-Anwendung (im konkreten Fall also Excel) kommunizieren noch Daten mit dem aktuellen Dokument austauschen.

Für diese Formen der Interaktion benötigt man das *JavaScript-API für Office*. Dabei handelt es sich um eine relativ umfangreiche Funktionsbibliothek, die mittlerweile in der Version 1.1 vorliegt und auf einem Microsoft-Server online verfügbar ist (natürlich kann man die JavaScript-Datei auch herunterladen und lokal abspeichern). Um diese Bibliothek nutzen zu können, muss man sie durch ein (weiteres) *script*-Element im HTML-Kopf der Add-in-Webseite anmelden:

```
<script src=
  "https://appsforoffice.microsoft.com/lib/1/hosted/Office.js"
  type="text/javascript">
</script>
```

API-Anatomie

Das JavaScript-API stellt dem Office-Add-in ein Objektmodell für den programmierten Zugriff auf die Host-Anwendung und den Datenaustausch mit dem Dokument zur Verfügung. Eine vollständige Dokumentation finden Sie hier [\[Link 43\]](#):

<http://msdn.microsoft.com/library/fp142185.aspx>

An der Spitze des Objektmodells steht das *Office*-Objekt. Es repräsentiert das API und verweist mit seiner *context*-Eigenschaft auf ein gleichnamiges Objekt, das den Laufzeitkontext des Office-Add-ins verkörpert. Dieses Objekt wiederum stellt die Verbindung zum *Document*-Objekt her, welches Zugriff auf das konkrete Dokument gewährt und bei Aufgabenbereich- oder Inhalts-Add-ins stets im Mittelpunkt des Interesses steht.

Bevor das Objektmodell nutzbar ist, muss das API initialisiert werden. Dazu setzt man die folgende Anweisung an den Anfang der Skriptdatei:

```
Office.initialize = function (reason) {
}
```

Die Zeile installiert zugleich einen Ereignis-Handler, der beim Laden des Office-Add-ins ausgeführt wird. Den Handler kann man mit Code füllen, um eigene Initialisierungen vorzunehmen oder auf unterschiedliche Startbedingungen zu reagieren. So könnte es etwa von Interesse sein, ob das Add-in erstmals eingefügt wurde oder bereits im Dokument vorhanden war. Zur Beantwortung der Frage sollte man den von Office übergebenen *reason*-Parameter auswerten, was folgendermaßen aussehen kann:

```
Office.initialize = function (reason) {
  if (reason == "inserted")
    showText("Add-in wurde neu eingefuegt");
  if (reason == "documentOpened")
    showText("Add-in wurde mit Dokument geoeffnet");
}
```

```
}  
function showText(text) {  
    document.getElementById("TextBox1").value = text;  
}
```

Auf Dokumentereignisse reagieren

Alternativ lässt sich die *Office.initialize*-Routine auch nutzen, um einen Handler für ein Dokumentereignis – die Änderung der Auswahl beispielsweise – einzurichten. Dazu benutzt man die *addHandlerAsync*-Methode des *Document*-Objekts und übergibt ihr eine *Office.EventType*-Konstante, die das gewünschte Ereignis definiert, sowie den Namen der Routine, die das Ereignis behandeln soll. Das Beispiel

```
Office.initialize = function (reason) {  
    Office.context.document.addHandlerAsync(  
        Office.EventType.DocumentSelectionChanged, myHandler);  
    }  
    function myHandler(eventArgs) {  
        showText(„Auswahl geändert“);  
    }  
}
```

erklärt die Routine *myHandler* zum offiziellen Event-Handler für das *DocumentSelectionChanged*-Ereignis. Das tritt in Excel auf, sobald der Anwender eine andere Zelle innerhalb des Arbeitsblatts aktiviert. In Word wirkt sowohl das Verschieben der Schreibmarke als auch das Markieren von Inhalten als Event-Auslöser.

Rückgabe von markiertem Text

Wer nicht nur auf eine Änderung der Auswahl reagieren, sondern auch auf deren Inhalt zugreifen will, verwendet die *getSelectedDataAsync*-Methode des *Document*-Objekts und übergibt ihr eine passende *Office.CoercionType*-Konstante.

Die bestimmt den sogenannten Koersionstyp, das ist der Datentyp, den die Methode zurückgeben soll. Dabei kann es sich im einfachsten Fall um den reinen Text der Auswahl handeln, der sich wie folgt ermitteln lässt:

```
Office.context.document.getSelectedDataAsync(  
    Office.CoercionType.Text, function(result) {  
        if (result.status == "succeeded")  
            showText(result.value);  
        else  
            showText(result.error.message);  
    }  
);
```

Die Methode greift auf den Inhalt der Auswahl zu und retourniert das Objekt *result* an die eingeschlossene *function*-Routine. Die überprüft zunächst die *status*-Eigenschaft des Objekts. Enthält diese den Text „succeeded“, war der Abruf des Auswahltextes erfolgreich. Eine leere *status*-Variable oder der Text „failed“ deuten auf einen Fehler hin, dessen Beschreibung dann in *result.error.message* steckt.

Rückgabe von mehreren markierten Werten

Neben simplem Text unterstützt das JavaScript-API noch weitere Rückgabetypen: *Matrix* und *Table* beispielsweise. Die eignen sich für die Rückgabe respektive das Festlegen von tabellarischen Daten, wobei eine Table Kopfzeilen enthalten kann, eine Matrix nicht.

Die drei bislang genannten Typen funktionieren in allen Add-in-tauglichen Office-Anwendungen. Table- oder Matrix-Daten stehen also nicht nur in Excel zur Verfügung, was man erwarten würde, sondern beispielsweise auch in Word, wo es ja ebenfalls Tabellen gibt.

Das JavaScript-API wandelt Daten, die nicht im passenden Typ vorliegen, in den meisten Fällen automatisch um. Sollte der Entwickler also einen Text anfordern, der Anwender aber einen Bereich mit mehreren Tabellenzellen markiert haben, so liefert das API dennoch einen Text zurück – in Form eines tabulatorseparierten Strings nämlich, der die Werte aller ausgewählten Zellen enthält.

Wer keinen Sammel-String, sondern jeden markierten Zellwert einzeln verarbeiten möchte, fordert den Typ Matrix beim Aufruf der *getSelectedDataAsync*-Methode ausdrücklich an:

```
Office.context.document.getSelectedDataAsync(
  Office.CoercionType.Matrix, function(result){
    if (result.status == "succeeded")
      showData(result.value);
  }
);
```

Die *value*-Eigenschaft des *result*-Objekts enthält dann im Erfolgsfall ein zweidimensionales Array mit den markierten Zellwerten. Die kann die aufgerufene *showData*-Funktion wie folgt mit zwei verschachtelten *for*-Schleifen zur Anzeige bringen:

```
function showData(data) {
  var text = "";
  for (var y = 0 ; y < data.length; y++) {
    for (var x = 0; x < data[y].length; x++) {
      text += data[y][x] + ", ";
    }
  }
  document.getElementById("TextBox1").value = text;
}
```

Markierte Dokumentinhalte ändern

Soll das Office-Add-in den Inhalt der Auswahl nicht nur lesen, sondern aktiv verändern, wählt man die *setSelectedDataAsync*-Methode und übergibt ihr im einfachsten Fall einen Text, der die Dokumentauswahl ersetzt:

```
Office.context.document.setSelectedDataAsync("Neuer Text");
```

Im Fall von Word ersetzt der Text den Inhalt der Markierung, im Fall von Excel wird stets der komplette Inhalt der aktiven Zelle ausgetauscht, auch wenn der Anwender nur einen Teil davon markiert hat.

Soll die App die Inhalte *mehrerer* Zellen ändern, die Teil der aktuellen Auswahl sind, definiert man zunächst ein passendes Array, das die neuen Werte enthält:

```
var arr = [['a', 'b'], ['c', 'd'], ['e', 'f']];
```

Dieses Array leitet man dann nebst einem weiteren Argument zur Festlegung des Datentyps Matrix an die *setSelectedDataAsync*-Methode weiter. Das Beispiel

```
Office.context.document.setSelectedDataAsync  
(arr, {coercionType: 'matrix'});
```

füllt einen zwei Spalten breiten und drei Zeilen hohen Tabellenbereich, dessen obere linke Ecke die aktive Zelle bildet, mit den Buchstaben „a“ bis „f“.

Das deutlich spektakulärere Einfügen von Bildern gelingt ausschließlich im Office-eigenen Textprogramm Word. In Excel ist es bislang nicht vorgesehen.

Entwicklerdefinierte Dokumentinhalte ändern

Über die vom Anwender festgelegte Auswahl hinaus können Office-Add-ins auch mit Regionen eines Dokuments interagieren, die der Entwickler bestimmt. Dazu muss die Region allerdings einen eindeutigen Namen besitzen, was im Fall von Excel auf benannte oder als Tabelle/Liste formatierte Arbeitsblattbereiche, in Word unter anderem auf Inhaltsstuelemente zutrifft.

Für den Zugriff auf eine solche Region generiert die App zunächst eine „Bindung“, über die sie dann den Inhalt der Region sowohl lesen als auch ändern kann. Dazu stehen ihr die Methoden *getDataAsync* und *setDataAsync* zur Verfügung, die ähnlich funktionieren wie die zuvor vorgestellten Methoden *getSelectedDataAsync* und *setSelectedDataAsync* für den Umgang mit Markierungsinhalten. Die verfügbaren Datentypen sind in beiden Fällen identisch.

Die zuvor schon erwähnte JavaScript-API-Dokumentation [\[Link 43\]](#) enthält ausführliche Infos über den Zugriff auf Regionen.

Auf die Dokumentdatei zugreifen

Neben der Interaktion mit markierten Dokumentinhalten und benannten Regionen erlaubt das JavaScript-API auch einen programmierten Zugriff auf die Dokumentdatei. Den erhält man mit Hilfe der *getFileAsync*-Methode. Damit kann man das Dokument beispielsweise versenden oder im Firmennetzwerk veröffentlichen. Zugang zum gesamten Dokumentinhalt liefert die Methode aber nicht.

Auch hier müssen wir Sie auf die JavaScript-API-Dokumentation [\[Link 43\]](#) verweisen. Eine ausführliche Behandlung des Themas würde den Rahmen einer Einführung sprengen.

15.9.6 Beispiel 2: ComplexApp

Ein zweites Beispielprojekt soll den Einsatz des JavaScript-API und den dadurch ermöglichten Datenaustausch zwischen Anwendung und App demonstrieren. Es übernimmt einen in Celsius angegebenen Temperaturwert aus der aktuellen Zelle und rechnet diesen wahlweise in Kelvin oder Fahrenheit um. Ein Mausklick fügt das Ergebnis dann wieder in die aktuelle Zelle ein.

Wenn Sie dieses Beispielprojekt praktisch nachvollziehen möchten, fügen Sie dem *OfficeApps*-Stammordner zunächst einen neuen Unterordner namens *ComplexApp* hinzu. Dort speichern Sie die folgende HTML-Datei, die die Webseite der App bildet, unter dem Namen *ComplexApp.html* ab:

```
<! 15\OfficeApps\ComplexApp\ComplexApp.html >
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=Edge"/>
    <link rel="stylesheet" type="text/css"
      href="../../OfficeStyles.css" />
    <script src=
      "https://appsforoffice.microsoft.com/lib/1/hosted
      /Office.js" type="text/javascript"></script>
    <script src="ComplexApp.js"></script>
  </head>
  <body>
    <div>
      Aktuelle Auswahl: <br>
      <input type="text" value="" id="txtValue"
        style="width: 254px" />
      <input type="button" value="Celsius in Kelvin"
        id="btnToKelvin" onclick="convertKelvin()"
        style="margin-top: 5px; width: 258px" />
      <input type="button" value="Celsius in Fahrenheit"
        id="btnToFahrenheit" onclick="convertFahrenheit()"
        style="margin-top: 5px; width: 258px" />
      <hr width="258" align="left">
      Ergebnis: <br>
      <input type="text" value="" id="txtResult"
        style="width: 254px" />
      <input type="button" value="Ergebnis einfügen"
        id="btnInsert" onclick="insertResult()"
        style="margin-top: 5px; width: 258px" />
    </div>
    <div>
      <hr width="258" align="left">
      Meldung: <br>
      <textarea id="txtMessage" rows="6"></textarea>
      <hr width="210" align="left">
    </div>
  </body>
</html>
```


Der *head*-Bereich der Webseite unterscheidet sich kaum von seinem Gegenstück in *SimpleApp.html*. Einziger Unterschied ist das zweite *script*-Element, das – wie zuvor schon beschrieben – den Verweis auf das JavaScript-API herstellt.

