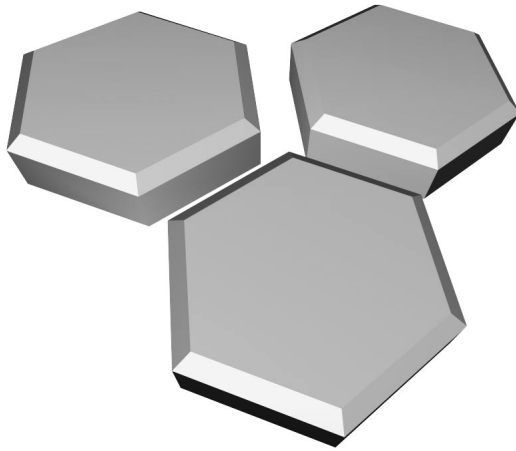


Das Excel-VBA Codebook



Monika Weber,
Melanie Breden

Das Excel-VBA Codebook

eBook

Die nicht autorisierte Weitergabe dieses eBooks
an Dritte ist eine Verletzung des Urheberrechts!



ADDISON-WESLEY

An imprint of Pearson Education

München • Boston • San Francisco • Harlow, England
Don Mills, Ontario • Sydney • Mexico City
Madrid • Amsterdam

Bibliografische Information Der Deutschen Bibliothek

Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über <<http://dnb.ddb.de>> abrufbar.

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

Für Verbesserungsvorschläge und Hinweise auf Fehler sind Verlag und Herausgeber dankbar.

Alle Rechte vorbehalten, auch die der fotomechanischen Wiedergabe und der Speicherung in elektronischen Medien.

Die gewerbliche Nutzung der in diesem Produkt gezeigten Modelle und Arbeiten ist nicht zulässig.

Falls alle Hardware- und Softwarebezeichnungen, die in diesem Buch erwähnt werden, sind gleichzeitig auch eingetragene Warenzeichen oder sollten als solche betrachtet werden.

Umwelthinweis:

Dieses Buch wurde auf chlorfrei gebleichtem Papier gedruckt.

Die Einschrumpffolie – zum Schutz vor Verschmutzung – ist aus umweltverträglichem und recyclingfähigem PE-Material.

10 9 8 7 6 5 4 3 2 1
06 05 04

ISBN 3-8273-2101-8

© 2004 by Addison-Wesley Verlag,

ein Imprint der Pearson Education Deutschland GmbH

Martin-Kollar-Straße 10–12, D-81829 München/Germany

Alle Rechte vorbehalten

Korrekturat: G & U Technische Dokumentation GmbH, Flensburg

Lektorat: Frank Eller, feller@pearson.de

Herstellung: Monika Weiher, mweiher@pearson.de

Satz: reemers publishing services gmbh, Krefeld – gesetzt aus der Minion

Umschlaggestaltung: Marco Lindenbeck, mlindenbeck@webwo.de

Druck und Verarbeitung: Kösel, Kempten (www.Koeselbuch.de)

Printed in Germany

Inhaltsverzeichnis

Teil I Einführung	17
Vorwort	19
Einführung	21
1 Hinweise zum Buch	21
2 Die Geschichte von Excel	22
Teil II Rezepte	27
Grundlagen	29
1 Sicherheitseinstellungen in Excel	29
2 Zellen selektieren oder nicht?	30
3 Tabellenblätter selektieren oder nicht?	33
4 Codezeilen kommentieren	34
5 Codezeilen umbrechen	35
6 Die letzte (nicht) belegte Zelle ermitteln	35
7 Löschen von Zellen, Zellinhalt, Spalten und Zeilen	37
8 Ausschneiden, kopieren und einfügen von Zellinhalt	38
9 Löschen von Tabellenblättern	38
10 Einfügen von Tabellenblättern	39
11 Arbeiten mit Kopf- und Fußzeilen	40
12 Kopf- und Fußzeilen mit Bildern	41
13 Kopf- und Fußzeilen auf jedem Tabellenblatt	42
14 Eine Farbtabelle erstellen	43
15 RGB-Farben	44
16 Kommentare einfügen	46
17 Kommentare ergänzen	48
18 Kommentare auslesen	49
19 Kommentare formatieren	51
20 Kommentare löschen	52
21 Verfügbare Systeminformationen auslesen (Environ)	53
22 Eigenschaften auslesen	54
23 Eigenschaften ausfüllen bzw. erstellen	56
24 Funktionen	57
25 Excel-Tabellenfunktionen in Visual Basic verwenden	58
26 VBA-Funktionen	60
27 Benutzerdefinierte Tabellenfunktionen	61
28 Funktion einer Kategorie zuweisen	62
29 Beschreibung einer Funktion hinzufügen	65
30 Pfad, Dateiname und Blattname ermitteln	67
31 Flüchtige Tabellenblattfunktionen	69
32 Alle Blattnamen einer Arbeitsmappe ermitteln	70
33 Existiert ein bestimmtes Tabellenblatt?	71
34 Ist eine Arbeitsmappe geöffnet?	72

35	Aufrufende Formelzelle ermitteln	73
36	Fehlerwerte in Funktionen	73
37	Spaltenbuchstabe ermitteln	76
38	Funktion Seitenzahl	77

Allgemein 81

39	Formeln mit Zirkelbezügen berechnen	81
40	VBA-Addition mit eigenem Zellenwert	83
41	Zellenwert kumulieren	84
42	Welche Zellen sind gesperrt?	86
43	Gesperrte Zellen einfärben	88
44	Formelzellen schützen	90
45	Autofilter trotz Blattschutz	91
46	Gliederung trotz Blattschutz	92
47	Sortierung trotz Blattschutz	92
48	Schutzoptionen auslesen	94
49	Schutzoptionen einstellen	98
50	Nicht gesperrte Zellen markieren und formatieren	99
51	Nicht gesperrte Zellen formatieren (ab Excel 2002)	100
52	Nicht gesperrte Zellen formatieren (alle Excel-Versionen)	101
53	Schutzoptionen ändern	101
54	Benutzerberechtigte Bereiche hinzufügen	104
55	Benutzerberechtigte Bereiche löschen	105
56	Schutz der benutzerberechtigten Bereiche aufheben	106
57	Benutzerberechtigte Bereiche schützen	107
58	Benutzerberechtigte Bereiche – Passwort ändern	108
59	Benutzerberechtigte Bereiche – Titel ändern	108
60	Benutzerberechtigte Bereiche – Bereich ändern	111
61	Ausgewählte Bereiche freigeben	113
62	Löschen von Blättern verhindern	117
63	Schreibgeschützte Datei ohne Speichern-Rückfrage schließen	119
64	Entfernen des Schreibschutzes ignorieren	120
65	Benutzerdefinierte Ansichten schützen	121
66	Informationen des Betriebssystems auslesen	125
67	Zwei Benutzernamen auslesen	127

Datum/Zeit 129

68	Datum und Zeitformat anpassen	129
69	Wochentag des Geburtstages ermitteln	133
70	Anzeigen des Systemdatums	133
71	Schaltjahre bestimmen	134
72	Dateizugriffe dokumentieren	135
73	Bestandsveränderungen dokumentieren	137
74	Vereinfachte Datumseingabe	139
75	Uhrzeiten mit Autokorrektur	140
76	Uhrzeiten ohne Doppelpunkt eingeben	141
77	Aktuelles Datum finden	144
78	Beliebiges Datum finden	144
79	Arbeitsstunden in Meldungsfenster anzeigen	146
80	Arbeitszeiten summieren	147
81	Kalenderwoche nach DIN 1355 berechnen	149

82	Anzahl Kalenderwochen bestimmen	151
83	Feiertage mit Formeln berechnen	152
84	Formeln in Tabelle anzeigen	152
85	Funktion zur Formelanzeige	154
86	Feiertage mit VBA berechnen	155
87	Tageskalender erstellen	156
88	100-jähriger Kalender und Arbeitszeittabelle	160
89	Lohn berechnen	163
90	Minuszeiten berechnen	164
91	Negative Zeitberechnung mit Funktion	165
92	1904-Datumswerte für Minuszeiten	166
93	Anzeige des eingestellten Datumssystems in Statuszeile	168
94	Eingabe von Minuszeiten	170
95	1904-Datumswerte mit VBA kopieren	170
96	Datumswerte aus anderem Datumssystem importieren	172
97	Datums-Funktionen bei 1904-Datumswerten	174
98	Berechnung von Datumsdifferenzen	175
99	Zeitdifferenzen mit VBA DateDiff	177
100	VBA-Ersatz-Prozedur für die DateDiff-Tabellenfunktion	179
101	Funktion, um Lebensalter zu berechnen	182
102	Stoppuhr mit Excel	183
103	Excel kann warten	186
104	Signaltöne erzeugen	186
105	Töne bei aktuellem Datum ausgeben	187
106	Meldung zeitbegrenzt anzeigen	188

Auswertungen **191**

107	Bezüge archivieren	191
108	Bezug-Fehler reparieren	194
109	Tabellenblätter vergleichen (Fehlerausgabe in einem separaten Tabellenblatt)	196
110	Tabellenblätter vergleichen (Fehlerausgabe in einer separaten Mappe)	200
111	Formeln in einem Tabellenblatt ermitteln	204
112	Einfacher Autofilter	206
113	Autofiltern nach Datum	209
114	Autofilter deaktivieren	211
115	Spezialfilter mit mehr als zwei Kriterien	211
116	Doppelte Einträge ausblenden	213
117	Spezialfilter: Alle Datensätze wieder einblenden	214
118	Spezialfilter als Ereignisprozedur	214
119	Gültigkeit	216
120	Gültigkeit auf ein anderes Tabellenblatt	218
121	Gültigkeit als Ereignis-Prozedur	219
122	Eine Pivot-Tabelle erzeugen	221
123	Zwei Pivot-Tabellen auf einem Blatt erzeugen	225
124	Eine bestehende Pivot-Tabelle formatieren	227
125	Datenpunkte ausblenden	228
126	Pivot-Tabellen berechnen	230
127	Sortieren nach Farben	231
128	Sortieren nach selbst festgelegter Farbfolge	234
129	Tabellenblätter alphabetisch sortieren	236
130	Tabellenblätter nach Codenamen sortieren	239