Innerhalb des *body*-Blocks sind die Unterschiede größer. Die Anweisungen dort fügen der Webseite gleich zwei Textboxen, drei Buttons und ein mehrzeiliges Textarea-Control aus dem HTML-Fundus hinzu. Die erste Textbox (id="txtValue") zeigt den Inhalt der aktuell markierten Zelle, die zweite (id="txtResult") das Ergebnis, das nach dem Klick auf die oberen beiden Buttons (id="btnToKelvin" und id="btnToFahrenheit") berechnet wird. Button Nummer Drei (id="btnInsert") überträgt das Ergebnis zurück in das Dokument. Das Textarea-Control schließlich (id="txtMessage") ist für die Anzeige möglicher Fehlermeldungen zuständig.

Die Skriptdatei von ComplexApp

Die Skriptdatei des Beispiel-Add-ins trägt den Namen *ComplexApp.js* und ist ebenfalls im Unterordner *ComplexApp* zu speichern.

```
/* 15\OfficeApps\ComplexApp\ComplexApp.js */
Office.initialize = function (reason) {
    Office.context.document.addHandlerAsync(
        Office.EventType.DocumentSelectionChanged, myHandler);
}

function myHandler(eventArgs) {
    showMessage("");
    Office.context.document.getSelectedDataAsync(
        Office.CoercionType.Text, function (result) {
            if (result.status == "succeeded")
                document.getElementById("txtValue").value = result.value;
            else
                showMessage(result.error.message);
        });
}

function convertKelvin() {
    var celsius =
        parseFloat(document.getElementById("txtValue").value)
    var kelvin = celsius + 273.15
    document.getElementById("txtResult").value = kelvin
}

function convertFahrenheit() {
    var celsius =
        parseFloat(document.getElementById("txtValue").value)
    var fahrenheit = (celsius * 1.8) + 32
    document.getElementById("txtResult").value = fahrenheit
}

function insertResult() {
    showMessage("");
    var correctedResult =
```

```

    document.getElementById("txtResult").value.replace(".", ",");
    Office.context.document.setSelectedDataAsync(correctedResult,
    function (result) {
        if (result.status == "failed")
            showMessage(result.error.message);
    });
}

function showMessage(message) {
    document.getElementById("txtMessage").value = message;
}

```

Die *Office.initialize*-Zeile, die ja beim Laden des Office-Add-ins automatisch aufgerufen wird, erklärt die Funktion *myHandler* zum zuständigen Event-Handler, der bei jeder Änderung der Dokumentauswahl ausgeführt wird. Die Funktion ermittelt dann den Wert der aktuellen Zelle mit Hilfe der API-Funktion *getSelectedDataAsync* und trägt diesen in die Textbox mit der ID *txtValue* ein.

Das Anklicken des obersten Buttons fällt in die Zuständigkeit der *convertKelvin*-Routine. Diese liest den Wert der Textbox *txtValue*, addiert die Zahl 273,15 dazu und trägt das Ergebnis in die Textbox *txtResult* ein.

Bei *convertFahrenheit* verhält es sich ähnlich. Die Routine tritt beim Anklicken des zweiten Buttons von oben in Aktion. Sie liest ebenfalls den Wert der Textbox *txtValue*, multipliziert diesen aber mit 1,8 und rechnet 32 hinzu, ehe sie das Ergebnis in die Textbox *txtResult* überträgt.

Ein Klick auf den dritten Button schließlich ruft die *insertResult*-Routine auf den Plan. Diese ermittelt den momentanen Wert der Textbox *txtResult*, tauscht einen eventuell darin befindlichen Dezimalpunkt gegen das hierzulande übliche Dezimalkomma aus und überträgt das Ergebnis mit Hilfe der *setSelectedDataAsync*-Methode des JavaScript-API in die markierte Zelle.

Die Manifestdatei von ComplexApp

Die Manifestdatei unseres zweiten Beispiel-Add-ins hat folgenden Inhalt und ist unter dem Namen *ComplexApp.xml* im Stamm-/Manifestordner *OfficeApps* zu speichern. Wie schon im ersten Beispiel, so sind auch hier die verwendeten (absoluten) URLs, die auf die Speicherorte der Webseiten- und der Icon-Datei verweisen, an die eigenen Gegebenheiten anzupassen.

```

<! 15\OfficeApps\ComplexApp.xml >
<?xml version="1.0" encoding="utf-8"?>
<OfficeApp
  xmlns="http://schemas.microsoft.com/office/appforoffice/1.1"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:type="TaskPaneApp">
  <Id>08afd7fe-1631-42f4-84f1-5ba51e242f12</Id>
  <Version>1.0</Version>
  <ProviderName>Ralf Nebelo</ProviderName>
  <DefaultLocale>EN-US</DefaultLocale>

```

```
<DisplayName DefaultValue="ComplexApp"/>
<Description DefaultValue="Mein zweites Office-Add-in"/>
<IconUrl DefaultValue=
  "\\NAS-Archive\Public\OfficeApps\ComplexApp\ComplexApp.png"/>
<Hosts>
  <Host Name="Document"/>
  <Host Name="Workbook"/>
</Hosts>
<DefaultSettings>
  <SourceLocation DefaultValue=
    "\\NAS-Archive\Public\OfficeApps\ComplexApp\ComplexApp.html"/>
</DefaultSettings>
<Permissions>ReadWriteDocument</Permissions>
</OfficeApp>
```

ComplexApp anzeigen und nutzen

Zum Starten des Beispiel-Add-ins wählen Sie **EINFÜGEN | MEINE ADD-INS**, wobei Sie wiederum *nicht* auf das Icon, sondern auf dessen Pfeilsymbol klicken. Nach einem Klick auf **ALLE ANZEIGEN** und dann auf **FREIGELEGEBENE ORDNER** sollte das Dialogfeld einen Eintrag namens „ComplexApp“ enthalten, dessen Icon aus einem Stern besteht. Fehlt der Eintrag, klicken Sie auf **AKTUALISIEREN**, um die Anzeige auf den neuesten Stand zu bringen. Anschließend markieren Sie „ComplexApp“ und wählen **EINFÜGEN**.

Das Office-Add-in sollte nun am rechten Rand des Excel-Fensters erscheinen. Testen Sie es, indem Sie ein paar Temperaturwerte in das Arbeitsblatt schreiben und diese anschließend der Reihe nach anklicken. Der jeweilige Zellwert erscheint dann in der Textbox **AKTUELLE AUSWAHL** der ComplexApp.

Nach einem Klick auf **CELSIUS IN KELVIN** beziehungsweise **CELSIUS IN FAHRENHEIT** sollte das Ergebnis der Umrechnung in der Textbox **ERGEBNIS** auftauchen, von wo aus Sie es mit einem Klick auf **ERGEBNIS EINFÜGEN** in die aktuelle Zelle übertragen können.

The screenshot displays the Microsoft Excel interface with a spreadsheet and a floating application window titled "ComplexApp".

Spreadsheet Data:

	A	B	C	D	E	F
1	Celsius	Kelvin	Fahrenheit			
2		32	305,15	89,6		
3		19	292,15	66,2		
4		27	300,15			
5						
6						
7						
8						
9						
10						
11						
12						
13						
14						
15						
16						
17						
18						
19						
20						
21						
22						

ComplexApp Interface:

- Aktuelle Auswahl:** 27
- Buttons:** Celsius in Kelvin, Celsius in Fahrenheit
- Ergebnis:** 80,6
- Buttons:** Ergebnis einfügen
- Meldung:** (Empty text area)

BILD 15.30: ComplexApp macht sich durchaus nützlich, indem sie Temperaturwerte von Celsius in Kelvin und Fahrenheit umrechnet.