131	Tabellenblätter numerisch sortieren	241
132	Tabellenblätter nach Farben sortieren	242
Steuerelemente		245
133	Steuerelemente der Formular-Symbolleiste	246
134	Formular-Schaltfläche erstellen	248
135	Formular-Schaltfläche löschen	249
136	Makro an Formular-Schaltfläche zuweisen	250
137	Steuerelement-Toolbox	250
138	Steuerelemente erstellen	254
139	Steuerelemente löschen	256
140	Alle Steuerelemente löschen	256
141	Steuerelemente eines Typs löschen	257
142	Fokus der Steuerelemente verwalten	257
143	Wert in Kontrollkästchen (CheckBox) ändern	258
144	Kontrollkästchen (CheckBox)-Wert auslesen	260
145	Wahrheitswert eines Kontrollkästchens übertragen	261
146	Steuerelement-Eigenschaften ändern	261
147	Mit Umschaltfläche Steuerelement steuern	262
148	Gruppen von Steuerelementen erstellen	264
149	Optionsfelder auswerten	266
150	Optionsfelder und Kontrollkästchen zurücksetzen	267
151	Arbeitsblätter über Listefeld auswählen	271
152	Listefeld durch Mausbewegung füllen	272
153	Monate in Listefeld auflisten	273
154	ListefeldEinträge bei Auswahl ändern	274
155	Mehrspaltiges Listefeld füllen	277
156	Kombinationsfeld nach Auswahl füllen	279
157	Passworteingabe durch Sternchen ersetzen	281
158	Textfeld-Werte umwandeln	283
159	Zeilenumbruch in Textfeld (TextBox) per Code einfügen	284
160	Mit Drehfeld (SpinButton) Monatswerte verändern	285
161	Mit Bildlaufleiste (ScrollBar) zu Datum wechseln	287
162	Steuerelemente mit Tastatur aktivieren	290
163	Tabellenblatt-Zugriffe mit Steuerelementen verwalten	292
Befehlsleisten		301
164	Befehlsleistendatei finden	302
165	Befehlsleistendatei sichern	303
166	Symbolleistenkonfigurationen speichern	305
167	Befehlsleistendatei laden	307
168	Originaldatei *.xlb laden	308
169	Befehlsleisten ermitteln	309
170	Menüs und Steuerelemente ermitteln	311
171	Menü-Auflistung im Direktfenster	312
172	Ganzes Menü deaktivieren oder aktivieren	313
173	Ganzes Menü über ID aus- und einblenden	314
174	Untermenü-Auflistung im Direktfenster	315
175	Untermenüs deaktivieren und aktivieren	317
176	Untermenü des Untermenüs deaktivieren und aktivieren	317
177	Befehlsleisten zurücksetzen	318

178	Menüleisten ein- und ausblenden	320
179	Symbolleisten ein- und ausblenden	321
180	Kontextmenüs einblenden	323
181	Zelle-Kontextmenü deaktivieren	324
182	Alle Befehlsleisten ein- und ausblenden	325
183	Einzelne Symbolleistentypen ein- und ausblenden	326
184	Einzelne Symbolleisten vor Änderungen schützen	328
185	Befehlsleisten vor Änderungen schützen in XL97 und XL2000	329
186	Befehlsleisten-Schutz aufheben	331
187	Befehlsleisten-Schutz wechseln	332
188	Befehlsleisten vor Änderungen schützen ab XL2002	332
189	Fehlerroutinen vor Erstellung von Befehlsleisten	334
190	Löschen von Befehlsleisten	335
191	Erstellen einer Menüleiste	335
192	Wechseln und Löschen von Menüleisten	337
193	Symbolleisten erstellen und verankern	338
194	Symbolleisten schwebend positionieren	339
195	Symbolleiste neben Standard-Symbolleiste andocken	340
196	Kontextmenü mit Datum und Zeit erstellen	341
197	Neues Farben-Menü erzeugen	342
198	Menü suchen und aufklappen	346
199	Arbeiten mit personalisierten Menüs	347
200	Steuerelemente kopieren	349
201	Integrierte Steuerelemente in Symbolleiste einfügen	351
202	Schaltflächen-ID ermitteln	352
203	Schaltflächen mit Quickinfo erstellen	353
204	Symbolleiste mit zwei Ebenen erstellen	354
205	Steuerelemente mit ID erstellen und anzeigen	356
206	Schaltflächen über ID löschen	357
207	Menüpunkte mit Hyperlinks	358
208	Schaltfläche einem Menü hinzufügen	361
209	Tabellenreiter über Dropdownfeld färben	363
210	Zellenschutz im Kontextmenü anzeigen und ändern	370
211	Kontextmenü durch Kaskaden-Menü ersetzen	373
212	Welches Kontextmenü wurde aufgerufen?	376
213	Symbole über Kontextmenü einfügen	377
214	Alle FaceIDs auf einen Blick	381
Objekte		383
215	Einen Bereich in eine Grafik umwandeln	383
216	Einen Bereich als Grafik exportieren	385
217	Eine verknüpfte Grafik	388
218	Eine Textbox, die den Cursor verfolgt	389
219	Eine Grafik in eine Zelle einfügen	390
220	Bei Klick auf Zelle A1 eine Grafik in Zelle B1 einfügen	392
221	Eine Grafik einfügen und der Zellengröße anpassen	393
222	Eine Grafik exakt in einen Bereich einfügen	394
223	Eine Grafik in einer Zelle zentrieren	396
224	Grafik(en) benennen	397
225	Grafik(en) kopieren	398
226	Grafik(en) löschen	399

227	Grafik formatieren	400
228	Bedingtes Einfügen einer Grafik	403
229	Eine animierte Grafik einbinden	404
230	Ein WordArt-Objekt erzeugen	406
231	Ein WordArt-Objekt formatieren	407
232	AutoFormen einfügen	410
233	AutoFormen formatieren	411
234	Objekte lokalisieren	413
235	Ein ClipArt in der Farbe verändern	413
236	Eine schematische Darstellung oder ein Organigramm erzeugen	416
237	Diagrammknoten löschen	420
238	Objekte löschen	421
239	Nur Objekte von einem bestimmten Typ löschen	422
240	Objekt-Typ ermitteln	423
241	Nur bestimmte Objekte gruppieren (Array)	424
242	Der Office-Assistent	426
243	Einen eigenen Titel und Text in der Sprechblase ausgeben	428
244	Den Pfad des Assistenten ausgeben	429
245	Schaltflächen in einer Sprechblase	431
246	Eine Sprechblase mit Aufzählungspunkten	432
247	Eine Sprechblase mit Nummerierung	434
248	Eine Sprechblase mit Kontrollkästchen	435
249	Den Text in einer Sprechblase formatieren	437
250	Grafiken in einer Sprechblase anzeigen	438
251	Eine Sprechblase mit Hyperlinks	439
252	Den Assistenten animieren	441

Diagramme **445**

253	Diagrammtypen und -index	447
254	Hintergrund Diagramm- und Zeichnungsfläche	452
255	Wände und Böden von 3-D-Diagrammen	455
256	Diagrammtitel hinzufügen	457
257	Datenreihen formatieren	460
258	Datenpunkte formatieren	462
259	Markierer und Linien formatieren	462
260	Trendlinien hinzufügen	463
261	Trendlinien entfernen	466
262	Trendlinien-Typen	467
263	Gleichungen und Bestimmtheitsmaß ein- und ausblenden	469
264	Trendperiode vor- oder rückwärts	470
265	Achsen formatieren	473
266	Achseneinstellungen speichern	477
267	Achseneinstellungen rückgängig machen	479
268	Primär- oder Sekundärachse	481
269	Datentabelle formatieren	482
270	Legende verwalten	484
271	3-D-Oberflächen-Diagramm rotieren lassen	486
272	Ein benutzerdefiniertes Diagramm erstellen	488
273	Einen benutzerdefinierten Diagrammtypen löschen	490
274	Einen benutzerdefinierten Diagrammtypen anwenden	491
275	Pivot-Diagramm	491

276	Alle Diagramme drucken und löschen	493
277	Diagramme drucken	495
278	Diagramm-Säulen mit Grafiken	496
279	Diagramm als Grafik exportieren	497
280	Objektnamen in Diagrammen ermitteln	499
281	Zellfarben an Datenreihe übergeben	499
282	Zellfarben an Datenpunkte übergeben	501
283	Minuswerte invertieren	502
284	Diagramm mit bedingter Formatierung (Säulenfarbe)	503
285	Diagramm mit bedingter Formatierung (Werte)	505
286	Halbtransparente Diagramme	507
287	Verbund-Diagramm	510
288	Blasendiagramm-Blume	513

Ereignisse 519

289	Allgemeine Prozeduren	519
290	Ereignis-Prozeduren	520
291	Workbook-Ereignisse	521
292	Aktuelles Monatsblatt anzeigen	525
293	Ausschneiden und Löschen verhindern	527
294	Ereignisse ignorieren	530
295	Ereignisse aus- und einschalten	531
296	Excel-Datei ohne Makros unbrauchbar	532
297	Datei nur mit Makrounterstützung anzeigen	540
298	Registerreiter neu nummerieren	542
299	Neues Blatt beschriften	544
300	Diagramm nach Ansicht löschen	548
301	Tabellenblatt-Ereignisse (Worksheet)	549
302	Zellenkontextmenü verhindern	551
303	Auf den Spuren von Target	552
304	Bereichsauswahl verhindern	554
305	Cursorbewegungen protokollieren	554
306	Lupe vergrößert markierten Zellenbereich	555
307	Cursor in eine Eieruhr verwandeln	556
308	Formeln in Kommentaren anzeigen	557
309	Bei Blattwechsel Monatsspalten anzeigen	558
310	Blattschutz ohne Blattschutz	559
311	Ereignisbedingte Formatierung	561
312	Ereignisbedingte Formatierung durch Formelbezüge	563
313	Makro aus Hyperlink aufrufen	565
314	Adresse zu Hyperlink ermitteln	567
315	Autoprozeduren	568
316	Datei öffnen und Auto_Open-Prozedur aufrufen	569
317	Datei schließen und Auto_Close-Prozedur aufrufen	569
318	Begrüßung nach der Tageszeit	570
319	Zoomfaktor über Bildschirmauflösung	571
320	Zoomfaktor mit Workbook_Open	572
321	Zeitgesteuerte Makros	574
322	Uhrzeit in Statuszeile zeitbegrenzt anzeigen	575
323	Uhrzeit in Statuszeile steuern	578
324	Uhrzeitmeldung	579

325	Tastenkombinationen erstellen	579
326	Aufgabenbereich ein-/ausblenden	583
327	Navigation mit Fadenkreuz	585
328	Makro rückgängig machen	588

UserForm 591

329	Bezeichnungsfelder (Label)	594
330	Textfelder (TextBox)	594
331	Schaltflächen (CommandButton)	597
332	Rahmen (Frame)	599
333	Listenfelder (ListBox)	600
334	Mehrspaltige Listenfelder	603
335	Kombinationsfelder (ComboBox)	604
336	Werte von Kombinationsfeld nach Listenfeld übertragen	607
337	Optionsfelder (OptionButton)	608
338	Kontrollkästchen (CheckBox)	610
339	Umschaltfelder (ToggleButton)	612
340	Register (TabStrip)	613
341	Multiseiten (Multipage)	614
342	Bildlaufleisten (ScrollBar)	618
343	Drehfelder (SpinButton)	620
344	Anzeige (Image)	622
345	RefEdit	624
346	Position und Größe eines UserForm	626
347	UserForm dynamisch in der Größe verändern	627
348	UserForm aus UserForm aufrufen	628
349	Landkarte mit Quickinfo	631
350	Passworteingabe	633
351	Automatisches Ausrichten von Elementen	634
352	Tabellenblatt in UserForm einbinden	636
353	Diagramm in UserForm	643
354	Diagramm in UserForm mit Daten aus Tabelle	645
355	Mappenbaum in UserForm (TreeView)	647
356	Animierte Grafik in UserForm	649
357	Hyperlink in UserForm	650
358	Eine eigene Werkzeugsammlung zusammenstellen	651