Stichwortverzeichnis

Symbole

64-Bit-Programmierung 56, 758, 789

– ActiveX 789

– API-Funktionen deklarieren 792

– Bedingte Kompilierung 794

– Dynamic Link Libraries 790

– LongPtr-Datentyp 792

– PtrSafe-Schlüsselwort 792

– VBA7-Konstante 794

– Windows-API 790

2000 (Jahr) 225

\$ (String-Datentyp) 81

[BenutzerVerzeichnis] 300

*.cub 699

@ (Currency-Datentyp) 81

(Datum und Zeit) 227

*.dll (Dynamic Link Libraries) 756

(Double-Datentyp) 81

#Else 68

#End If 68

#If 68

% (Integer-Datentyp) 81

& (Long-Datentyp) 81

_ (mehrzeilige Anweisungen) 71

.NET-Framework 797

[OfficeVerzeichnis] 300

*.olb (Objektbibliotheken) 118

[RootVerzeichnis] 300

! (Single-Datentyp) 81

Styles 298

*.tlb (Objektbibliotheken) 118

*.udl 654

& (Verkettungsoperator) 219

*.xlam 49

*.xlam (Add-Ins) 737

*.xlb-Datei 400

*.xlsb (Konfigurationsdatei) 300

*.xlsm 49

*.xlsx 49

*.xltn 49

*.xltn-Dateien 304

*.xltx 49

*.xltx-Dateien 304

A

A 624

A1-Schreibweise 185

Abbrechen-Button 344

Abfragegenerator 622

Abfragen 663

– in MS-Query 642

– SQL 640

– *siehe auch* Verzweigungen 102

Abrunden 215

Abs 216

Absolute Adressen 177

AbsolutePosition 660

Accelerator 346

Access

– via ActiveX-Automation steuern 764

– Import von Excel-Tabellen 646

Access-Datenbanken, ConnectionString 653

Activate 130, 174

– Beispiel 415

– Blatt 202

– Diagramme 517

– OLE 775

– Pane 200

– UserForm 375

– Window 200

– Workbook 198

ActivateMicrosoftApp 787

ActivateNext 200

ActivatePrevious 200

ActivateTab-Methode 447

ActivateTabMso-Methode 447

ActiveCell 173

ActiveChart 201, 706

- ActiveChart-Objekt 519
- ActiveControl 364
- ActivePane 200
- ActivePrinter 297
- ActiveSheet 201
- ActiveWindow 199
- ActiveWorkbook 198
- ActiveX-Automation 564 f., 761
 - Access-Beispiel 764
 - ActiveX-Server programmieren 772
 - Excel als ActiveX-Server 768
 - Excel als OLE-Client 762
 - Excel steuern 768
 - neue Objekte erzeugen 772
 - Objektklasse 763
 - für OLE-Objekte 776
 - VB.NET 778
 - Verbindung herstellen 762
- ActiveX Data Objects 647
- ActiveX-Server, selbst programmieren 772
- ActiveX-Steuer-elemente 348
 - neu erstellen 349
- Add 146 f.
 - CalculatedFields 710
 - ChartObject 518
 - Name 177
 - OLE 777
 - Sheets 202
 - Window 200
 - Workbook 198
- AddComment 178
- AddControl 364
- Add-ins 452
 - Add-In-Manager 739
 - Dateigröße minimieren 101
 - Grundlagen 737
 - laden 739
 - selbst erstellen 737
 - Viren 166
- Add-ins-Register 52
- Add-Ins-Registerkarte 393
- Add-ins-Verzeichnis 300
- AddinInstall 740
- AddinUninstall 740
- AddItem 356
 - CommandBarComboBox 405
- Address 176
 - Beispiel 611
- Adressen
 - Konvertierung 177
- AddressLocal 176
- AddSmartArt-Methode 53
- ADO 647
- ADODB 648
- ADOMD 701
- ADOX 649
- AdvancedFilter 594
- AfterXmlExport 736
- Aktionsabfragen 666
- Aktive Arbeitsmappe 198
- Aktives Fenster 199
- Aktive Objekte 115
- Aktive Zelle 173
- Aktivierreihenfolge 347
- Aktuelles Datum 229
- Aktuelles Verzeichnis 246
- Aktuelle Zeit 229
- Alias 758
- AllowEditRange 291
- AllowXxx 290
- Altersberechnung 233, 582
- AltStartupPath 257
- And 103, 162
- Änderungen, rückgängig machen 72
- Anfügen (Symboleleisten) 400
- ANSI-Code 221
- ANSI-Format 217
- Anwendungsmenü erstellen 413
- Anzeigefeld 371
- API-Funktionen 756
- AppActivate 787
- AppendOnImport 731
- Application 198
 - Ereignisse 134, 139
 - Ereignisse, Beispiel 411
 - Ereignis bei Tasteneingabe 132
 - in Klassenmodulen 127
 - Optionen 297
- ApplyChartTemplate-Methode 519
- ApplyCustomType 518
- ApplyLayout 520
- ApplyLayout-Methode 53
- Apps für Office 52
- Arbeitsblätter *siehe* Worksheet 201
- Arbeitsblattgröße 48
- Arbeitsmappen
 - auswählen 199
 - eigenes Menü 609
 - erzeugen, Beispiel 524
 - freigeben 292
 - laden 204, 609
 - schützen 289
 - speichern 199
 - Syntaxzusammenfassung 208
 - Umgang mit 197
 - testen, ob schon geladen 609
 - *siehe* Workbook 198
- Areas 175

- Array 86
- AS (SQL) 665
- As Any 759
- As datentyp 81
- As New 121
- Asc 221
- ASMX 751
- Assistenten, selbst programmieren 387
- AtEndOfLine 251
- AtEndOfStream 251
- Aufgabenbereich-Add-in 813
- Aufgabenbereiche anlegen 55
- Auflistungen 106, 112, 146
- Auflistungsobjekt 112
 - selbst erzeugen 146
- Aufrufdialog 321
- Aufrunden 215
- Aufzeichnung von Makros 74
- Ausdruck 299
 - Beispiel Share.xlsm 501
 - Beispiel Speedy 493
 - Diagramme 511
 - Ereignis 130
 - Seitenvorschau 501
 - von Tabellen 483
- Ausrichtung (Winkel) 179
- Ausschnitte von Fenstern 200
- Auswahl *siehe* Selection 173
- Auto_Close 128
- Auto_Öffnen 128
- Auto_Open 128
 - Beispiel 207, 609
- Auto_Schließen 128
- Autofilter 587
- AutoFilter 594
- Autoformate 507
- AutoForm!Objekte (Shape) 570
- Automakros 128
- Automation, VB.NET 778
- Automatische Skalierung 42, 448
- Automatisch speichern 72
- AutoOutline 688
- Autoprozeduren 128
- AutoShapeType 571
- AutoSize 351
- AutoUpdate 776
- Average 286
- AVG (SQL) 665, 678
- Axis 515
- B**
- Backstage-Ansicht 44, 462
 - bottomItems-Bereich 467
 - dynamische Excel-Gruppen erweitern 471
 - Dynamische Gruppen anlegen 469
 - eigene Gruppen in Excel-Tabs einfügen 471
 - Excel-Tabs anpassen 470
 - FastCommand-Bereich 464
 - Gruppen anlegen 466
 - Primärschaltfläche anlegen 466
 - Programmierung 462
 - Schaltfläche einfügen 464
 - Spaltenbreiten einstellen 466
 - Tabs anlegen 465
 - topItems-Bereich 467
 - zweite Tab-Spalte nutzen 467
- Backup-Datei 297
- Bankleitzahl 808
- Baumkarte 563
- Bearbeitungsfeld 352
- Bedingte Formate 483
- Bedingte Formatierung 535
 - Datenbalken 46
 - Farbskalen 47
 - Symbolsätze 47
- Bedingte Kompilierung 68, 794
- Bedingte Zahlenformate 482
- Bedingungen 103
- Befehlsgruppen 42
- Befehlsleisten 401
- Befehlsregisterkarten 42
- BeforeClose 129
- BeforeDoubleClick 131
- BeforePrint 130
- BeforeSave 130
- BeforeXmlExport 736
- BeginGroup 404
- Beispieldateien XIX
 - Highlights 34
- Beispielprogramme XIX
- Benannte Parameter 96
- Benannte Zellbereiche 177, 491
- Benutzerdefinierte Diagrammtypen 306, 507, 518
- Benutzerdefinierte Dialoge 340
- Benutzerdefinierte Funktionen 26, 277
- Benutzeroberfläche 54, 393
 - schützen 289
- Benutzerverzeichnis 300
- berechnete Felder 710
- Bereich
 - Adresse ermitteln 176
 - auswählen 173
 - benannte Bereiche 177
 - komplexe Bereiche auswählen 187
 - zusammengesetzter Bereich 175, 189
 - in anderes Blatt kopieren 211
 - Format 179

- Geschwindigkeitsoptimierung 193
- Inhalt 178
- markieren 172
- in Mathematica-Liste umwandeln 268
- als Parameter in Funktionen 280
- Rahmen 192
- Schriftart 179
- Schriftart verändern 191
- Syntaxzusammenfassung 195
- Umrandung 192
- verknüpfen 212
- verschieben 210
- versetzen 174
- alle Zellen bearbeiten 189
- zusammensetzen 176
- Zwischenablage 210
- Bericht mit Access ausdrucken 764
- Bewegungsradius einschränken 288
- Bezeichnungsfeld 351
- Bibliothek 117
- BIC 808
- Bildfeld 371
- Bildlaufleiste 369
 - synchronisieren mit Textfeld 385
- Binärdateien 255
- Bing Maps 50
- BlackAndWhite 299
- Blatt
 - Blattgruppe 201
 - Blattgruppe bearbeiten 271
 - löschen 206
 - schützen 288, 380
 - Syntaxzusammenfassung 208
 - Umgang mit Blättern 201
- Blattliste
 - erstes Blatt auswählen 207
 - letztes Blatt auswählen 207
- Blattregister 365
- Blattreihenfolge schützen 289
- Blattwechsel 130
- Block ein-/ausrücken 71
- BOF (Recordset) 659
- Bookmark 660
- Boolean 81
- Border 192
- Borders 180
- BorderAround 180
- BottomMargin 299
- BottomRightCell 380, 572
- BoundColumn 358
- Boxplot 561
- Box-Whisker-Plot 561
- BrowseForFolder-Methode 565
- BuildPath 250

- Builtin 180
- Buttons 17, 362
 - in Tabellenblättern 382
- Byte 81
- ByVal 92

C

- Calculate 131, 308
- Calculation 307
- CalculationState 311
- CallBack-Routine, control-Argument 435
- CallBack-Routinen 424, 427
- CallByName 116
- CallOut 572
- Cancel 344, 350
- CancelUpdate 661
- Caption 200, 351
- Car-Sharing-Beispiel 496
- Car_template.xlt 606
- CCur 221
- CDate 230
- CDbl 221
- CD-ROM 832
- Ceiling 215
- Cell 173
- Cells 175
 - Beispiel 680
- CellFormat 184
- CenterFooter 299
- CenterHeader 299
- CenterHorizontally 299
- CenterVertically 299
- Change 131
 - Listenfeld 359
- ChangeFileAccess 297
- ChangePassword 291
- Characters 179
- Chart 513
 - Ereignisse 140
- Charts 201, 514
- ChartArea 514
- ChartFormat-Objekt 53
- ChartGroup 514
- ChartObject 513, 517
- ChartStyle 520
- ChartStyle-Eigenschaft 53
- ChartType 513
- ChartWizard 516
- ChDir 246
- ChDrive 246
- CheckBox 361
- Chr 222
- CInt 214, 221
- Class.cls 773

- Class_Initialize 145
- Class_Terminate 145
- Clear 178, 184, 518
 - Clipboard 772
 - CommandBarComboBox 405
- ClearComment 178
- ClearContents 178, 518
 - Beispiel 680
- ClearFormats 178, 518
- ClearNotes 178
- ClearOutline 688
- Click, Listenfeld 359
- Clipboard 772
- ClipboardFormats 211
- Clipboard.vbp 773
- CLng 215, 221
- Close 254
 - TextStream 251
 - Window 200
 - Workbook 198
- Code-Beispiele XIX
- Codeeingabe 69
- Codefenster, Block ein-/ausrücken 71
- CodeModule 136
- Collection 146
- Colors 297
- Column 176, 251
- Columns 176
- ColumnCount 357
- ColumnDifferences 176
- ColumnFields 709
- ColumnGrand 708
- ColumnHead 357
- ColumnRange 708
- ColumnWidth 180, 357
- COM-Add-Ins 741, 799
 - Viren 166
- COM-Architektur 51
- COM-Automation 761
- Command, ADO 662
- CommandBar 401
- CommandBarButton 404
- CommandBarControl 403
- CommandBarControls 403
- CommandBarPopup 405
- CommandBars 52, 401, 458
- CommandButton 362
 - in Tabellenblättern 382
- CommandText
 - PivotCache 713
 - QueryTable 643
- CommandType
 - PivotCache 713
- Comment 178
- Compiler
 - bedingte Kompilierung 68
- Connection 652
 - Beispiel 649
 - QueryTable 744
- ConnectionString 652
- ConnectionTimeout 652
- Const 84
- Control 350
- Controls 402f.
 - Frame 364
- ControlSource 350
 - ListBox 357
- ControlTipText 362
- ConvertFormula 177
 - Beispiel 614
- Copy 413
 - Blatt 202
 - Diagramme 517
 - File/Folder 249
 - Range 210
- CopyFace 404
- CopyFile 249
- CopyFolder 249
- CopyFromRecordset 662
- CopyPicture 521
- Count 147
- COUNT (SQL) 665
- CreateBackup 297
- CreateEmbed (Visual Basic) 769
- CreateEventProc 138
- CreateFolder 249
- CreateObject 763
 - Beispiel 766
 - VB.NET 780
- CreateTextFile 251
- CSng 221
- CStr 222
- CSV-Format 267
- CubeFields 709
- cubes 699
- CurDir 246
- Currency 81
- CurrentArray 187
- CurrentPage 709
- CurrentRegion 176, 187
 - Beispiel 358, 612
- CursorLocation 656
- CursorType 656
- CustomUI 54
- Custom UI Editor 424, 452, 462
- customUI.xlm 447
- customUI.xml 423, 425, 457ff., 462
- Cut 210

- CutCopyMode 210
- CVErr 329
 - Beispiel 282
- D**
- DAO 648
- Datar-Objekt 53
- DataBodyRange 708
- DataFields 709
- DataLabelRange 708
- DataLink-Datei 654
- DataObject 212
- DataRange 709
- Data Source 649
- data warehouse 698
- Date 81, 229
 - Fehler 228
 - internes Format 227
- DateAdd 230
- DateCreated 248
- DateDiff 230
- Datei drucken
 - Ereignis 130
- Datei speichern
 - Ereignis 130
- Dateien 242
 - Auswahldialog 335
 - automatisch laden 302
 - Dateiname auswählen 258
 - Eigenschaften 249
 - erzeugen/kopieren/löschen 249
 - lesen/speichern 253
 - Name auswählen 204
 - suchen 250
 - Syntaxzusammenfassung 273, 275
 - temporäre 247
 - Textformat 271
 - Unicode 251, 255
- Dateiauswahl 199
- Dateiauswahldialog 258, 335
- Dateiformate 49
- DateLastAccessed 248
- DateLastModified 248
- Datenbalkendiagramm 535
 - Anlegen 537
 - Ausgabebreite festlegen 537
 - Details festlegen 537
 - Farben wählen 537
 - Wertbezüge ändern 537
- Datenbanken 575
 - Anbindung für Mustervorlagen 485
 - Autofilter 587
 - Begriffsdefinition 576
 - Datensatzlisten 655
 - Einführungsbeispiel 15
 - erstellen 581
 - Export im dBase-Format 646
 - Filterkriterien 590
 - filtern 587
 - Funktionen 596
 - Glossar 577
 - Grundfunktionen 580
 - Grundlagen 576
 - gruppieren 685
 - Häufigkeitsverteilung 598
 - konsolidieren 530, 599
 - Mitarbeiterdatenbank 581
 - Recordsets 655
 - relationale 617
 - Relationen 618
 - sortieren 585
 - Spezialfilter 589
 - Suchen von Daten 587
 - Syntaxzusammenfassung 595
 - Teilergebnisse 685
 - Textdatei importieren 260
- Datenbankfunktionen 596
- Datenbankmaske 583
 - anzeigen 594
 - in Symbolleiste für den Schnellzugriff integrieren 583
- Datenfelder 86
 - in Tabellen übertragen 314
- Datenglättung 509
- Datenkanal 253
- Datenprotokollierung 521
- Datenpunkte 508
- Datenquelle 632
- Datenreihen 508
- Datensatzlisten 655
 - Navigation 659
 - verändern 661
- Datentypen 81
 - eigene 83
 - eigene, laden / speichern 257
- DateSerial 229
- DateValue 229
- Datum
 - 360-Tage-Jahr 229
 - Altersberechnung 233, 582
 - Beispiel 499, 582
 - durch Drehfeld einstellen 390
 - Konvertierung von/zu Zeichenketten 230
 - rechnen mit 232
 - Syntaxzusammenfassung 241
 - Umgang mit 227
- DATUM 231
 - Beispiel 22

- Datumsformat 183
- Datum und Zeit
 - Formate 225
 - Tabellenfunktionen 231
 - VBA-Funktionen 229
- DATWERT 231
- Day 229
- Days360 229
- DBANZAHL 598
- dBase-Format
 - Daten speichern 646
 - Excel-Tabellen speichern 646
- DBxxx-Funktionen 596
- DDE 788
- Deactivate 130
 - Beispiel 415, 612
 - UserForm 375
- Deaktivieren von Diagrammen 517
- Debug 69, 323
- Debugging 319
- Decimal 81
- Declare 757
- Default 344, 350
- Defaulteigenschaften 113
- DefaultFilePath 257
- Defaultobjekte 113
 - in Klassenmodulen 127
- DefaultWebOptions 747
- Defaultzeichensatz 297
- DefDatentypXxx 82
- Deklaration von DLL-Funktionen 757
- Delay 369
- Delete 178
 - Blatt 202
 - CommandBars 406
 - Diagramme 517
 - File/Folder 249
 - OLE 775
 - Smart Tags 749
 - Worksheet, Probleme 676
- DELETE 666
- DeleteFile 249
- DeleteFolder 249
- Dependents 176
- Destination 643
 - QueryTable 744
- Diagonale Linien 192
- Diagramme 505
 - mit ActiveX-Automation 768
 - Ausdruck 511, 521
 - ausrichten 518
 - auswählen und aktivieren 517
 - Beispiel 521
 - benutzerdefinierte Typen 507
 - Bestandteile 507
 - Datenglättung 509
 - Datenprotokollierung 521
 - deaktivieren 517
 - Diagrammassistent 506
 - einfügen 517
 - Einführung 505
 - Export 521
 - Fehlerindikatoren 511
 - Formatierungsmöglichkeiten 507
 - als GIF exportieren 521
 - kopieren 517
 - löschen 517
 - Optionen 509
 - Pivotdiagramme 703, 706
 - Programmierung 511
 - Syntaxzusammenfassung 534
 - in Tabellen einlagern 505
 - Trendlinien 509
 - Typen 506
 - Verbunddiagramme 506
- Diagramm-Engine 45
- Diagrammentwurf 506
- Diagrammtypen
 - Histogramm 558
 - Kastengrafik 561
 - Pareto 560
 - Sunburst 568
 - Treemap 563
 - Wasserfall 556
- Diagrammtypen2016.xlsm 557
- Diagrammvorlagen 306, 507, 519
- Dialog
 - Assistent 387
 - aufrufen 345
 - Beispiel 384
 - zur Dateiauswahl 335
 - Datenbankmaske 336, 583
 - dynamisch verändern 388
 - Einführungsbeispiel 342
 - Erscheinungsposition 339
 - exportieren 349
 - gegenseitig aufrufen 386
 - optische Gestaltung 347
 - Gültigkeitskontrolle 384
 - InputBox 337
 - Kaskade 386
 - Kette 387
 - mehrblättrige 365
 - Programmieretechniken 384
 - selbst definieren 339
 - Standarddialoge verwenden 333
 - Warnungen unterdrücken 336
- Dialogs 333

- Dialogeditor 346
 - DialogSheets 201
 - Dictionary 147
 - Diese Arbeitmappe 198
 - Digitales Zertifikat 169
 - Dim 80
 - Dim As New 121
 - DirectDependents 176
 - DirectoryMap 564
 - DirectoryMap.xlsm 564
 - DirectPrecedents 176
 - Direktbereich 74, 316
 - zur Fehlersuche 321
 - DisplayAlerts 206, 297, 310
 - Beispiel 501
 - DisplayAutomaticPageBreaks 298
 - DisplayDrawingObjects 297
 - DisplayFormularBar 297
 - DisplayFormulas 298
 - DisplayGridlines 200, 298
 - DisplayHeadings 200, 298
 - DisplayHorizontalScrollBar 298
 - DisplayNotelIndicators 297
 - DisplayOutline 298, 688
 - DisplaySmartTags 749
 - DisplayStatusBar 309
 - DisplayUnit 515
 - DisplayUnitCustom 515
 - DisplayUnitLabel 515
 - DisplayVerticalScrollBar 298
 - DisplayWorkbookTabs 298
 - DisplayZeros 298
 - DivID 746
 - DLL
 - anwendungsspezifische DLL 756
 - Definition 756
 - Einführung 756
 - Funktionen deklarieren 757
 - Parameterliste 758
 - Do 107
 - DoCmd (Access) 764
 - DocumentSelectionChanged-Ereignis 823
 - DoEvents 311
 - Dokument schützen 289
 - Dokumentenvorschau 480
 - Dokumentgröße 48
 - Doppelklick 131
 - Double 81
 - Download-Dateien zum Buch
 - Beispieldateien XIX
 - Download XIX
 - Inhalte 831
 - Internetadressen XIX
 - Drehfeld 369
 - Beispiele 390
 - Drehwinkel 179
 - Drill-Down 697
 - Drive 245, 248
 - Drives 245
 - DriveExists 250
 - DriveType 245
 - DropDownLines 410
 - Dropdown-Liste 355
 - DropDownWidth 410
 - Druckbereich 299
 - Drucken
 - Ereignis 130
 - von Tabellen 483
 - Druckereinstellung 299
 - Druckoptionen 295
 - DTD-Datei 724
 - Duplicate 517
 - Dynamic Data Exchange 788
 - Dynamic Link Libraries *siehe* DLL 756
 - dynamische Felder 85
- E**
- Each 106
 - Edit (DAO) 661
 - Editor 69
 - Eigene Datentypen 83
 - Eigene Tabellenfunktionen definieren 277
 - Eigenschaften 6, 112
 - Defaulteigenschaften 113
 - Projekt 68
 - Unterscheidung zu Methoden 113
 - Eigenschaftsprozeduren 143
 - Eingabeabsicherung 24
 - Eingabeerleichterung 13
 - Einzelschrittmodus 323
 - Elseif 102
 - E-Mail 742
 - EmbedSmartTags 749
 - Empty 82
 - EnableAutoFilter 298
 - EnableAutoRecover 297
 - EnableCancelKey 310, 330
 - EnableEvents 129
 - EnableOutlining 298
 - EnablePivotTable 298
 - Enabled, Beispiel 368
 - End 187
 - EntireColumn 176
 - EntireRow 176
 - Entwicklungsumgebung 61
 - Optionen 66
 - EnvelopeVisible 743

Environ 247, 760, 768
EOF 254
- Beispiel 649
EOF (Recordset) 659
Eqv 162
Erase 86
Ereignisse 7, 124
- externe Objekte 134
- Syntaxzusammenfassung 138
Ereignisprozeduren 125
- Beispiel 411
- per Programmcode erzeugen 136
- Steuerelemente 344
Err 329
Error 329
ErrorBars 516
Ersetzen (Replace-Methode) 184
Erweiterter Editor 626, 628
Euro 219
Evaluate 172
Even 215
Event 144
Excel
- individuell konfigurieren 295
- Konfigurationsdateien 300
Excel4IntlMacroSheets 201
Excel4MacroSheets 201
Excel9.olb 118
Excel 64 Bit 56
- Kompatibilitätsprobleme 789
Excel-2016-Diagramme 556
Excel-Add-in 452
Excel-Befehle auflisten 450
Excel-Bibliothek 117
Excel-Version 315
Excel.xlb 400
Execute 749
ExecuteExcel4Macro 314
Exit Do 107
Exit For 106
Exit Function 90
Exit Sub 90
Export 521
- Textformat 267
ExportAsFixedFormat 184, 202
Export nach Access 646

F

Fakultät 97
False 103
Farbpalette 297, 573
Farbskalendiagramm 535
- Anlegen 538
- Farbbereiche definieren 538

Fehler
- Absicherung 325
- Behandlungsroutine 328
- Suche 319
- Unterbrechung 74, 310, 330
Fehlerindikatoren in Diagrammen 511
Feiertage 235
Felder 84
- Anzahl der Dimensionen ermitteln 92
- Datenfelder 86
- dimensionieren 84
- dynamische 85
- Indexbereich ermitteln 86
- laden / speichern 257
- löschen 86
- an Prozeduren übergeben 92
- in Tabellen übertragen 313
- umdimensionieren 85
Fenster
- alle schließen 126
- Anordnung schützen 289
- Ausschnitte 200
- ein- und ausblenden 200
- Ereignisse 131
- alle Fenster in Icons verwandeln 205
- Gitterlinien 200
- Größe 200
- Kopfzeilen 200
- Optionen 296
- Position 200
- Syntaxzusammenfassung 208
- teilen 205
- Titel 200
- Umgang mit 199
- Windows 199
- Zustand 200
File 249
FileCopy 275
FileDateTime 275
FileDialog 259
FileExists 250
FileLen 275
Files 248f.
FileSearch-Objekt 53, 153, 250
FileSystemObject 243
File System Objects 243, 565
- GetFolder-Methode 566
Fill 572
Filter 218, 594
- in Datenbanken 587
Filters 594
Filterkriterien 590
- in MS Query 640