Web/Mail 653

359	Eine einfache E-Mail versenden	653
360	Eine E-Mail mit HTML-Formatierungen	656
361	Ein Tabellenblatt als E-Mail versenden	657
362	Zellbereich per E-Mail versenden	659
363	Serien-Mail mit einheitlichem Inhalt	660
364	Serien-Mail mit unterschiedlichem Inhalt	661
365	Interne Hyperlinks	663
366	Hyperlink mit Quickinfo	664
367	Mittels Hyperlinks ein Inhaltsverzeichnis erstellen	665
368	Hyperlinks und Zelladressen (Suchen/Weitersuchen)	667
369	Ein neues Dokument erstellen	670
370	Externe Hyperlinks	671
371	Hyperlink auf lokale Datei	672

372	Hyperlinks zu einer Textmarke	672
373	Hyperlinks zu Zellen/Tabellen einer anderen Mappe	673
374	E-Mail-Adressen	674
375	Eine Schaltfläche mit Hyperlink	675
376	URL in Nebenzelle ausgeben	676
377	Hyperlinks löschen	676
378	Nur E-Mail-Links löschen	678
379	E-Mail-Links nicht löschen	678
380	Den HTML-Code einer Excel-Datei ansehen	679
381	Eine Webabfrage erstellen	681
382	Webabfragen aktualisieren	684
383	Eine Webabfrage bereinigen	685
384	Webabfrage aktualisieren und bereinigen (kombiniert)	686
385	Webabfragen löschen	687
386	Einen Webbrowser einbinden	688
387	Eine XML-Datei erzeugen	689
388	Import und Export von XML-Dateien	692
389	XML-SS-Schemas	696
390	XSL(T) Stylesheets für XML	698
391	Smart Tags	702
392	Ein eigenes Smart-Tag-Menü erstellen (XML)	703
393	Smart Tags löschen	706
Extern		709
394	Dateien löschen	710
395	Dateien umbenennen	711
396	Mit ActiveX ein Objekt erzeugen	714
397	Dateien verschieben	715
398	Dateien kopieren	717
399	Datei-Eigenschaften auslesen	718
400	Dateiattribute setzen	721
401	Datei- und Verzeichnisnamen auslesen	723
402	Dateien suchen	724
403	Dateien alphabetisch sortiert ausgeben	725
404	Dateien aus Unterverzeichnissen auslesen	726
405	Verzeichnis erstellen	727
406	Verzeichnis löschen	728
407	Verzeichnis wechseln	728
408	Größe eines Verzeichnisses ermitteln	728
409	Laufwerk oder Ordner?	730
410	Laufwerkbuchstaben ermitteln	731
411	Laufwerk-Eigenschaften	733
412	System-Informationen auslesen (Environ)	733
413	Das File System Object (FSO)	734
414	Eine Textdatei erzeugen	737
415	Eine Textdatei ergänzen	739
416	Eine Textdatei auslesen	740
417	Import aus Textdatei	743
418	Export nach Textdatei	748
419	Import aus Word	751
420	Export von Excel nach Word	754

421	Export von Zwischenablage nach Word	757
422	Import aus PowerPoint	758
423	Export nach PowerPoint	760
424	Outlook-Kontakte auslesen	762
425	Outlook-Kalender auslesen	765
426	Outlook-Posteingang auslesen	767
427	Outlook-Aufgaben auslesen	768
428	Outlook-Aufgaben-Fenster anzeigen	769
429	Outlook-Notiz erstellen	770
430	Access: Ein Query erstellen	771
431	Import aus einer Access-Tabelle	776
432	Export in eine Access-Tabelle	779
433	Export nach Zwischenablage	781
434	Import aus Zwischenablage	783

Gemischtes **785**

435	VBE – Entwicklungsumgebung	785
436	Verweise auslesen	786
437	Verweise aktivieren	788
438	Verweise deaktivieren	789
439	VBA-Projekteigenschaften auslesen	790
440	Projektname auslesen und ändern	791
441	Ist Projekt-Ansicht gesperrt?	792
442	Mit VBA Projektschutz aufheben	792
443	Entwicklungsumgebung ein- und ausblenden	794
444	Codemodul-Fenster auflisten	795
445	Codemodul-Fenster anzeigen	796
446	Text in Direktbereich löschen	797
447	Alle Projekte auslesen	798
448	Module und Prozeduren des aktiven VBA-Projektes auslesen	800
449	Code einer Prozedur auslesen	802
450	Blatt- und Codenamen einer Tabelle auslesen	803
451	Codenamen einer Tabelle ändern	805
452	Codenamen nach Tabellennamen benennen	808
453	Modul erstellen	808
454	Modul mit Namensvergabe erstellen	809
455	Modul löschen	810
456	Vorhandene Module überprüfen	811
457	Vorhandene Prozeduren überprüfen	812
458	Prozedur in ein Modul schreiben und aufrufen	813
459	Prozedur in eigenes Modul schreiben und aufrufen	814
460	Ereignis-Prozedur erstellen	814
461	Prozeduren löschen	816
462	Code eines Moduls löschen	817
463	Menü in VBE-Menüleiste erstellen	818
464	Ereignis-Prozeduren für integrierte Ereignisse erstellen	822
465	Meldungen über alle Zellpositions-Änderungen	827
466	Add-Ins	829
467	Add-Ins hinzufügen und installieren	831
468	Add-In von Diskette auf Festplatte kopieren	833
469	Add-Ins auflisten	834

470	Add-Ins über Explorer installieren	835
471	Add-In deinstallieren und aus Liste entfernen	837
472	Titel und Beschreibung einem Add-In hinzufügen	839
473	Code-Änderungen in Add-In speichern	840
474	Funktionen in Add-In einbinden	841
475	Makros aus Add-In aufrufen	843
476	Makros aus Add-In über Schaltfläche aufrufen	843

Specials 849

477	Ein kleines Puzzle	849
478	Die Farbpalette von Excel verändern	853
479	Einen Lottoschein ausfüllen (Array)	857
480	Formeln, Werte und Text eines Tabellenblattes ermitteln	861
481	Sprachtrainer	864
482	Tabelleblatt-Übersicht in Menüleiste	868
483	Präsentation in Excel	874
484	Auto-Berechnung aus Statuszeile in Zwischenablage einfügen	876
485	Menüeinträge mit Tastenkombinationen aufrufen	880

Teil III Anhang 887

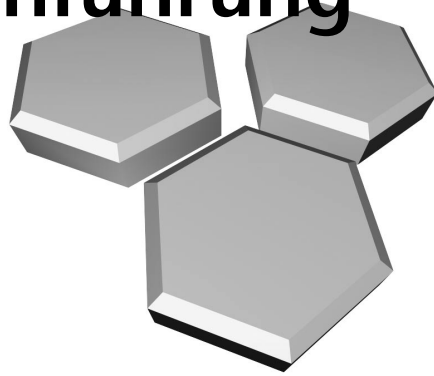
Anhang 889

1	Die englischen Funktionsnamen für die deutschen Tabellenblatffunktionen	889
2	VBA-Schlüsselwörter	892
3	MsgBox-Konstanten	897
4	Application.InputBox-Methode	898
5	Vergleichs-Operatoren	899
6	Arithmetische Operatoren	900
7	Verkettungs-Operatoren	902
8	SpecialCells-Methode	903
9	RGB-Funktion	904
10	ColorIndex-Eigenschaft	905
11	ColorFormat-Objekt (Fülleffekte)	905
12	FormatCondition-Objekt (Bedingte Formatierung)	906
13	Formatcodes für Kopf- und Fußzeilen	907
14	Datentypen	909
15	Typ-Umwandlungsfunktionen	910
16	Benutzerdefinierte Datums- und Zeitformate (Format-Funktion)	911
17	DateDiff-Funktion	913
18	OnKey-Methode	914
19	Attributes-Eigenschaft	916
20	Verschiedene Konstanten	916
21	ClipboardFormats-Eigenschaft	917
22	Spezifikationen und Einschränkungen von Excel	918
23	Liste der integrierten Dialogfeldargumente	922
24	Liste der integrierten Dialogfeldargumente für Diagramme	930
25	Liste der integrierten Dialogfeldargumente für Pivottabellen	933
26	Liste der integrierten Dialogfeldargumente für Solver	934
27	Liste der integrierten Dialogfeldargumente für Steuerelemente	934

Stichwortverzeichnis 937

TEIL I

Einführung



Vorwort

Seit Oktober 2003 ist der Nachfolger von Office XP, das *Office System 2003*, auf dem Markt erhältlich. Der Verlag Addison-Wesley hat deshalb beschlossen, dass es an der Zeit ist, die erste Auflage des Excel VBA Codebooks entsprechend anzupassen. Aufgrund eines Autorenwechsels war eine zweite Auflage im herkömmlichen Sinne nicht möglich und deshalb wurde das Buch von Grund auf neu geschrieben. Für alle Leser, die bereits im Besitz der ersten Auflage sind, bedeutet das, dass mit dem Erscheinen dieses Buchs quasi *Band 2* vorliegt. Ein Blick in das neue Buch wird sich somit durchaus lohnen. Änderungen sind bereits am Aufbau der Kategorien zu erkennen, denn diese wurden teils komplett ersetzt. Sie werden erneut unzählige Codebeispiele und Highlights in diesem Buch finden, die Ihnen den Alltag in der Programmierung erleichtern werden.

Sämtliche Beispiele in diesem Buch wurden auf die Excel-Versionen 2000, 2002 und 2003 abgestimmt. Nahezu alle hier enthaltenen Codes lassen sich problemlos in all diesen drei Versionen ausführen. Falls Versions-Unterschiede bestehen sollten, werden Sie an entsprechender Stelle darauf hingewiesen.

Als Zielgruppe für dieses Buch wurde hauptsächlich der fortgeschrittene Anwender, bis hin zum Profi gewählt. Das Buch ist kein Lehrbuch, sondern eine Art Lexikon. Es soll Ihnen als Nachschlagewerk dienen. Sie finden hier eine sehr umfangreiche Auswahl an Code-Elementen, die sich nach belieben zusammenfügen lassen. Wir haben zudem Wert darauf gelegt, neben umfangreichen Codes auch kurze Beispiele zu erstellen, so dass es auch einem Einsteiger möglich ist, sich besser mit dieser Programmiersprache vertraut zu machen. Für die erfahrenen VBA-Anwender sind viele komplexe Beispiele enthalten.

Da VBA durchaus nicht immer der einzige Weg ist, der zum Ziel führt, haben wir uns nach Möglichkeit darum bemüht, neben dem VBA-Code auch Lösungswege aufzuzeigen, die ohne Programmierung auskommen.

Danksagung von Monika Weber

Es war für mich eine große Herausforderung, das Codebook komplett neu zu schreiben. Mein erster Dank gilt meiner neuen Buchpartnerin Melanie Breden, die ich für dieses Projekt gewinnen konnte. Sie hat sich als kompetente Partnerin beim Erstellen des Buches erwiesen.