- FilterMode 298
 - Find 184, 592, 660
 - FindControl
 - Beispiel 411
 - FindFormat 184
 - FindNext 593
 - FindPrevious 593
 - FindWindow 791
 - FirstPageNumber 299
 - Fix 215
 - Floor 215
 - Fluent 418
 - Fluent-Oberfläche 42
 - Fm20.dll 118
 - Fokusprobleme 382
 - Folder 247
 - FolderExists 250
 - Font 179
 - FooterMargin 299
 - For 105
 - ForAppending 251
 - ForEach 106
 - ForeColor 573
 - Format 222
 - FormatCondition 483
 - FormatCurrency 222
 - FormatDateTime 222, 231
 - Formatierung
 - bedingte 483
 - Formatierung suchen und ersetzen 184
 - FormatNumber 222
 - FormatPercent 222
 - Formatvorlage 7, 180
 - Formeln eintragen 530
 - Formelfeld 372
 - Formelsprache M 626
 - Table.SelectRows 627
 - Formula 179, 185
 - PivotField 710
 - FormulaHidden 288
 - FormulaR1C1 179, 185, 531
 - FormulaR1C1Local 185
 - Formulare
 - ausdrucken 483
 - Beispiel Speedy 488
 - intelligente 477
 - *siehe* Dialoge 339
 - ForReading 251
 - ForWriting 251
 - Fragebogen 668
 - Frame 363
 - FreeFile 254
 - FreeSpace 245
 - FreezePanels 200, 298
 - Freigegebene Arbeitsmappen 292
 - FROM (SQL) 663
 - FSO 243
 - FullName
 - Workbook 199
 - Function 89
 - PivotField 709
 - Funktionen
 - aufrufen (Run-Methode) 96
 - Kategorien 278
 - Rückgabewert 91
 - selbst definieren 26
 - *siehe auch* Prozeduren 89
 - Tabellenfunktionen 214
 - Fußzeile 299
- ## G
- Ganze Zahlen 83
 - Gauß-Algorithmus 235
 - Geburtsmonat 582
 - Geschwindigkeitsoptimierung 193, 307
 - Gestaltung eigenständiger Programme 412
 - Get 256
 - GetAbsolutePath 250
 - GetAttr 275
 - GetBaseName 250
 - GetDriveName 250
 - GetFileName 250
 - GetFolder 247
 - GetFormat 772
 - GetObject 763
 - Beispiel 764
 - VB.NET 780
 - GetOpenFileName 199, 204, 258
 - GetParentFolderName 250
 - GetSaveAsFileName 199, 204, 258
 - GetTempName 247
 - GetText 772
 - GetWindowsDirectory 760
 - GIF (Diagramm-Export) 521
 - Gitternetzlinien 508
 - Gliederung 710
 - GoalSeek 194
 - GoTo 175, 327
 - Grenzen von VBA 55, 796
 - GridlineColor 298
 - GridlineColorIndex 298
 - Gridlines 515
 - Group 710
 - GROUP BY (SQL) 663, 679
 - GroupItems 572
 - Grundsätzlich Neuberechnen 283
 - Gruppenfeld 363

Gruppierung
– Listen 685
– Pivottabellen 710
GUID 818
Gültigkeitskontrolle 480
Gültigkeitsregeln 24

H

Haltepunkte 324
HasDisplayUnitLabel 515
HasFormula 179
Häufigkeitsverteilung 598
HeaderMargin 299
Height 380
– Dialog 388
– PlotArea 515
– Window 200
Hello-World-Beispielprogramm 69
Herkömmliche Makrokommandos 314
Hidden 287
HiddenFields 709
Hide 344
Hierarchie von Objekten 512
Hintergrundberechnungen 311
Histogramm 41, 48, 53, 556
– Definition 558
– Programmierung 559
Hochformat 299
HorizontalAlignment 179
Hour 229
HTML 751
HTML-Datei 816
HTML-Export 745
HTML-Import 744
HTML-Optionen 295
HtmlType 746
HTTP 751
Hyperlinks 831
Hyperlinks.doc XIX

I

Icon 812
Icons Gallery 428
ID-Nummer 787
Id-Nummern (CommandBar) 407
If 102
Image 371
Imp 162
Implements 145, 149
Import
– HTML-Datei 744
– Textdateien 260
– Web-Import 744
– XML-Daten 734

ImportOption 731
IndentLevel 180
Index (Datenbanken) 577
Individuelle Konfiguration 295
Inhalts-Add-in 813
Inhaltsverzeichnis generieren 565
Initialize 145, 375
INNER JOIN (SQL) 663
Input 255
InputBox 337
– Beispiel 614
Insert 178
INSERT INTO 666
InsertLines 138
InsideHeight 364, 515
InsideWidth 364, 515
Instancing 145
InStr 217
– Beispiel 613
InstrRev 218
Int 215
Integer 81
– Probleme 83
Integrated Security 653
Intelligente Formulare 477
– Einführungsbeispiel 21
– Grenzen 503
IntelliSense 70, 801
Interactive 310
Interaktive HTML-Seiten 745
Internet 742
Internetadressen XIX
Intersect 176
– Beispiel 358
Invalidate-Methode 445
Is 123, 161
– Probleme 716
IsAddin 738
IsCalculated 710
IsDate 82
IsEmpty 82
– Beispiel 500
IsError 82
IsMissing 94
IsNull 82
IsNumeric 82, 222
– Beispiel 286
IsObject 82
IsRootFolder 248
Item 113

J

JAHR 231
 Jahr 2000 225
 Jahreskalender erzeugen 238
 JavaScript 812
 JavaScript-API für Office 812, 822
 – addHandlerAsync 823
 – Bindung 825
 – CoercionType 823
 – context-Eigenschaft 822
 – Document-Objekt 822
 – Dokumentereignisse 823
 – Ereignis-Handler 822
 – getDataAsync 825
 – getFileAsync 825
 – getSelectedDataAsync 823
 – Initialisierung 822
 – Koersionstyp 823
 – Matrix 824
 – Objektmodell 822
 – Office.initialize 823
 – Office-Objekt 822
 – reason-Parameter 822
 – Regionen 825
 – setDataAsync 825
 – setSelectedDataAsync 824
 – Table 824
 JETZT 231
 Join 218

K

Kalender erzeugen 238
 Kanalnummer ermitteln 254
 Kaskade von Dialogen 386
 Kastengrafik 41, 48, 53, 556
 Kastengrafik-Diagramm
 – Definition 561
 – Programmierung 562
 Katalog-Control 43
 Kette von Dialogen 387
 KeyDown 354
 KeyPress 354
 KeyUp 354
 Kill 275
 Klassen 160
 Klassenmodul 773
 – Beispiel 135
 – Defaultobjekt 127
 KlassikMenü.xlam 452
 Klassische Menüs 449
 Kombinationslistenfeld 355
 Kommentare 16, 71, 178
 Kompilierung, bedingte 68

Konfiguration 295
 – Konfigurationsdateien 300
 – Optionen 295
 Konsolidierung von Tabellen 530, 599
 Konstanten 84
 Kontextmenüs 458
 – Ändern 408
 – Anlegen 409
 – Befehle ausblenden 460
 – Befehle einfügen 460
 – Controls einfügen 460
 – programmiert anpassen 459
 – Untermenüs anlegen 461
 Kontrollausgaben 323
 Kontrollkästchen 361
 Konvertierung
 – Adressen, absolut/relativ 177
 Koordinatenachsen 508
 Kopfzeile 299
 Kreisfläche 26
 Kreuztabellen *siehe* Pivottabellen 689
 Kurzschreibweise
 – für benannte Bereiche 177
 – Evaluate 172
 – Item 113
 – für Zellbezüge 172

L

Label
 – Bezeichnungsfeld 351
 – Sprungmarke 327
 LabelRange 709
 LargeChange 369
 late binding 780
 Laufwerke 245
 Laufzeitfehler 325
 LBound 86, 92
 LCase 217
 Lebensdauer von Variablen 99
 Left 217, 380
 – Range 339
 – Window 200
 LeftFooter 299
 LeftHeader 299
 LeftMargin 299
 Legende 509
 Len 217
 Lib 757
 LibraryPath 257
 Like 103, 161, 220
 – Beispiel 391
 Line 573
 – Shape 572
 – TextStream 251

- Line Input 255
 - LinkedCell 604
 - LinkedControl 382
 - List 356
 - ListBox
 - in Tabellenblättern 382
 - ListColumn 722
 - ListCount 356
 - Listen 721
 - Listenfeld 355, 604
 - in Symbolleisten 404
 - in Tabelle, Beispiel 499, 611
 - in Tabellenblättern 382, 672
 - ListFillRange 382
 - Beispiel 611, 672
 - ListHeaderRows 592
 - ListIndex 356 f.
 - ListObject 722
 - ListObject-Objekt 629
 - ListRow 722
 - ListStyle 356, 383
 - Literaturdatenbank 15
 - Livevorschau 43
 - LoadPicture 371
 - Loc 254
 - Location 706
 - Locked 288
 - LockType 656
 - LOF 254
 - Logische Bedingungen 103
 - Long 81
 - Probleme 83
 - Loop 107
- M**
- MacroOptions 279
 - Mail-Add-in 813
 - Mailer 742
 - MailSystem 742
 - Makro 3
 - absolute/relative Aufzeichnung 76
 - aufrufen 73
 - aufzeichnen 9, 74
 - ausführen 10
 - Funktionskategorie 278
 - herkömmliche Makrosprache 314
 - Optionen 73, 278
 - relativ aufzeichnen 13
 - Makroarbeitsmappe 75
 - Makroaufzeichnung
 - Pivottabellen 707
 - Makrosicherheit 797
 - Makro-Verzeichnis 300
 - Makrovorlagen 314
 - Manifestdatei 812
 - MAPI 742
 - Mathematica 268
 - Matrixfunktionen 282
 - Matrizen als Parameter 94
 - Mausereignisse 131
 - Mausklick, Ereignis 131
 - MaxMinVal 194
 - Max, ScrollBar 369
 - Me 116, 145
 - Mehrblättrige Dialoge 365
 - Mehrdimensionale Daten(banken) 698
 - Mehrfarbige Zahlenformate 482
 - Mehrzeilige Anweisungen 71
 - Meldung anzeigen 337
 - MemoryUsed 712
 - Menü 393
 - Verwaltung, Beispiel 609
 - Menüband 42, 418
 - Änderungen revidieren 423
 - Anpassen 54
 - Anpassungen permanent bereitstellen 452
 - ausblenden 425
 - Automatische Skalierung 448
 - Befehle hinzufügen 422
 - Befehlsgruppen anlegen 421, 426
 - Befehlsgruppen ausblenden 421
 - Befehlsgruppen verschieben 421
 - Beschriftung 430
 - Callback-Routinen 424, 427
 - Custom UI Editor 424
 - Design-Code 54
 - Dialogfeldstarter anlegen 441
 - dynamische Menüs anlegen 436
 - Funktions-Code 54
 - geteilte Schaltflächen anlegen 438
 - Icons 428
 - Kataloge anlegen 442
 - Kataloge dynamisch füllen 443
 - Kombinationsfelder anlegen 439
 - Konfiguration sichern 422
 - Kontrollkästchen anlegen 433
 - Listenfelder anlegen 439 f.
 - Makros integrieren 422
 - manuell anpassen 419
 - Menü nachbilden 449
 - Menüs anlegen 434, 436
 - Menütrennlinien einfügen 439
 - Programmierung 423
 - Registerkarten aktivieren 447
 - Registerkarten anlegen 420, 426
 - Registerkarten ausblenden 420, 426
 - Registerkarten verschieben 421
 - RibbonX-Controls 430

- Schaltflächen anlegen 426, 430
- statische Menüs anlegen 434
- Symbole 428
- Symbolleisten nachbilden 451
- Textfelder anlegen 433
- Wechselschaltflächen anlegen 432
- zurücksetzen 423, 445
- Menüleisten *siehe* Symbolleisten 395
- Menü- und Symbolleisten 44, 52, 393
- Änderungen speichern 400
- Anwendungen integrieren 412
- Ausblenden 406, 416
- bearbeiten 395
- Befehle gruppieren 399
- Beschriftung festlegen 398
- erstellen 397
- Makro zuweisen 399
- nachbilden 449
- Programmiert erstellen 394
- Programmierung 401
- Symbol festlegen 397
- meta-Element 816
- Methoden 6, 112
- Unterscheidung zu Eigenschaften 113
- Mid 217
- Beispiel 613
- Min
- ScrollBar 369
- Minisymbolleiste 44
- Minute 229
- MINUTE 231
- Mitarbeiterdatenbank 581
- MkDir 275
- Mod 161, 215
- Mode
- Connection 652
- Modf 215
- Modulblätter 69
- Moduleigenschaft 144
- MONAT 231
- Monat mit Drehfeld einstellen 390
- Monatsprotokoll 529
- Month 229
- MonthName 231
- Move
- File/Folder 249
- MoveAfterReturn 297
- MoveAfterReturnDirection 297
- MoveFile 249
- MoveFirst 659
- MoveFolder 249
- MoveLast 659
- MoveNext 659
- Beispiel 649

- MovePrevious 659
- Msado15.dll 118
- MSDASQL 654
- MSDE 653
- MS-Forms 57
- Bibliothek 340
- Steuerelemente 350
- MsgBox 337
- Mso9.dll 118
- MS Query 622, 632
- Abfragen 642
- Datenquelle definieren 632
- dynamischer Pfad 644
- Filterkriterien 640
- OLAP 699
- Rechenfunktionen 638
- Relationen 638
- Sortierkriterien 640
- SQL 640
- Multidimensionale Daten(banken) 698
- Multifunktionsleiste *siehe* Menüband 42, 418
- MultiPage 365
- MultiRow 366
- MultiSelect 357
- Multitasking 311
- Mustervergleich 220
- Mustervorlagen 304, 477, 479
- mit Datenbankanbindung 485
- Grenzen 503

N

- Nachkommaanteil ermitteln 215
- Name 199
- für Datenbankbereich 594
- Folder 248
- MS-Query-Parameter 643
- Objekt 177
- VBE versus Excel 136
- Virengefahr 168
- von Zellbereichen 491
- Name As 275
- Namensraum (XML) 725
- Names 177
- Zuordnungsprobleme 177
- Navigator 624
- Netzwerkfestplatte 815
- Netzwerkfreigabe 815
- Neuberechnung 131, 308
- bei eigenen Tabellenfunktionen 283
- grundsätzlich Neuberechnen 283
- Neue Diagrammtypen 48, 53, 556
- New 121
- NewWindow 200
- NewWorkbook 135

- Next 105, 327
- Nodes 572
- Not 103
- Notepad++ 815
- NoteText 178
- Nothing 122
 - Beispiel 765
- Notizen 16, 178
- Now 229
- Null 82
 - Vergleich mit 82
- NullString 760
- NumberFormat 180
- NumberFormatLocal 180
- NWind.mdb 624, 626

- O**
- Oberflächengestaltung 412
- Object 381, 776
 - OLE 764
- Object Linking and Embedding
 - *siehe* OLE 774
- Objekte 6
 - Bibliothek 117
 - Defaultobjekte 113
 - Katalog 117
 - Objektvariablen 121
 - Typ 123
 - Umgang mit Objekten 111
 - Unterscheidung zu Methoden und Eigenschaften 114
 - Vergleich mit Is 123
 - With 120
 - Zugriff auf aktive Objekte 115
- Objekthierarchie, Diagramme 512
- Objektkatalog 117
- Objektklasse 763
- Objektreferenz 831
- Objekttyp 121
 - mit TypeName ermitteln 123
- Objektvariablen 121
 - löschen 122
- ODBC
 - ADO 654
- Odd 215
- ODER 482
- Öffentliche Variablen 98
- Office-2003-Menüs nachbilden 44
- Office-Add-ins 41, 49, 52, 56, 811
 - Aufgabenbereich-Add-in 813
 - Bedienoberfläche 812
 - Beispiel ComplexApp 826
 - Beispiel SimpleApp 815
 - Capabilities 818
 - Description 818
 - DisplayName 818
 - GUID 818
 - Icon 812
 - IconUrl 818
 - Id-Element 818
 - Inhalts-Add-in 813
 - JavaScript-API 812, 822
 - link-Element 816
 - Mail-Add-in 813
 - Manifest 812, 819
 - Manifestdatei 817
 - meta-Element 816
 - Napa Development Tools 815
 - Netzwerkfreigabe 815
 - Office.initialize 828
 - Office Store 811, 818
 - Permissions 819
 - ProviderName 818
 - ReadDocument 819
 - ReadWriteDocument 819
 - Restricted 819
 - script-Element 816
 - Sicherheit 813
 - Skriptdatei 812, 817
 - SourceLocation 819
 - Stylesheet 812
 - Typen 813
 - veröffentlichen 818
 - Version 818
 - Vertrauenswürdiger Katalog 819
 - Webseite 812
 - Werkzeuge 815
 - WriteDocument 819
- xsi
 - type 818
- Zugriffsrechte 819
- Office-Apps 49, 811
- Office-Icons 428
- Office-Menü 44
- Office Store 811
- OfficeStyles.css 812, 816
- Offset 174, 176
- OK-Button 344
- OLAP 698
- OLAP-Cubes 699
- OLE 774
 - ActiveX-Automation 776
 - neue Objekte einfügen 777
 - Objekte bearbeiten 775
- OLE-Feld (Visual Basic) 769
- OLEFormat 381
- OLEObject 775
- OLEObjects 775

- OLEType 381, 775
 - OnAction 131
 - CommandBarButton 404
 - onclick-Attribut 812
 - OnError
 - Beispiel 137
 - OnError GoTo 326
 - OnError Resume Next 326
 - OnEventXxx 127
 - OnKey 132
 - Beispiel 207
 - Online-Hilfe 316
 - OnRepeat 132
 - OnTime 133
 - OnUndo 132
 - Open 199, 253
 - Connection 649
 - Ereignis 129
 - Recordset 649
 - OpenAsTextStream 251
 - OpenCurrentDatabase 764
 - OpenReport (Access) 765
 - OpenText 263
 - OpenTextFile 251
 - OpenXml 730
 - Open-XML-Format 49
 - Operator Is 56
 - Operatoren 161
 - Optimale Konfiguration 295
 - Optimierungen 56
 - Optionen 295
 - Diagramme 509
 - Drucken 295
 - Entwicklungsumgebung 66
 - Fenster 296
 - Makros 73
 - Speichern 295
 - Option Base 85
 - OptionButton 361
 - Option Compare Text 162, 219
 - Option Explicit 80, 320
 - Option Private Module 99, 101
 - Optional 94
 - Optionale Parameter 94
 - Optionsfelder 361
 - Or 103, 162
 - ORDER BY (SQL) 663
 - Organigramme 548
 - Orientation 179, 299, 709
 - Fehler bei Formelfeldern 710
 - OriginalValue 661
 - Ostern 235
 - OutlineLevel 688
- P**
- Page 365
 - Pages 366
 - PageFields 709
 - PageRange 708
 - PageSetup 298
 - Panes 200
 - PaperSize 299
 - Papiergröße 299
 - ParamArray 95
 - Beispiel 280, 286
 - Parameter
 - benannte 96
 - DLL 758
 - Prozeduren 91
 - ParentFolder 248
 - Pareto 41, 48, 53, 556
 - Pareto-Diagramm
 - Definition 560
 - Programmierung 560
 - Passwortschutz für Zellbereich 291
 - Paste
 - Diagramme 517
 - Excel 770
 - OLE 778
 - Worksheet 210
 - PasteFace 404
 - PasteSpecial 210
 - Beispiel 494, 532, 676
 - Daten verknüpfen 212
 - Path 199
 - Folder 248
 - Path (ActiveWorkbook) 257
 - Path (Application) 257
 - PDF-Export 184, 202
 - Periodischer Aufruf von Prozeduren 133
 - Personal.xlsb 75, 302
 - Persönliche Makroarbeitsmappe 75, 302
 - Pi 84
 - PIA (Primary Interop Assembly) 779, 783
 - Picture 371
 - Pictures 778
 - PictureAlignment 371
 - PictureSizeMode 371
 - PictureTiling 371
 - PivotCache 712
 - Pivotdiagramme 507, 703
 - mit Tabelle verbinden 717
 - per Code erzeugen 706
 - Pivotelemente 710
 - Pivotfelder 709
 - PivotItem 710

- Pivottabellen 689
 - Beispiel 690
 - berechnete Felder 710
 - erzeugen per Code 704
 - Gruppierung 710
 - Layout 693
 - löschen per Code 706
 - Makroaufzeichnung 707
 - Programmierung 703
 - Syntaxzusammenfassung 718
- PivotTableWizard 705
- Placement 380, 572
- PlotArea 514
- Point 515
- Polymorphismus 145, 403
- Popup-Menüs 409
- Position 402
- Power Query 622, 624
 - Abfrage-Editor 624
 - Connection-String 629
 - Erweiterter Editor 626
 - Formelsprache M 626
 - ListObject-Objekt 629
 - Manuelle Abfrage 624
 - Navigator 624
 - NWind.mdb 624
 - Programmierung 628
 - Queries-Auflistung 628
 - QueryTable-Objekt 630
 - SQL-Befehl 631
 - WorkbookQuery-Objekt 53, 628
- Precedents 176
- Preserve 86
- Primary Interop Assembly (PIA) 779, 783
- Print 69, 74
- Print # 254
- PrintArea 299
- PrintObject 485
- PrintOut 521
 - Diagramme, Beispiel 526
 - Tabelle 493, 501
- PrintTitleColumns 299
- PrintTitleRows 299
- Private 100
- Private Prozeduren 100
- Programm
 - aktivieren 786
 - Anweisungen überspringen 323
 - im Einzelschrittmodus ausführen 323
 - starten 786
 - steuern 788
 - Unterbrechung 310, 330
- Programmcode
 - Block ein-/ausrücken 71
 - Programmfehler
 - in verschachtelten Prozeduren 328
 - Programm fortsetzen 327
 - Programmieretechniken 171
 - Programmierumgebung 69
 - Projekteigenschaften 68
 - Projektfenster 64
 - Prompt 653
 - PromptForSummaryInformation 297
 - Properties 137
 - Properties (ADO) 652
 - Property (ADO) 652
 - Property Get 144
 - Property Let 144
 - Property Set 144
 - Protect 289f., 297
 - ProtectContents 289
 - ProtectDrawingObjects 289
 - Protection 290, 294
 - ProtectScenarios 289
 - ProtectSharing 292
 - ProtectStructure 290
 - ProtectWindows 290
 - Protokolle 521
 - Provider 649
 - Prozedur 89
 - aufrufen 95
 - aus anderen Arbeitsmappen 101
 - benannte Parameter 96
 - definieren 70
 - erlaubte Namen 90
 - Felder als Parameter 92
 - Gültigkeitsbereich 98
 - logische Bedingungen 103
 - Matrizen als Parameter 94
 - Namenstabelle aufräumen 101
 - optionale Parameter 94
 - Parameterliste 91
 - periodisch aufrufen 133
 - variable Parameteranzahl 95
 - vorzeitig verlassen 90
 - Prozedurale Programmierung 89
 - Syntaxzusammenfassung 108
 - Prozedurliste 321
 - Public
 - Beispiel 608
 - Prozeduren 101
 - Type 84
 - Variablen 98
 - Publish 747
 - PublishObject[s] 746
 - Put 256

Q

- Querformat 299
- Queries-Auflistung 628
 - Add-Anweisung 628
- QueryClose 376
- QueryTable 643 f.
 - Datenbankimport 643
 - HTML-Import 744
 - Textimport 262, 265
- QueryTable-Objekt 630
 - CommandText-Eigenschaft 631
- QueryType
 - PivotCache 713
 - QueryTable 644