Ein weiteres Dankeschön möchte ich an unseren Lektor Frank Eller richten. Er hat mir mit diesem Werk bereits das dritte Projekt bei Addison-Wesley ermöglicht. Selbstverständlich gilt der Dank auch unserer Korrektorin Anja Schmitz, die dafür gesorgt hat, dass unsere Tippfehler aus den Buchseiten entfernt wurden.

Meinen US-MVP Kollegen Bob Umlas und John Walkenbach möchte ich für die Inspiration und Unterstützung in zwei sehr interessanten Beispielen (Array und Diagramme) danken.

Ein letztes und besonders herzliches Dankeschön geht an meine Liebsten und Familienangehörigen, die erneut das Verständnis dafür aufgebracht haben, dass meine Freizeit während der Dauer des Schreibens sehr eingeschränkt war.

Monika Weber, Schweiz, <http://www.jumper.ch>

Microsoft MVP für Excel

Danksagung von Melanie Breden

Mein Dank geht an unseren Lektor Frank Eller und meine Buchpartnerin Monika Weber. Das große Vertrauen, welches durch dieses herausfordernde Projekt an mich übertragen wurde, hat mich sehr geehrt. Dieser Aufgabe habe ich mich mit Freude gestellt und auch beim Erarbeiten der Kategorien und Beispiele neue Möglichkeiten und Lösungen der großen Excel-Welt ergründen können.

Ich bedanke mich vor allem bei meinem Mann Michael und meinen Kindern Juliane und Lukas. Sie haben sehr viel Verständnis für mich aufgebracht, wenn ich mal wieder sehr vertieft bei meiner Arbeit war und sie nicht die sonst gewohnte Aufmerksamkeit erhalten haben.

Ein weiterer Dank geht an meinen Schweizer MVP Kollegen Thomas Ramel. Ich danke ihm herzlich für den innovativen Gedankenaustausch und für das Testen umfangreicherer Programmierbeispiele.

Ein letzter Dank geht an die Benutzer der *microsoft.public.de.excel*-Newsgroup. Durch ihre Fragestellungen bin ich auf die alltäglichen Probleme mit VBA aufmerksam geworden. In der Hoffnung, auch Ihnen liebe Leser, das Leben mit VBA zu vereinfachen, sind einige meiner dort aufgezeigten Lösungswege in dieses Buch eingeflossen.

Melanie Breden, Deutschland

Microsoft MVP für Excel

Einführung

1 Hinweise zum Buch

Nachfolgend finden Sie einige Erläuterungen, die sich generell auf den Aufbau dieses Buches beziehen.

Aufgrund der besseren Lesbarkeit des Textes haben wir auf eine geschlechtliche Trennung verzichtet.

Die Schreibkonventionen

Das vorliegende Buch hat ein einheitliches Erscheinungsbild, das folgenden Schreibkonventionen unterliegt:

Element	Beispiel	Formatierung
Dateiname und Pfad	C:\Codebook\Test.xls	kursiv
Dateinamenserweiterungen	*.bmp	kursiv
Abkürzungen mit Erläuterung	OWC (Office Web Components)	kursiv
Menüpunkte und Benutzeroberfläche	EXTRAS OPTIONEN	Kapitalchen
Tasten und Tastenkombinationen	Strg C	Tasten
Code	Sub Modulname() ... End Sub	Courier

Tabelle 1: Schreibkonventionen

Die beiliegende CD

Auf der beiliegenden CD sind sämtliche im Buch abgedruckten Beispiele zu finden. Die CD ist in zwei Hauptordner unterteilt. Im Ordner *Buchdaten* sind, nach Kategorie getrennt, die kompletten Excel-Arbeitsmappen mit den zugehörigen Codebeispielen zu finden. Im Verzeichnis *Repository* finden Sie das Codebook-Repository, das es Ihnen ermöglicht, mit ein paar Mausklicks auf alle Codebeispiele des Buchs zuzugreifen und diese sofort in Ihre Anwendungen zu kopieren.

Damit Sie die Beispiele auf der CD leicht finden können, sind sämtliche abgedruckten Codes mit einem so genannten *Header* versehen. Im Header steht an erster Stelle der Pfad der Datei und an zweiter Stelle der Dateiname. Falls die Mappe mehrere Tabellenblätter umfasst, ist der Blattname, bzw. sind die Blattnamen, die zu dem Rezept gehören, aufgeführt. An letzter Stelle finden Sie den Namen des Moduls oder der Klasse der Ereignisprozedur.

Beispiel:

```
' =====  
' Auf CD      Buchdaten\Beispiele\07_Objekte\  
' Dateiname  07_01_Pictures.xls  
' Tabelle    07_Objects_RangePicture  
' Modul      mdl_01_PhotographPicture  
' =====
```

2 Die Geschichte von Excel

Am 2. Mai im Jahre 1985 wurde ein hervorragendes Tabellenkalkulationsprogramm durch Microsoft freigegeben. Es war eine Version für den Apple Macintosh. Jedoch noch am Tag vor der Einführung kam es bei Tests zu einem Crash. Glücklicherweise konnte der Fehler noch vor der Premiere behoben werden und es verlief alles glatt. Excel hat bis heute seinen Status als beliebteste und erfolgreichste Tabellenkalkulation halten können.

Im Jahre 1987 wurde Excel 2.0 erstmals auf einem Personal Computer mit dem Betriebssystem DOS 3.0 eingesetzt. Aber wo ist der wirkliche Beginn der elektronischen Tabellenkalkulation?

Die erste elektronische Tabellenkalkulation

Angefangen hat die Geschichte rund um die elektronische Tabellenkalkulation viel früher. Die Wurzeln reichen bis ins Jahre 1961 zurück. Professor Richard Mattessich hatte die Illusion ein computerisiertes Tabellenkalkulations-System zu entwerfen.

Sowohl seine Vorstellungen, als auch die von anderen Entwicklern von Tabellenkalkulationen, wurden von Dan Bricklin, einem damaligen Studenten der Harvard Business School, im Jahre 1978 aufgenommen. Zusammen mit Bob Frankston entwickelte er ein Programm mit dem Namen *VisiCalc*. Es war das erste wirkliche Tabellenkalkulations-Programm. Es wurde für den Apple II entwickelt und führte schließlich auch zu dessen Erfolg.

In den frühen 80er Jahren, mit der erfolgreichen Einführung des IBM PC, wuchs der Markt für elektronische Tabellenkalkulations-Programme sprunghaft an. Die Entwickler von *VisiCalc* waren zu langsam, um auf diesen fahrenden Zug aufzuspringen. Im Jahre 1983 wurde schließlich Lotus 1-2-3, zum damaligen Marktführer. Lotus 1-2-3 war das erste Tabellenkalkulationsprogramm, das Tabellen, Diagramme und Datenbankfunktionalitäten verarbeiten konnte.

Ein weiterer Pionier der 80er-Jahre war *Multiplan*, das von der Firma Microsoft entwickelt wurde. Um genau zu sein, wurde das Jahr 1982 geschrieben, als die erste Version von *Multiplan* im Handel erhältlich war.

Die ersten Makros tauchten auf

Legendär ist die Tatsache, dass im Jahre 1983 in Lotus 1-2-3 erstmals Makros verfügbar waren. Es war somit für Entwickler möglich, Makros zu schreiben und zu testen. Für Nicht-Programmierer war es möglich Makros aufzuzeichnen und damit auf sehr einfache Weise den Alltag in der Arbeit mit der Tabellekalkulation zu erleichtern und zu automatisieren. Es wird erzählt, dass diese Funktionalität erst im letzten Moment kurz vor der Veröffentlichung der definitiven Version eingebaut wurde. Zum Glück für Lotus 1-2-3, denn damit hob sich diese Software entscheidend von andern Programmen ab, was letztlich ein Grund für seinen Erfolg war.

Die ersten Makros in Excel

Die ersten Makros, die 1988 in Excel 2.1 geschrieben wurden, mussten in einem speziell dafür vorgesehenen Makro-Blatt erfasst werden. Dieses Makro-Blatt wurde getrennt in einer Datei mit der Endung **.xlm* abgespeichert. Die Excel-Arbeitsmappen hingegen trugen die Endung **.xls*, so wie das auch heute noch der Fall ist. Diese Makros wurden XLM-Makros oder Excel4-Makros genannt. So konnten sie später namentlich von den VBA-Codes der Excel-Version 5 unterschieden werden. Die Makro-Sprache von Excel war viel mächtiger als jene von Lotus 1-2-3. Excel stellte viele hundert Funktionen zur Programmierung zur Verfügung. Dem konnte Lotus 1-2-3 nicht gerecht werden.

Das ganze komplexe Makro-Konzept von Excel hatte Vor- und Nachteile. Erfahrene Programmierer, die mit der Macht dieses Instrumentes umgehen konnten, hießen es willkommen. Diese Vielfalt war jedoch gleichzeitig eine Barriere für die meisten Benutzer. Es gab keine einfache Verbindung zwischen der manuellen Benutzung von Excel und deren Programmierung. Für einen einfachen Excel-Anwender bedeutete es viel Lernaufwand, sich diese Programmiersprache anzueignen, wovon viele zurückschreckten. Ein weiterer Nachteil war die Tatsache, dass das Betriebssystem Windows erforderlich war, das wiederum für seinen hohen Ressourcenverschleiß bekannt war.

Der Glaube, dass OS/2 das Betriebssystem Windows letztendlich ablösen würde, war ein großer Fehler von Lotus. Die Programmierer planten nicht, ein Lotus 1-2-3 für Windows zu entwickeln. Schnell wurde deutlich, dass trotz allem Windows vor OS/2 das Rennen machen würde. Den ersten Versuch von vielen, doch noch auf die Windows-Schiene aufzuspringen, tat Lotus im Jahre 1991. Die neue Lotus1-2-3 Version war jedoch eher eine Entwicklung für DOS mit einer grafischen Oberfläche, als eine für Windows. Microsoft hatte mit Windows und seinen Office-Programmen mittlerweile die Vorherrschaft auf dem Markt erreicht. Es war zu spät um noch mit Excel mithalten zu können und mit dem Erfolg von Lotus 1-2-3 war es vorbei.

Excel 5

Im Jahre 1993 hat Microsoft den Schritt getan, den Programmcode der einzelnen Office-Applikationen zu vereinheitlichen. Die Programmiersprache VBA (*Visual Basic for Applications*) wurde erstmals in Excel 5 eingeführt und war in dieser Version sogar in Deutsch verfügbar. Die Programmiersprache änderte sich jedoch schon in Excel 7 (Verkaufsname *Excel 95*). Zwar wurden die deutschen Programmcodes noch unterstützt, es war jedoch abzusehen, dass diesbezüglich eine Vereinheitlichung aller Office-Applikationen in Richtung englischer Sprache zu erwarten war. Schrittweise wurde VBA auch in Access, Word, PowerPoint und Outlook eingeführt. Alle Microsoft Office Anwendungen verfügen heute über VBA.

In Excel 5 kamen etliche Neuerungen hinzu. Das Arbeitsmappenformat wurde neu konzipiert. Es konnten beliebig viele Tabellenblätter in eine Mappe eingefügt werden. Es war nun möglich, Textdateien zu importieren und Direkteingaben in Zellen vorzunehmen. Der Hilfeassistent wurde eingeführt und der Funktionsassistent überarbeitet. Ebenso konnten 3-D-Formeln und benannte Bereiche eingesetzt werden.

Seit der Version Excel 5 werden beide Makro-Sprachen, sowohl XLM also auch VBA unterstützt. Was die Zukunft bringen wird ist ungewiss. Fest steht jedoch, dass auch in Office 2003 beides noch verfügbar ist.

VBA ist eine objektorientierte Programmiersprache, die in seiner Struktur und der Behandlung der Objekte mit der Programmiersprache Visual Basic identisch ist. Wenn Sie einmal gelernt haben VBA in Excel zu programmieren, wird es für Sie einfach sein, diese Programmiersprache in anderen Office-Applikationen anzuwenden. Wichtig dabei ist, dass Sie mit dem Objekt-Modell der einzelnen Anwendungen vertraut sind. Die Hierarchie ist immer nach dem gleichen System aufgebaut.

Excel 7

Um eine Vereinheitlichung der Versionsnummer in den einzelnen Office-Programmen zu erzielen, wurde die Version Excel 6 ausgelassen. Im Jahre 1995 kam die Version Excel 7 auf den Markt. Seit dieser Version führt jede Excel-Version neben der Versionsnummer einen Verkaufsnamen. Excel 7 ist auch unter dem Namen Excel 95 bekannt. Des Weiteren gab es nur sehr wenige Änderungen. Es wurden nun lange Dateinamen unterstützt und Excel 95 war zudem die letzte Version,

welche die deutsche Makrosprache unterstützte. Englisch ist seit dieser Version die Standardsprache für VBA. Als Übersetzungshilfe wurde die Datei *VBALISTE.xls* zur Verfügung gestellt, die auch heute noch mitgeliefert wird. Microsoft aktualisiert diese jedoch seit einiger Zeit nicht mehr um neue Funktionen und Schlüsselwörter.

Excel 97

Im Jahre 1997 wurde mit Excel 8, Verkaufsname Excel 97, eine dramatisch veränderte Version auf den Markt gebracht. Neben der Einführung der bedingten Formatierung wurde das Arbeitsmappen-Dateiformat geändert. Ältere Excel-Versionen konnten diese Dateien nicht immer lesen. Die deutschsprachigen Anwender mussten sich damit abfinden, dass VBA nun nur noch in Englisch gehalten wurde. Es gab keine Möglichkeit mehr zwischen deutschem und englischem Programmcode zu wählen. VBA bediente sich zudem einer anderen Schnittstelle und am Objekt-Modell waren auch gewisse Änderungen festzustellen. Die Makros waren auf einmal nicht mehr im Vordergrund der Excel Applikation zu sehen. Sie wurden in den Hintergrund »verbannt«. Module und Tabellenblätter einer Mappe wurden in VBA Projekte zusammengefasst und konnten nur noch im Visual Basic Editor eingesehen und verändert werden, so wie das auch heute noch der Fall ist. Neben den Standard-Modulen wurden auch Klassen-Module integriert. Klassen-Module erlauben es, eigene Objekte und Ereignisse zu programmieren. Viele neue Ereignis-Prozeduren waren verfügbar und ActiveX-Controls hielten Einzug. Die Extensibility-Bibliothek wurde zur Verfügung gestellt. Diese ermöglicht es, die Entwicklungsumgebung VBE zu verändern und mittels eines VBA-Codes auf einen anderen VBA-Code zuzugreifen und diesen zu verändern.

Alles in allem war diese Version eine große Wende in der Excel-Geschichte.

Excel 2000

Im Jahre 1999 folgte Excel-Version 9. In dieser Version, mit dem Verkaufsnamen Excel 2000, blieben solch einschneidende Veränderungen, wie es in der Version 97 der Fall war, aus. Dennoch wurde Excel um einiges reicher. Es war erstmals möglich, Pivot-Diagramme zu erstellen. Ein weiterer großer Schritt wurde in Richtung Internet getan. Es war nun möglich, eine Excel-Tabelle als Webseite im Format *.htm abzuspeichern. Dies sogar mit Interaktivität. Das bedeutete, dass in einem Web-Dokument, das auf Excel basierte, gewisse Excel-Funktionen zur Verfügung standen. Das wiederum besagte, dass online Veränderungen an der Tabelle vorgenommen werden konnten. Um sich auszutauschen wurde die ONLINEZUSAMMENARBEIT im Menüpunkt EXTRAS eingeführt.

Für die Benutzer von VBA wurde den UserForms die Eigenschaft `modeless` hinzugefügt. Diese ermöglicht es, bei geöffnetem Dialogfenster Änderungen an der Tabelle vorzunehmen. Vor der Version 2000 war das nicht möglich. Es musste erst das Dialogfenster geschlossen werden, um das Tabellenblatt zu editieren, oder zwischen einzelnen Tabellenblättern oder Excel Dateien zu wechseln.

Excel 2002

Auch im Jahre 2001, mit Einführung der Version 10, hat sich erneut wenig Sensationelles, aber dennoch Interessantes, getan. Neben dem Verkaufsnamen Excel 2002 wurde diese Version auch unter dem Namen Excel XP bekannt. Wie so oft wurde offensichtlich viel Wert darauf gelegt, die Optik aufzupeppen. Microsoft ging zudem immer mehr den Weg in Richtung Web und hat diesbezüglich erneut einige Features eingebaut. Die Unterstützung von XML (Extensible Markup Language) wurde verbessert. Eine für Programmierer möglicherweise interessante Neuerung waren SmartTags, RTD (Real Time Data) und ein neues Objektmodell zum Thema Blattschutz.

SmartTags sind kleine Symbole auf dem Tabellenblatt, die nur beim Eintritt eines bestimmten Ereignisses erscheinen. So wird zum Beispiel ein SmartTag eingeblendet, wenn eine fehlerhafte Formel im Tabellenblatt enthalten ist. Bei Klick auf ein solches SmartTag öffnet sich ein Drop-Down-Feld mit einer Auswahl an Menüpunkten die speziell zur Fehlerbehebung der Formel gedacht ist. Es gibt noch eine ganze Reihe weiterer SmartTags.

Den Entwicklern war es nun möglich, mittels RTD, eine COM-Automatisierungsserver (*Component Object Model*) aufzurufen, um Daten in Echtzeit zu übertragen. In früheren Versionen wurde DDE (*Dynamic Data Exchange*) für diesen Zweck verwendet.

Zum Thema Blattschutz (EXTRAS | SCHUTZ | BLATTSCHUTZ) wurde einiges getan. Es ist nun eine ganze Liste an Objekten verfügbar, die wahlweise bei der Aktivierung des Blattschutzes eingestellt werden können. Es besteht dadurch die Möglichkeit, ein Blatt komplett nur gegen Veränderungen der Zellinhalte zu schützen. Änderungen an der Struktur und Format des Tabellenblattes, Spalten, Zeilen und Zellen können individuell eingestellt werden. In Bezug auf Formatierungen besteht die Möglichkeit, Tabellenreiter in unterschiedlichen Farben darzustellen.

Den Anwender erwarteten zudem Neuerungen in der Bedienerführung. Es war endlich möglich tabellenübergreifend zu suchen. Die Sortierfunktion wurde verbessert und ein Formelprüfer eingeführt. Kopf- und Fußzeilen sind besser bearbeitbar und Grafiken können darin eingebunden werden. In vorherigen Versionen musste dies umständlich über die Wiederholungszeilen improvisiert werden. Ebenso kann der Dateipfad nun über eine Schaltfläche hinzugefügt werden. Das war bislang nur mit einer VBA Prozedur möglich.

Excel 2003

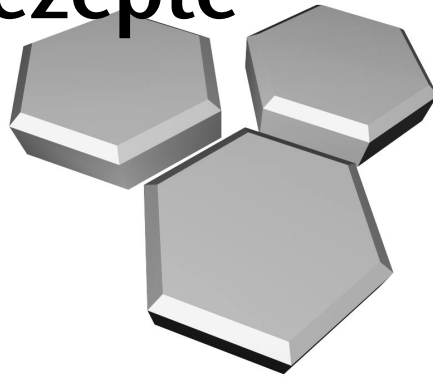
In Excel 11 lautet der Verkaufsname Excel 2003, der mit dem Jahr der Veröffentlichung wieder vereinheitlicht worden ist. Excel hat sich heute, in der neusten Version, entgegen aller Erwartungen, bis auf die Optik, die ja laufend etwas modernisiert wird, wenig verändert. Ein großer Fokus wird auf XML gelegt. Die Richtung ist auch weiterhin das Web.

Eine ganz nette Neuerung ist ein weiteres Objekt, das zur Verfügung steht. Es geht dabei um Listen, die automatisch erstellt werden können, bzw. die bei der Erstellung ihre Unterstützung anbieten. Ein Rechtsklick auf eine Zelle oder einen Bereich verrät, dass im Kontextmenü ein neuer Eintrag vorhanden ist. Der Menüpunkt nennt sich LISTE ERSTELLEN. Ein Klick auf den Menüpunkt DATEN zeigt den Eintrag LISTE. Beides führt zum selben Ergebnis. Die Liste, die dabei erstellt wird lässt sich sehen. Es wird beispielsweise automatisch ein Autofilter eingeblendet. Zudem wird eine eigene Symbolleiste angezeigt, die den Namen LISTE UND XML trägt. Per Klick beispielsweise auf die Schaltfläche XML lassen sich automatisch entsprechende Dateien importieren. Bei aktiviertem Autofilter wird als oberstes Auswahlkriterium die Sortierfunktion integriert.

Es sieht ganz so aus, als dürften frühestens in der nächsten Version von Excel wieder große Schritte nach Vorne zu erwarten bzw. zu erhoffen sein.

TEIL II

Rezepte



In dieser zweiten Auflage des Codebooks haben wir darauf verzichtet, erneut ausführlich über die Grundlagen von VBA zu schreiben. Sie werden in dieser Kategorie eine Anzahl an einfachen VBA-Codebeispielen, Tipps und Antworten auf häufig gestellte Fragen (FAQs – Frequently Asked Questions) finden, die auch für weniger erfahrene VBA-Programmierer leicht verständlich sind. Da sich das Buch an bereits erfahrene VBA-Anwender richtet, haben wir auf eine Beschreibung des Umganges mit dem VBA-Editor verzichtet und uns hauptsächlich auf Codebeispiele konzentriert.

1 Sicherheitseinstellungen in Excel

Excel-Arbeitsmappen, welche keinen VBA-Programmiercode enthalten, werden ohne irgendwelche Rückfragen anstandslos geöffnet.

Enthält eine Excel-Datei Makro-Code, entscheidet die eingestellte Sicherheitsstufe unter EXTRAS | MAKRO | SICHERHEIT, wie Excel sich beim Öffnen einer Datei verhält. Schon durch das Vorhandensein eines allgemeinen Moduls ohne enthaltenen Code wird eine Makrowarnung ausgegeben.