R

- Rahmen 192
 - alle löschen 193
- Rahmenfeld 363
- RaiseEvent 144
- Randomize 216
- Range 172
 - PDF- und XPS-Export 184
- RangeSelection 173
- Read 251
- ReadAll 251
- ReadLine 251
- ReadWrite.bat 832
- Ready 311
- Rechenfunktionen in MS Query 638
- Rechnen mit Datum und Zeit 232
- Rechnungsformular 488
- Recognize 749
- RecordCount 660
 - PivotCache 712
- Recordset 655
 - Beispiel 649
 - in Tabellenblatt kopieren 662
 - Navigation 659
 - Typen 656
 - verändern 661
- Redim 85
- Redundanz 618
- RefEdit 372
- ReferenceStyle 185
- RefersTo 196
- RefersToR1C1 196
- RefersToR1C1Local 196
- Refresh
 - QueryTable 265
 - XmlDataBinding 734
 - XmlMap 731
- Regedit.exe 763

- Registerkarten *siehe* Menüband 421
 - aktivieren 43
- Rekursion 97, 158
- Relationale Datenbanken 617
 - Grundlagen 618
 - MS-Query 638
- Relationen 619
- Relative Adressen 177
- Relative Makroaufzeichnung 13
- Remove 147
- RemoveControl 364
- RemoveItem
 - CommandBarComboBox 405
- RemoveSubtotal 688
- Repeating 732
- Replace 184, 218
 - Beispiel 677
- ReplaceFormat 184
- Report (Access) 765
- Reset 254
- Resize 176, 722
- Resume 327
- RGB 573
- RGB-Funktion 537, 545
- RGP 282
- RibbonX 418
 - Schemadefinition 425
- RibbonX-Controls 430
 - button 430
 - checkBox 433
 - comboBox 439
 - dialogBoxLauncher 441
 - dropDown 439
 - dynamicMenu 436
 - editBox 433
 - für Backstage-Ansicht 463
 - gallery 442
 - hyperlink 463
 - imageControl 464
 - labelControl 430
 - menu 434
 - menuSeparator 439
 - splitButton 438
 - toggleButton 432
- RibbonX-Oberfläche 42
- Right 217
- RightFooter 299
- RightHeader 299
- RightMargin 299
- Rmdir 275
- Rnd 216
- RootElementName 731
- RootElementNamespace 731
- Round 215

- RoundDown 215
- RoundUp 215
- Row 176
 - Beispiel 614
- Rows 176
- RowDifferences 176
- RowFields 709
- RowGrand 708
- RowHeight 180
- RowRange 708
- RowSource 350, 357
- Rückgabeparameter 91
- Rückgängig-Kommando 132
- Run 96, 314
- RunAutoMacros 128
- Runden von Zahlen 214
- Rundungsfehler 105

- S**
- Save 199
- SaveAs 199, 267
 - Beispiel 493
- SaveChartTemplate-Methode 519
- SaveCopyAs 199
- Saved 199
- Schaltjahre 228
- Schemadefinition 425
- Schematische Darstellungen 548
- Schleifen 105
 - Rundungsfehler 105
 - Syntaxzusammenfassung 108
- Schlüssel 577
- Schutz
 - Arbeitsmappen 289
 - Benutzeroberfläche 289
 - Blätter 288
 - Blattreihenfolge 289
 - Fensteranordnung 289
 - Mechanismen 286
 - Passwort für Zellbereich 291
 - Symbolleisten 294
 - Szenarios 289
 - Tabellenblatt 288, 380
 - Zeichnungsobjekte 289
 - Zellen 288, 479
 - Zellen, Beispiel 491
- ScreenUpdating 307
- Scripting Runtime Bibliothek 243
- ScrollArea 288
- ScrollBar 369
- ScrollBars (Rahmenfeld) 364
- ScrollColumn 200
- ScrollHeight 364
- ScrollRow 200

- ScrollWidth 364
- Scrrun.dll 118, 243
- SearchFile
 - Aufruf 159
 - Execute 156
 - FileTypes 155
 - FoundFiles 155
 - LookIn 155
 - NewSearch 156
 - SearchSubFolders 155
- SearchFile-Klasse 53, 153
- Second 229
- Seek 254
- Seiteneinstellung 299
- Seitenvorschau 501
- SEKUNDE 231
- Select 174
 - Blatt 202
 - Diagramme 517
 - OLE 775
- Select Case 104
- SELECT INTO 666
- SELECT (SQL) 663
- SelectedSheets 201
- Selection 173
 - PDF- und XPS-Export 184
- SelectionChange 131
- SendKeys 20, 788
- SendMail 742
- Sequentielle Dateien 254
- Series 515
- Set 121
- SetAttr 275
- SetBackgroundPicture 298
- SetFocus 347
- SetSourceData 706, 717
- SetText
 - Clipboard 772
- SetValue 734
- Sgn 216
- Shadow 572
- Shape 380
 - SmartArt-Diagramm 550
 - Überblick 570
- ShareName 245
- Sheets 201
- Sheet_Activate 130
- SheetBeforeDoubleClick 131
- SheetCalculate 131
- SheetChange 131
- Sheet_Deactivate 130
- SheetsInNewWorkbook 297
- Shell 786
- Shell-Objekt 564

- ShortName 248
- ShortPath 248
- Show 333
 - FileDialog 259
- ShowDataForm 594
 - Beispiel 612
 - Zellbereich Datenbank 20
- ShowLevels 688
- ShowPopup 409
- ShowTotals 723
- ShowWindow 791
- Sicherheitscenter 165
 - Digitales Zertifikat 169
 - Vertrauenswürdige Dokumente 169
 - Vertrauenswürdige Speicherorte 168
- Signatur 427, 432
- Signum-Funktion 216
- Single 81
- Size
 - File / Folder 248
- Skip 251
- SkipLine 251
- Skriptdatei 812
- SmallChange 369
- SmartArt-Diagramme 46, 548
 - Anlegen 549
 - Beschriftung 552
 - Eigene Diagrammlayouts 554
 - Knoten 552
 - Knoten anlegen 554
 - Knoten herabstufen 553
 - Knoten heraufstufen 554
 - Knoten vergrößern 553
 - Knoten verschieben 553
 - Layout festlegen 550
 - Programmierung 549
 - Unterknoten anlegen 554
 - Verweis beschaffen 551
- SmartArt-Objekt 53
- SmartTagActions 749
- SmartTagRecognizers 749
- Smart Tags 747
 - anlegen 55
- SmartTags 749
- SOAP (Simple Object Access Protocol) 751
- Soap Type Library 750
- Solver-Add-In 194
- SolverOk 194
- SolverOptions 194
- SolverSolve 194
- Sort
 - Beispiel 687
- Sort-Objekt 593
- Sorted 356
- Sortieren
 - Datenbanken 585
 - MS Query 640
 - Vergleich von Zeichenketten 219
- SourceType 746
- Space 218
- Spalten 176
 - ausblenden 287
- Sparklines-Diagramme 47, 543
 - anlegen 544
 - Datenpunkte anzeigen 547
 - Datenpunkte kennzeichnen 546
 - Erster und letzter Datenpunkt 547
 - Liniendicke ändern 545
 - Linienfarbe ändern 545
 - Negative Werte 547
 - Programmierung 544
- SpecialCells 176
- Speicheroptionen 295
- Speicherplatz sparen 581
- Spezialfilter 589
- Spezialschrift-Beispiel 191
- SpinButton 369
- Split 200, 218, 298
- SplitColumn 200, 298
- SplitRow 200, 298
- Sprungmarke 327
- Sql
 - QueryTable 643
- SQL 640
 - Beispiele 664
 - Grundlagen 663
 - Syntax 663
- sql2string.exe 707
- SQL-Befehl 631
- SQL Server 653
- SQLLEDB 653
- Standarddialoge 333
- StandardFont 297
- StandardFontSize 297
- StartupPath 257
- StartupPosition 374
 - Beispiel 387
- Startverzeichnis, Xlstart 302
- Static 99
- Statische Variablen 99
- Statistikfunktionen 216
- StatusBar 308
- Statuszeile 308 f.
- STDEV (SQL) 678
- StdOle2.tlb 118
- Step 105
- Steuerelement 342, 812
 - Aktivierreihenfolge 347

- in Arbeitsblättern 57
- Eigenschaften 344
- Ereignisprozeduren 344
- neu erstellen 349
- in Tabellen 490
- direkt im Tabellenblatt 377
- Typen 350
- Zugriffstaste 346
- Steuerung fremder Programme 788
- Str 222
- StrComp 219
- Strg+Untbr 310, 330
- String 81, 216, 218
- StrReverse 218
- STUNDE 231
- Style 180
 - CommandBarButton 404
 - Listenfeld 672
- Styles 180
- Stylesheet 812
- Sub 89
- SubFolders 248
- Submenüs 405
- Subtotal 687
- Subtotals 709f.
- SubType 513
- Suchen (Find-Methode) 184, 592
- Suchen in Tabellen 587
- Suchkriterien 587
- Sum 214
- SummaryColumn 688
- SummaryRow 688
- SUMME 214
- Sunburst 41, 48, 53, 556
- Sunburst-Diagramm
 - Definition 568
 - P 569
- Supports 658
- Symbolleiste für den Schnellzugriff 44, 455
 - Änderungen revidieren 457
 - Änderungen sichern 456
 - Befehle einfügen 455
 - Befehle entfernen 456
 - Befehle für das aktuelle Dokument hinzufügen 458
 - Controls hinzufügen 457
 - Makros integrieren 455
 - manuell anpassen 455
 - Programmierung 457
- Symbolleisten 393, 395
 - anfügen 400
 - bearbeiten 395
 - kopieren 413
 - Objekthierarchie 401

- schützen 294
- speichern 400
- Symbolsatzdiagramm 536
 - Anlegen 540
 - Anzeige von Zellwerten unterdrücken 540
 - Gestaltung 540
 - Symbolsatz wählen 540
 - Wertebereiche festlegen 541
 - Wertzuordnung ändern 540
- Syntaxfehler 69
- Syntaxüberprüfung 66
- Syntaxzusammenfassung
 - Arbeitsmappen 208
 - Blätter 208
 - Dateien 273, 275
 - Datenbanken 595
 - Datum und Zeit 241
 - Diagramme 534
 - DLL 760
 - Ereignisse 138
 - Fenster 208
 - Klassen 160
 - Operatoren 163
 - Optionen 296
 - Pivottabellen 718
 - prozedurale Programmierung 108
 - Variablenverwaltung 88
 - Zahlen und Zeichenketten 223
 - Zellbereiche 195
- Szenarios schützen 289

T

- Tabellen
 - Ereignis bei Eingabe 131
 - Ereignis bei Neuberechnung 131
 - platzsparend speichern 581
 - sichtbaren Bereich verkleinern 287
- Tabellenblatt
 - effizienter Umgang 312
 - Inhalte aus Datenfeldern kopieren 314
 - Inhalte aus Feldern kopieren 313
 - schützen 380
 - Werte rasch eintragen 312
- Tabellenfunktionen 214
 - eigene 26, 277
- Tabs 44
- TablIndex 347
- TableRange1/2 708
- TabOrientation 366
- TabRatio 298
- TabStop 347
- TabStrip 365
- Tabulatorweite 71