Abbildung 1: Sicherheitseinstellungen vornehmen

Wurde die Sicherheitseinstellung auf HOCH gesetzt, werden nicht signierte Makros automatisch deaktiviert und die Datei ohne Makrounterstützung geöffnet.

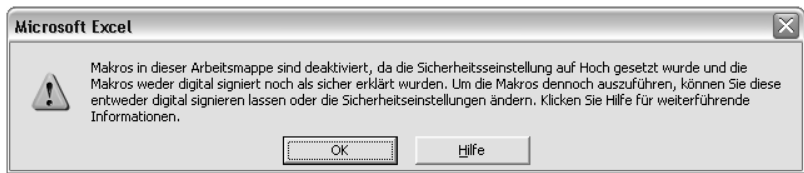


Abbildung 2: Makros wurden deaktiviert

Mit der Option NIEDRIG werden alle Excel-Dateien ohne Rückfrage geöffnet. Der enthaltene Makro-Code ist voll funktionsfähig. Diese Einstellung ist nicht zu empfehlen. Wollen Sie zum Beispiel fremde Excel-Dateien öffnen, bei denen Sie keine Makros vermuten, würde Excel Sie auch nicht vor möglichen Makro-Viren warnen.

Wählen Sie die Sicherheitseinstellung MITTEL, dann wird Excel bei vorhandenen Makros vor Dateioffnung einen Dialog anzeigen. Hier können Sie auswählen, ob die Datei mit oder ohne Makrounterstützung geöffnet werden soll.

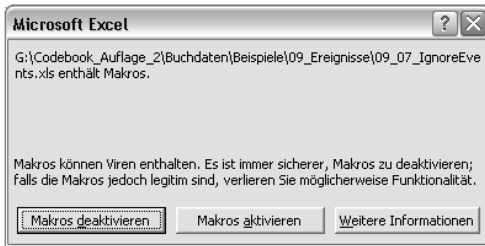


Abbildung 3: Makroabfrage vor Dateioffnung

Hinweis

Es ist mit VBA nicht möglich, die Sicherheitseinstellungen zu ändern oder auszulesen.

2 Zellen selektieren oder nicht?

Ein oft diskutiertes Thema ist das Selektieren von Zellen mittels VBA. Viele Einsteiger, die ihre ersten Schritte in VBA tun, verwenden anfangs häufig den Befehl `Select`. Das ist sicherlich unter anderem darauf zurückzuführen, dass der Makro-Rekorder diesen Befehl beim Aufzeichnen von Befehlsabläufen oft verwendet. Bei erfahrenen Programmierern ist der Befehl jedoch regelrecht verpönt. Warum ist der Befehl so ungern gesehen? Er geht zu Lasten der Geschwindigkeit. Wenn Zellen während des Ausführens eines Codes jedes Mal selektiert werden, ist das Programm viel langsamer als ohne ein Markieren der Zellen bzw. Tabellenblätter. In den meisten Fällen kann problemlos auf das `Select` verzichtet werden. Natürlich bestätigt auch hier die Ausnahme die Regel. Um nur ein Beispiel zu nennen: Wenn Sie sicherstellen möchten, dass nach Ablauf eines Codes die Zelle A1 des ersten Tabellenblattes aktiv sein soll, ist ein `Select` erforderlich.

Tipp

Falls in Ihrem Code `Selects` vorkommen, kann es sein, dass ein Bildschirm-Flackern auftritt, während die Prozedur ausgeführt wird. Das Flackern können Sie unterdrücken, indem Sie zu Beginn der Prozedur den Befehl `Application.ScreenUpdating = False` verwenden. Wichtig dabei ist, dass Sie die Einstellung am Ende wieder aktivieren, indem Sie den Wert auf `True` setzen: `Application.ScreenUpdating = True`.

Wenn Sie mit dem Makro-Rekorder Befehlsfolgen aufzeichnen, ist es sehr wichtig, dass dieser Code danach ordentlich bereinigt wird. An dieser Stelle möchten wir Ihnen zeigen, wie ein solcher Code bereinigt werden kann. Zuerst werden wir ein Makro aufzeichnen und diesen Code dann anpassen, so dass kein `Select` mehr darin vorkommt. Natürlich könnte man alleine zum Thema »Bereinigen von Codes« viele Buchseiten schreiben. Sie werden jedoch auf den folgenden Buchseiten eine Fülle an Beispielen finden, die Sie jederzeit als geeignete Vorlage für verschiedenste Anwendungen einsetzen können. Nur in den seltensten Fällen wird dabei ein `Select` verwendet. Unser erstes Codebeispiel wurde durch den Makro-Rekorder aufgezeichnet. Folgende Schritte wurden während der Aufzeichnung vorgenommen:

1. Die Zelle A1 wurde selektiert und es wurde eine gelbe Hintergrundfarbe zugewiesen
2. Die Zelle B2 wurde selektiert und es wurde eine rote Hintergrundfarbe zugewiesen.

- 3. Die Zelle C3 wurde selektiert und es wurde eine grüne Hintergrundfarbe zugewiesen.
- 4. Die Zellen D4:D6 wurden selektiert und es wurde eine blaue Hintergrundfarbe zugewiesen.

Ein Wechsel in die VBA-Umgebung (**Alt** + **F11**) zeigt im entsprechenden Modul folgenden Code an, der aufgezeichnet wurde:

```
'=====
' Auf CD      Buchdaten\Beispiele\01_Einstieg\
' Dateiname   01_01_FirstSteps.xls
' Tabelle     Tabelle1
' Modul       mdl_01_SelectCells
'=====

Sub Makro1()
    Range("A1").Select
    With Selection.Interior
        .ColorIndex = 6
        .Pattern = xlSolid
    End With
    Range("B2").Select
    With Selection.Interior
        .ColorIndex = 3
        .Pattern = xlSolid
    End With
    Range("C3").Select
    With Selection.Interior
        .ColorIndex = 4
        .Pattern = xlSolid
    End With
    Range("D4:D6").Select
    With Selection.Interior
        .ColorIndex = 5
        .Pattern = xlSolid
    End With
End Sub
```

Der obige Code verwendet für jeden Bereich, der während der Aufzeichnung selektiert wurde, die `Select`-Methode. Zudem wurde neben der Eigenschaft `ColorIndex`, welche für die Farbe steht, auch die Eigenschaft `Pattern` aufgezeichnet. Die Eigenschaft `Pattern` wird verwendet, um das Hintergrund-Muster festzulegen, z.B. ein schraffierter Hintergrund. Die Codezeilen mit diesen Informationen können gänzlich weggelassen werden, denn das Argument `xlSolid` hinter der Eigenschaft `Pattern` ist die Standardeinstellung. Standardeinstellungen müssen nicht explizit angegeben werden. Damit erübrigt sich auch das Verwenden der `With`-Anweisung, denn der Einsatz einer `With`-Anweisung macht nur dann Sinn, wenn damit mehrere Befehle für ein Objekt zusammengefasst werden können. In diesem Fall gehören sowohl `ColorIndex` als auch `Pattern` zum Objekt `Interior`. Da die `Pattern`-Eigenschaft jedoch wegfällt, ist auch die Zusammenfassung `With` überflüssig.

Nachdem also das `Pattern` und das `With` entfernt wurden, sieht der Code wie folgt aus:

```
'=====
' Auf CD      Buchdaten\Beispiele\01_Einstieg\
' Dateiname   01_01_FirstSteps.xls
```

Grund-lagen

Allge-mein

Datu Zeit

tungen

Steuer elemen

Befehl leisten

Obj ekte

Dia-gramm

Ereig-nisse

User-Form

Web/ Mail

Extern

Ge-misch tes

Specia

```
' Tabelle      Tabelle1
' Modul      mdl_01_SelectCells
' =====

Sub Makro2()
    Range("A1").Select
    Selection.Interior.ColorIndex = 6
    Range("B2").Select
    Selection.Interior.ColorIndex = 3
    Range("C3").Select
    Selection.Interior.ColorIndex = 4
    Range("D4:D6").Select
    Selection.Interior.ColorIndex = 5
End Sub
```

Überflüssig sind immer noch die `Select`-Methode und die dazugehörige `Selection`-Eigenschaft. Beides kann gänzlich weggelassen werden, denn die Hintergrundfarbe kann auch ohne ein Selektieren direkt dem entsprechenden Bereich zugewiesen werden. Die beiden Codezeilen, diejenige, die mit `Select` endet, und diejenige, welche mit `Selection` beginnt, werden zusammengesetzt. Der komplett bereinigte Code kann demnach aus lediglich vier Codezeilen bestehen.

```
' =====
' Auf CD      Buchdaten\Beispiele\01_Einstieg\
' Dateiname   01_01_FirstSteps.xls
' Tabelle     Tabelle1
' Modul      mdl_01_SelectCells
' =====

Sub Makro3()
    Range("A1").Interior.ColorIndex = 6
    Range("B2").Interior.ColorIndex = 3
    Range("C3").Interior.ColorIndex = 4
    Range("D4:D6").Interior.ColorIndex = 5
End Sub
```

Tipp

Falls Sie unsicher sind, ob eine aufgezeichnete Codezeile für Ihren Zweck tatsächlich erforderlich ist oder nicht, dann empfehlen wir Ihnen, diese nicht gleich zu löschen, sondern erst auszukommentieren. Das heißt: Sie setzen ein Hochkomma (') vor die betreffende Codezeile. Die Zeile wird damit zum Kommentar und wird nicht als Bestandteil des ausführbaren Codes betrachtet. Platzieren Sie den Cursor innerhalb der Prozedur und führen Sie den Code erneut aus F5. Falls Sie nun feststellen sollten, dass die Codezeile doch erforderlich ist, weil die Prozedur nicht mehr richtig funktioniert, können Sie lediglich das Hochkomma zu Beginn der Zeile entfernen. Sie können sich damit viel Arbeit und Ärger sparen. Denn wenn Sie die Codezeile gelöscht hätten, wäre es nun unter Umständen notwendig, den gesamten Code erneut aufzuzeichnen, um die entsprechenden Befehle wieder zu erhalten.

Ein Beispiel zum Thema »Kommentieren« finden Sie im Rezept 4.

3 Tabellenblätter selektieren oder nicht?

Bei den Tabellenblättern verhält es sich genauso, wie bei den Zellen, welche im vorangegangenen Rezept behandelt wurden. Auf die `Select`-Methode kann in der Regel verzichtet werden.

Auch hier zeichnen wir erst den Code mit dem Makro-Rekorder auf und bereinigen danach den Code. Es werden folgende Schritte ausgeführt:

- 1. Ausgehend vom ersten Tabellenblatt wird der Bereich A20:B21 selektiert.
- 2. Wir vergeben die Hintergrundfarbe Magenta.
- 3. Das zweite Tabellenblatt wird selektiert.
- 4. Der Bereich A20:B21 des nun aktiven Tabellenblattes wird selektiert.
- 5. Wir vergeben die Hintergrundfarbe Cyan.