- Tag 350
 - Beispiel 410
 - TAG 231
 - Tage360 229
 - TAGE360 231
 - Tagesdatum 229
 - Tagesprotokoll 526
 - TakeFocusOnClick 363, 382
 - Tastaturfokus, Probleme 382
 - Tasteneingabe 132
 - Tastenkürzel 14, 44
 - in Visual Basic 76
 - Teilergebnisse 685
 - TemplatesPath 257
 - Temporäre Dateien 247
 - Temporäres Verzeichnis 247
 - Terminate 145, 375
 - Testfenster 74
 - Text 179, 219
 - CommandBarComboBox 405
 - TextAlign 351
 - Textarea-Control 827
 - Textassistent 260
 - Textausrichtung (Winkel) 179
 - TextBox 352
 - TextColumn 358
 - Textdateien
 - importieren 260
 - Open 254
 - TextStream 251
 - Texteditor 815
 - TextEffect 572
 - Textexport 267
 - Textfeld
 - synchronisieren mit Bildlaufleiste 385
 - TextFilePromptOnRefresh 267
 - Textimport
 - OpenText 263
 - QueryTable 265
 - TextStream 251
 - ThisWorkbook 198
 - ThreeD 572
 - Time 229
 - TimeSerial 229
 - TimeValue 229
 - Timer 229
 - Tipps und Tricks 307
 - ToggleButton 362
 - TooltipText 405
 - Beispiel 410
 - CommandBarButton 404
 - Top 380
 - Range 339
 - Window 200
 - TopLeftCell 380, 572
 - TopMargin 299
 - TotalsCalculation 723
 - TotalSize 245
 - TransitionEffect 366
 - TransitionPeriod 366
 - Treemap 41, 48, 53, 556
 - Treemap-Diagramm
 - Definition 563
 - Inhaltsverzeichnisse visualisieren 564
 - Programmierung 563
 - Trendline 516
 - Trendlinien 508f.
 - TripleState 361
 - True 103
 - Trust Center 819
 - Trusted Connection 654
 - Type 83
 - Chart 513
 - File 249
 - Folder 248
 - Get / Put 257
 - Shape 571
 - TypeName 123, 317
 - Typenkonvertierungen 83
- ## U
- Überlauf 83
 - Überwachungsausdruck 324
 - Überwachungsbereich 74
 - UBound 86, 92
 - UCase 217
 - UDDI 751
 - Uhrzeit
 - durch Drehfeld einstellen 390
 - Umgang mit 227
 - Umfrage 668
 - Umrandung 192
 - Umschaltbuttons 362
 - Umwandlungsfunktionen 221
 - UNC-Pfad 819
 - UND 482
 - Undo 72
 - Ungroup 688
 - Unicode 217
 - Dateien 251, 255
 - Union 176, 188
 - Unlist 723
 - Unload 33, 341, 375
 - Unprotect 289
 - UnprotectSharing 292
 - Unterbrechung 310, 330
 - Untermenüs 405
 - Unterprogramm *siehe* Prozedur 89

- Update 661
 - OLE 776
- UPDATE 666
- UsableHeight 200
- UsedRange 173
- Users 292
- UserAccess 292
- UserAccessList 292
- UserForm 340, 374
- UserLibraryPath 257

- V**
- Val 221
- Validitätsregeln 480
- Validitätskontrolle 24
- Value
 - ListBox 357
 - OptionButton/CheckBox 361
 - Range 179
- Value2 179
- Variablen 79
 - aus anderen Arbeitsmappen 101
 - Definition 80
 - Gültigkeitsbereich 98
 - in Dialogmodulen 344
 - Lebensdauer 99
 - Namen 80
 - Namenstabelle aufräumen 101
 - öffentliche 98
 - statische 99
 - Syntaxzusammenfassung 88
 - Typen 81
 - Verwaltung 79
- Variablendeklaration 67
- Variant 82
- VarType 82
- VBA 3
 - Bibliothek 117
 - Einführung 6
 - Entwicklungsumgebung 61
 - herkömmliche Makros 314
- VBA-Grenzen 55, 796
- VBA-Projektobjektmodell 167
- VBComponent 136
- VBE-Bibliothek 136
- VBIDE, Beispiel 136
- VB.NET, Automation 778
- VBProject 137
- Verb 775
- Verbunddiagramm 506
- Vergleichsoperatoren 161
- Vergleich von Zeichenketten 219
- Verlag.mdb 617
- Versetzen von Zellbereichen 210

- Version 315
- Verteiler 486
- VerticalAlignment 179
- Verweis
 - andere Arbeitsmappen 119
 - auf Objekte 121
 - Objektbibliotheken 119
- Verzeichnis 242, 247
 - aktuelles 246
 - erzeugen/kopieren/löschen 249
 - rekursiv verarbeiten 250
 - temporäres 247
- Verzeichnisauswahldialog 259
- Verzeichnisbaum lesen 250
- Verzinsungstabelle 21
- Verzweigungen 102
 - Syntaxzusammenfassung 108
- Viren 303
 - Add-ins 166
 - Selbst entdecken 168
 - VBA-Projektobjektmodell 167
- Virenschutzoptionen 296
- Visible 200, 287, 350
 - Blatt 202
 - Diagrammfenster 517
 - Name 177
 - Workbook 298
 - Worksheet 298
- VisibleFields 709
- Visual Basic
 - ActiveX-Automation 768
 - für Applikationen 3, 6
- Visual Studio 815
- Visual Studio Tools 52
- Visual Studio Tools for Office 55, 286, 742, 748, 779, 796
 - Add-in-Projekte 802
 - Application-Objekt 802
 - Aufgabenbereich gestalten 803
 - Autokorrektur bei Code-Eingabe 801
 - Code-Snippets 801
 - Dokumentbasierte Projekte 800
 - Ereignisroutinen 802
 - IntelliSense 801
 - Preis 799
 - Projektvorlagen 799
 - Sicherheit 798
 - Steuerelemente 800
 - ThisApplication-Objekt 802
 - ThisWorkbook-Objekt 802
 - Unsichtbare Controls 801
 - Verwalteter Code 798
 - Web Services abfragen 808
- Vlassist.xla 487

Volatile 283
 VolumeName 245
 Vordefinierte Dialoge 333
 Vorlagenassistent 485
 Vorläuferprozedur 321
 Vorschau 480
 Vorzeichen 216
 VSTO 52, 55, 796

W

Währungsformat 183
 Warnungen nicht anzeigen 310
 Wasserfall 41, 48, 53, 556
 Wasserfall-Diagramm
 – Definition 556
 – Programmierung 557
 Watch-Expression 324
 Webdienst 808
 Web-Import 744
 Web-Optionen 295
 WebOptions 747
 Webseite 812
 Web Service 808
 Web Services 750
 Web Services Toolkit 808
 WeekDay 229
 WeekdayName 231
 Wend 108
 WENN, Beispiel 23, 481
 Werkzeugfenster 348
 Werte kopieren 676
 Wertparameter 92
 WHERE (SQL) 663
 While 108
 Width 380
 – Dialog 388
 – PlotArea 515
 – Window 200
 Wiederherstellen-Kommando 132
 Window
 – Ereignisse 131
 – Optionen 298
 WindowActivate 131
 WindowDeactivate 131
 WindowResize 131
 WindowState 200
 Windows 199
 Windows-Verzeichnis 247
 – mit DLL-Funktion ermitteln 760
 Winkel 179
 WithEvents 134
 Wochentag 229
 WOCHENTAG 231
 WordWrap 351

Workbooks 198
 – Einstellung von Optionen 297
 – Ereignisse 129, 139
 – PDF- und XPS-Export 202
 WorkbookAfterXmlExport 736
 WorkbookBeforeXmlExport 736
 Workbook_(De)Activate
 – Beispiel 415
 WorkbookQuery-Objekt 53, 628
 WorksheetFunction 214
 Worksheets 201
 – Ereignisse 129, 140
 – PDF- und XPS-Export 202
 WrapText 179
 WriteBlankLines 251
 Write, TextStream 251
 WriteLine, TextStream 251
 WSDL 751
 Wurzelverzeichnis 248

X

Xl8galry.xls 306
 Xlstart-Verzeichnis 75, 302
 XML 723, 751
 – Ereignisse 736
 – Microsoft XML Library 750
 – Schema 724
 XML-Attribute 425
 – autoScale 448
 – boxStyle 449
 – columns 442
 – columnWidthPercent 466
 – description 470
 – getContent 436
 – getItemCount 444
 – getItemID 444
 – getItemLabel 444
 – getPressed 432f.
 – getSelectedItemIndex 444
 – getText 433, 440
 – id 427
 – idMso 426, 450, 460
 – image 430, 464, 468
 – imageMso 427f., 430, 433, 435, 464, 470, 472
 – insertAfterMso 464, 471
 – insertBeforeMso 460
 – itemHeight 442
 – itemWidth 442
 – label 427, 450, 464, 470f.
 – layoutChildren 467
 – onAction 427, 432f., 435, 438, 441f., 445, 464
 – onChange 434, 441

- onLoad 446
- rows 442
- size 427
- startFromScratch 425, 457
- style 466
- target 463, 468
- visible 426
- xmlns 462
- XmlDataQuery 732
- XML-Datei 812
- XmlImport 730
- XmlImportXml 731
- XmlMap 731
- XmlMapQuery 732
- XML-Tags
 - backstage 464
 - bottomItems 467
 - box 449
 - button 427, 430, 464
 - category 469f.
 - checkBox 433
 - comboBox 439
 - contextMenu 459
 - contextMenus 459
 - control 450, 457, 460
 - customUI 425, 462
 - dialogBoxLauncher 441
 - documentControls 458
 - dropDown 440
 - dynamicMenu 436
 - editBox 433, 467
 - firstColumn 466
 - gallery 442
 - group 427, 448, 466, 470
 - hyperlink 463, 468
 - imageControl 464
 - item 439
 - labelControl 430
 - layoutContainer 467
 - menu 434, 450, 461
 - menuSeparator 439, 451
 - primaryItem 466
 - ribbon 425, 457
 - secondColumn 468
 - sharedControl 457
 - splitButton 438
 - tab 426, 465, 471
 - tabs 426, 457
 - task 470, 472
 - taskFormGroup 469
 - toggleButton 432
 - topItems 467, 470
- Xor 162
- XPath 726, 731

- XPS-Export 184, 202
- XSD-Datei 724
- XSLT 726

Y

- Y2K 225
- Year 229

Z

- Z1S1-Schreibweise 185
- Zahlen
 - runden 214
 - Syntaxzusammenfassung 223
 - umwandeln 214
 - in Zeichenketten 222
- Zahlenformate
 - mehrfarbige 482
 - für Zeiten in Zellen 235
- Zahlenformatierung 180
- Zeichenketten 216
 - eingeben 337
 - ein- und ausgeben 220
 - Groß-/Kleinbuchstaben 217
 - in Zahlen umwandeln 221
 - lesen / speichern 257
 - Mustervergleich 220
 - suchen 217
 - Syntaxzusammenfassung 223
 - vergleichen 219
- Zeichnungsobjekte
 - schützen 289
 - *siehe* Shape 570
- Zeilen 176
 - ausblenden 287
- Zeit
 - Beispiel 499
 - durch Drehfeld einstellen 390
 - Konvertierung von/zu Zeichenketten 230
 - rechnen mit 232
 - Syntaxzusammenfassung 241
- ZEIT 231
- Zeitdifferenz
 - in Jahren 233, 582
 - in Monaten 234
- Zeitereignis 133
- Zeitformate in Zellen 235
- ZEITWERT 231
- Zellen
 - Cells 175
 - schützen 288, 479
 - schützen, Beispiel 491
 - Umgang mit 171
- Zellen kopieren
 - Werte 676

Zellbereich

- Bewegungsradius einschränken 288
- *siehe* Bereich 172
- Umgang mit 171

Zellbezüge, Schreibweise 185

Zelldiagramme 46f., 535

Zielwertsuche 194

Zoom 200, 298

Zufallszahlen 216

- Beispiel 524

Zugriffstaste 346

Zusammengesetzte Bereiche 189

Zusatzsteuerelemente 348

Zwischenablage 210

- ActiveX-Automation 772

Zwischenergebnisse 710