Ein Blick in das Modul, in welchem sich das aufgezeichnete Makro befindet, gibt folgende Code-Zeilen preis:

```
'=====
' Auf CD      Buchdaten\Beispiele\01_Einstieg\
' Dateiname  01_01_FirstSteps.xls
' Tabelle     Tabelle1 und Tabelle2
' Modul      mdl_02_SelectSheets
'=====

Sub Makro4()
    Range("A20:B21").Select
    With Selection.Interior
        .ColorIndex = 7
        .Pattern = xlSolid
    End With
    Sheets("Tabelle2").Select
    Range("A20:B21").Select
    With Selection.Interior
        .ColorIndex = 8
        .Pattern = xlSolid
    End With
End Sub
```

Auch hier beginnen wir erst mal damit, das `Pattern` und `With` zu entfernen, genauso wie im vorangegangenen Rezept beschrieben. Danach können alle `Select`-Befehle entfernt werden. Die Codezeilen werden zusammengefügt, so dass am Ende nur noch zwei Zeilen übrig bleiben.

Der ersten Codezeile haben wir den Namen des Tabellenblattes `Worksheets("Tabelle1")` vorangestellt, welches bei der Aufzeichnung fehlte, da die Aufzeichnung ausgehend von der `Tabelle1` gestartet wurde. Die Aufzeichnung hat zudem lediglich den Ausdruck `Sheets` anstatt `Worksheets` verwendet. Das ist natürlich nicht verkehrt. Der Ausdruck `Sheets` steht stellvertretend sowohl für Tabellen- als auch für Diagrammblätter. Das Objekt `Worksheets` spricht jedoch ausdrücklich Tabellenblätter an.

Es spielt nun keine Rolle mehr, welches Tabellenblatt beim Ausführen des Codes aktiv ist. Es wird immer der angegebene Bereich der genannten Tabellenblätter entsprechend eingefärbt.

Grundlagen

Allgemein

Datum
Zeit

ungen

Steuerelemente

Befehle
leisten

Objekte

Diagramm

Ereignisse

User-Form

Web/Mail

Extern

Gemischtes

Spezial

```
' =====
' Auf CD      Buchdaten\Beispiele\01_Einstieg\
' Dateiname  01_01_FirstSteps.xls
' Tabelle    Tabelle1 und Tabelle2
' Modul      mdl_02_SelectSheets
' =====

Sub Makro5()
    Worksheets("Tabelle1").Range("A20:B21").Interior.ColorIndex = 7
    Worksheets("Tabelle2").Range("A20:B21").Interior.ColorIndex = 8
End Sub
```

4 Codezeilen kommentieren

Oftmals werden an Stelle von sprechenden Namen Index-Nummern verwendet. Beispielsweise ist das beim Zuweisen von Farben üblich, denn nicht für alle 56 verfügbaren Farben gibt es für sich sprechende Konstanten. Die verfügbaren Konstanten können Sie der Tabelle 2 entnehmen.

Damit beim Lesen des Codes deutlich zu erkennen ist, worum es sich handelt, kann ergänzend ein Kommentar eingefügt werden. Kommentare sind, wie bereits weiter vorne erwähnt, daran zu erkennen, dass ihnen ein Hochkomma (') vorangeht und die Schriftfarbe eine andere ist. Sie können entweder eine eigenständige Zeile als Kommentarzeile verwenden, oder hinter einer Codezeile einen Kommentar ergänzen. In unserem Beispiel werden die Kommentare im Schriftstil *Kursiv* dargestellt. Im VBA-Editor werden die Zeilen grün geschrieben, vorausgesetzt, Sie haben die Farbeinstellungen Ihrer VBA-Umgebung nicht verändert (Menü: EXTRAS | OPTIONEN, Registerkarte: EDITIERFORMAT).

```
' =====
' Auf CD      Buchdaten\Beispiele\01_Einstieg\
' Dateiname  01_01_FirstSteps.xls
' Tabelle    Tabelle1
' Modul      mdl_03_CodeComments
' =====

Sub CodeComments()
    ' Der Farbindex 6 steht für Gelb
    Range("A1").Interior.ColorIndex = 6
    Range("B2").Interior.ColorIndex = 3    ' Rot
    Range("C3").Interior.ColorIndex = 4    ' Grün
    Range("D4:D6").Interior.ColorIndex = 5 ' Blau
End Sub
```

Der folgenden Tabelle können Sie die verfügbaren Farbkonstanten entnehmen. Das Rezept 16 wird Ihnen zeigen, wie Sie per VBA eine Farbtabelle mit Farben und Index erstellen können.

Farbe	Farbindex	Farbkonstante
Schwarz	0	vbBlack
Blau	5	vbBlue
Cyan	8	vbCyan

Tabelle 2: Verfügbare Farbkonstanten

Farbe	Farbindex	Farbkonstante
Grün	4	vbGreen
Magenta	7	vbMagenta
Rot	3	vbRed
Weiß	2	vbWhite
Gelb	6	vbYellow

Tabelle 2: Verfügbare Farbkonstanten (Forts.)

5 Codezeilen umbrechen

Oftmals kann es vorkommen, dass Codezeilen recht lang werden. Der VBA-Editor lässt das zwar zu, das Ganze kann dadurch jedoch leicht unübersichtlich werden. Wenn immer von links nach rechts gescrollt werden muss, ist das unter Umständen etwas mühsam. Sie haben deshalb die Möglichkeit, Codezeilen zu umbrechen. Dazu wird jeweils ein Leerzeichen, gefolgt von einem Unterstrich »_«, eingefügt.

Falls es sich beispielsweise um Text handelt, der in einem Nachrichtenfenster `MsgBox` angezeigt wird, muss jede Zeile sauber abgeschlossen und neu begonnen werden. Das heißt, vor dem Umbruch muss der Text mit Schlusszeichen (") beendet werden. Die Zeilen werden zudem durch ein kaufmännisches Und »&« miteinander verbunden.

Entsprechende Umbrüche können demnach wie folgt aussehen – achten Sie auf die Unterstriche:

```
'=====
' Auf CD      Buchdaten\Beispiele\01_Einstieg\
' Dateiname  01_01_FirstSteps.xls
' Tabelle    Tabelle1
' Modul      mdl_04_CodeBreak
'=====

Sub CodeBreak()
    MsgBox "Es wird nun im zweiten Tabellenblatt " & _
           "dem Bereich ""A1:B5"" eine rote " & _
           "Hintergrundfarbe zugewiesen."

    Worksheets("Tabelle2"). _
        Range("A1:B5"). _
        Interior.ColorIndex = 3
End Sub
```

6 Die letzte (nicht) belegte Zelle ermitteln

Eine häufig gestellte Frage ist, wie man zur letzten belegten Zelle einer bestimmten Spalte gelangen kann. Das Ganze kann in einer einzelnen Codezeile gelöst werden. In unserem Beispiel wird dazu das Objekt `Cells` verwendet. Innerhalb von `Cells` wird an erster Stelle der Index der Zeile angegeben, nach dem Komma folgt der Index der Spalte. Hier die 1, die für die erste Spalte, also für die Spalte A steht.

Grundlagen

Allgemein

Datum
Zeit

tungen

Steuer
elemente

Befehle
leisten

Objekte

Diagramm

Ereignisse

User-
Form

Web/
Mail

Extern

Gemischtes

Spezial

Es wird in der Klammer von `Cells` an erster Stelle `Rows.Count` verwendet. Damit wird die Anzahl der insgesamt verfügbaren Zeilen gezählt. Die Zahl ist somit 65536. Anstelle von `Rows.Count` könnte natürlich auch die Zahl 65536 eingesetzt werden. Die Anweisung `End(xlUp)` veranlasst, dass von unten her die erste belegte Zelle selektiert wird.

```
'=====
' Auf CD      Buchdaten\Beispiele\01_Einstieg\
' Dateiname   01_01_FirstSteps.xls
' Tabelle     Tabelle1
' Modul       mdl_05_LastCell
'=====

Sub LastUsedCell()
    Cells(Rows.Count, 1).End(xlUp).Select
End Sub
```

Falls Sie die erste leere Zelle einer *Spalte* ermitteln möchten, wird die Eigenschaft `Offset` ergänzt. In Klammer (bei `Offset`) wird an erster Stelle eine 1 eingetragen. Das bedeutet, dass die *erste Zelle nach* dem belegten Bereich angesprochen wird. Für den Spaltenindex, welcher nach dem Komma folgt, wird der Wert 0 eingesetzt. Das bedeutet, dass es sich um dieselbe Spalte handelt, die in `Cells` eingetragen wurde. Um in der nächsten leeren Zelle der Spalte A einen Text einzutragen, kann auf ein `Select` verzichtet werden. Betrachten Sie dazu das folgende Beispiel:

```
'=====
' Auf CD      Buchdaten\Beispiele\01_Einstieg\
' Dateiname   01_01_FirstSteps.xls
' Tabelle     Tabelle1
' Modul       mdl_05_LastCell
'=====

Sub LastNotUsedCellRow()
    Cells(Rows.Count, 1).End(xlUp).Offset(1, 0).Value = "Codebook"
End Sub
```

Falls Sie die Spalte der letzten belegten Zelle einer *Zeile* ermitteln möchten, müssen die Werte entsprechend angepasst werden. In den runden Klammern nach der Eigenschaft `Cells` wird an erster Stelle angegeben, in welcher Zeile die erste nicht leere Zelle ermittelt werden soll. In unserem Beispiel handelt es sich um die erste Zeile 1. An zweiter Stelle, also nach dem Komma, wird die Anzahl der Spalten ermittelt. Alternativ zu `Columns.Count` kann auch der Wert 256 eingesetzt werden, welcher der Anzahl Spalten in einem Tabellenblatt entspricht. Die Richtung, die in runden Klammern nach der `End`-Eigenschaft eingetragen wird, lautet bei Zeilen `xlToLeft`.

```
'=====
' Auf CD      Buchdaten\Beispiele\01_Einstieg\
' Dateiname   01_01_FirstSteps.xls
' Tabelle     Tabelle1
' Modul       mdl_05_LastCell
'=====
```

```
Sub LastNotUsedCellColumn()  
    Cells(1, Columns.Count).End(xlToLeft).Offset(0, 1).Value = _  
        "Codebook"  
End Sub
```

Grundlagen

Allgemein

Datum
Zeit

Formeln

Steuerungselemente

Befehle
leisten

Objekte

Diagramm

Ereignisse

User-Form

Web/Mail

Extern

Gemischtes

Spezial

7 Löschen von Zellen, Zellinhalt, Spalten und Zeilen

Um Zellen, Spalten oder Zeilen zu löschen, verwenden Sie die Methode `Delete` in Kombination mit dem gewünschten Bereich.

Das folgende Beispiel zeigt in der ersten Codezeile, wie eine einzelne Zelle gelöscht werden kann. In der zweiten Codezeile wird ein Bereich gelöscht. In der dritten Codezeile wird die Spalte J gelöscht und in der letzten Codezeile die Zeile 20.

```
'=====
```

```
' Auf CD      Buchdaten\Beispiele\01_Einstieg\  
' Dateiname  01_02_DeleteCutPaste.xls  
' Modul      mdl_01_Delete  
'=====
```

```
Sub Delete()  
    Range("A1").Delete  
    Range("A2:B5").Delete  
    Columns("J").Delete  
    Rows(20).Delete  
End Sub
```

Achtung

Denken Sie beim Löschen und auch generell beim Ausführen von VBA-Codes daran, dass Sie VBA-Prozeduren nicht rückgängig machen können, außer Sie schreiben eigens dazu eine Prozedur. In der Kategorie »Ereignisse« finden Sie ein entsprechendes Rezept.

In Excel können für gewöhnlich bis zur Speicherung eine Anzahl an Schritten rückgängig gemacht oder wiederhergestellt werden. Mit dem Ausführen eines VBA-Codes werden beide Speicher gelöscht.

Beim Löschen von Zellen kann wahlweise angegeben werden, ob die Zellen nach links oder nach oben gelöscht werden sollen. Nach links ist die Standardeinstellung. Es kann somit auf die Angabe einer Konstanten verzichtet werden. Wenn die Zellen jedoch nach oben gelöscht werden sollen, dann ist die Angabe der Konstante `xlUp` erforderlich.

```
'=====
```

```
' Auf CD      Buchdaten\Beispiele\01_Einstieg\  
' Dateiname  01_02_DeleteCutPaste  
' Modul      mdl_01_Delete  
'=====
```

```
Sub DeleteCells()  
    Range("A1:B5").Delete Shift:=xlToLeft  
    Range("A1:B5").Delete Shift:=xlUp  
End Sub
```

Falls nur der Zelleninhalt gelöscht werden soll, verwenden Sie anstelle der `Delete`- die `ClearContents`-Methode.

8 Ausschneiden, kopieren und einfügen von Zellinhalt

Sie haben mittels VBA die Möglichkeit, Zelleninhalt auszuschneiden (`Cut`) oder zu kopieren (`Copy`) und an einer anderen Stelle wieder einzufügen (`Paste`).

In unserem Beispiel wird in der ersten Codezeile der Bereich `A1:B5` des ersten Tabellenblattes ausgeschnitten und im zweiten Tabellenblatt ab Zelle `A1` wieder eingefügt.

In der zweiten Codezeile geschieht dasselbe mit einem anderen Bereich. Diesmal wird der Zelleninhalt jedoch nicht ausgeschnitten, sondern kopiert.

```
'=====
' Auf CD      Buchdaten\Beispiele\01_Einstieg\
' Dateiname   01_02_DeleteCutPaste.xls
' Modul       mdl_02_CutCopyPaste
'=====
```

```
Sub CutCopyPaste()
    Worksheets(1).Range("A1:B5").Cut _
        Destination:=Worksheets(2).Range("A1")

    Worksheets(1).Range("A21:B25").Copy _
        Destination:=Worksheets(2).Range("A21")
End Sub
```

9 Löschen von Tabellenblättern

Beim Löschen von Tabellenblättern ist zu bedenken, dass immer ein Tabellenblatt vorhanden sein muss. Sie können Tabellenblätter entweder über deren Namen oder Index ansprechen. Wenn Sie, unabhängig vom Blattnamen, das fünfte Tabellenblatt löschen möchten, dann verwenden Sie die Angabe des Indexes `Worksheets(5).Delete`. Wenn Sie explizit das Tabellenblatt mit dem Namen `TABELLE5` löschen möchten, dann geben Sie in der runden Klammer den Blattnamen an `Worksheets("Tabelle5").Delete`.

Falls Sie nur diese eine Codezeile verwenden würden und kein Tabellenblatt mit dem Namen existieren würde, würde der Debugger gestartet. Um dies zu vermeiden, sollte erst mittels einer `If`-Entscheidung geprüft werden, ob ein solches Blatt vorhanden ist. Damit jedes Tabellenblatt auf den Namen geprüft wird, verwenden wir eine entsprechende `For`-Schleife. Falls das Blatt gefunden wird, erfolgt erst eine Rückfrage, ob wirklich gelöscht werden soll. Falls kein Blatt mit diesem Namen existiert, wird der Code fehlerfrei ausgeführt, ohne eine Löschung vorzunehmen.

Bevor ein Tabellenblatt gelöscht wird, löst Excel eine Warnmeldung aus, die rückfragt, ob das Tabellenblatt wirklich gelöscht werden soll. Falls Sie diese Meldung unterdrücken wollen, fügen Sie vor der Codezeile, die das Löschen veranlasst, die Anweisung `Application.DisplayAlerts = False` ein. Sie unterdrückt alle Warnmeldungen der gesamten Excel-Anwendung. Von daher ist es besonders wichtig, dass direkt nach der `Delete`-Methode die `DisplayAlerts`-Eigenschaft wieder auf `True` gesetzt wird.

```
'=====
' Auf CD      Buchdaten\Beispiele\01_Einstieg\
' Dateiname  01_02_DeleteCutPaste.xls
' Modul      mdl_03_DeleteSheet
'=====

Sub DeleteSheetName()
    Dim ws As Worksheet
    For Each ws In Worksheets
        If ws.Name = "Tabelle5" Then
            ws.Delete
        End If
    Next ws
End Sub
```

Wenn Sie mit dem Index des Blattnamens arbeiten, dann können Sie auf eine Schleife verzichten. Sie müssen lediglich prüfen, ob in der Mappe mehr als vier Tabellenblätter vorhanden sind. Wenn dies zutrifft, wird das fünfte Tabellenblatt gelöscht. Wenn nicht, wird die Prozedur dennoch fehlerfrei ausgeführt.

```
'=====
' Auf CD      Buchdaten\Beispiele\01_Einstieg\
' Dateiname  01_02_DeleteCutPaste.xls
' Modul      mdl_03_DeleteSheet
'=====

Sub DeleteSheetIndex()
    If Worksheets.Count > 4 Then
        Worksheets(5).Delete
    End If
End Sub
```

10 Einfügen von Tabellenblättern

Um ein Tabellenblatt in eine Mappe einzufügen, wird die Methode `Add` verwendet. Ohne weitere Angaben wird das Tabellenblatt vor dem aktiven Tabellenblatt eingefügt. Wenn Sie das Tabellenblatt jedoch beispielsweise an erster Stelle einfügen möchten, dann ergänzen Sie `Before:=Worksheets(1)`. Wenn das Tabellenblatt an letzter Stelle eingefügt werden soll, ergänzen Sie `After:=Worksheets(Worksheets.Count)`. In der runden Klammer wird ermittelt, wie viele Tabellenblätter bereits in der Mappe vorhanden sind. Nach dem letzten Tabellenblatt wird das neue Tabellenblatt eingefügt.

Hinweis

Um ein Diagrammblatt zu löschen, verwenden Sie anstelle von `Worksheets` das Objekt `Charts` oder `Sheets`.
Um ein Blatt zu kopieren, verwenden Sie an Stelle der Methode `Add` die Methode `Copy`.

```
'=====
' Auf CD      Buchdaten\Beispiele\01_Einstieg\
' Dateiname  01_02_DeleteCutPaste.xls
```

Grundlagen

Allgemein

Datum
Zeit

Formeln

Steuerelemente

Befehle

Objekte

Diagramm

Ereignisse

User-Form

Web/Mail

Extern

Gemischtes

Spezial

```
' Modul      mdl_04_InsertSheet
' =====

Sub InsertSheet()
    Worksheets.Add
    Worksheets.Add Before:=Worksheets(1)
    Worksheets.Add After:=Worksheets(Worksheets.Count)
End Sub
```

11 Arbeiten mit Kopf- und Fußzeilen

Falls Sie Kopf- und Fußzeilen per VBA in Ihre Tabelle einfügen möchten, zeichnen Sie den Vorgang mit dem Makro-Rekorder auf und bereinigen anschließend den Code. Um Kopf- oder Fußzeilen manuell einzufügen, wählen Sie den Menüpunkt ANSICHT | KOPF- UND FUSSZEILEN.

In VBA werden verschiedene Platzhalter verwendet, um vordefinierte Felder in Kopf- und Fußzeilen einzufügen. Die möglichen Felder können Sie der Tabelle 3 entnehmen.

```
' =====
' Auf CD      Buchdaten\Beispiele\01_Einstieg\
' Dateiname   01_03_HeaderFooter.xls
' Modul       mdl_01_HeaderFooter
' =====

Sub HeaderAndFooter()
    With ActiveSheet.PageSetup
        .LeftHeader = "Monika Weber und Melanie Breden"
        .CenterHeader = "&"&"Courier,Kursiv"&"Excel-VBA"
        .RightHeader = "&12Codebook"
        .LeftFooter = "&P von &N"
        .CenterFooter = "&D - &T"
        .RightFooter = "&Z&F"
    End With
End Sub
```

Achtung

Erst ab der Version 2002 (XP) ist es möglich, in Kopf- oder Fußzeilen ohne VBA den Pfad und Dateinamen einzufügen. Falls Sie mit einer früheren Version arbeiten, verwenden Sie die Anweisung `ActiveWorkbook.FullName` an Stelle von `&Z&F`.

Platzhalter	Beschreibung
»&12Codebook«	Schriftgröße und Text
»&«»Courier,Kursiv«»Codebook«	Schriftart, Schriftschnitt und Text
»&UCodebook«	Unterstrichen
»&ECodebook«	Doppelt unterstrichen
»&SCodebook«	Durchgestrichen

Tabelle 3: Platzhalter für Kopf- und Fußzeilen

Platzhalter	Beschreibung
»&XCodebook«	Hochgestellt
»&YCodebook«	Tiefgestellt
»&P«	Aktuelle Seitenzahl
»&N«	Seiten insgesamt
»&D«	Datum
»&T«	Zeit
»&Z«	Pfad (erst ab Version 2002 möglich)
»&F«	Dateiname
»&A«	Tabellenblattname

Tabelle 3: Platzhalter für Kopf- und Fußzeilen (Forts.)

12 Kopf- und Fußzeilen mit Bildern

Erst ab der Office-Version 2002 (XP) lassen sich auch Bilder in Kopf- oder Fußzeilen einfügen. Sie können Bilder einfügen, indem Sie den Menüpunkt ANSICHT | KOPF- UND FUSSZEILE wählen. Klicken Sie auf die Schaltfläche BENUTZERDEFINIERT KOPFZEILE oder BENUTZERDEFINIERT FUSSZEILE. Im folgenden Dialogfenster aktivieren Sie das gewünschte Eingabefeld. Klicken Sie dann auf das vorletzte Symbol (siehe Abbildung 4), um das Dialogfenster GRAFIK EINFÜGEN anzuzeigen. Wählen Sie in Ihrem Dateisystem die gewünschte Grafik aus.



Abbildung 4: Grafik in Kopf- oder Fußzeile einfügen

Falls Sie das Ganze mit dem Makro-Rekorder aufzeichnen und den Code anschließend bereinigen, müssen Sie darauf achten, dass pro Bild zwei Codezeilen erforderlich sind. Zum einen wird mittels `LeftHeaderPicture.FileName` das Bild eingefügt, zum anderen über `LeftHeader` der Platzhalter `&G` eingesetzt.

Hinweis

In unserem Beispiel wird das Bild aus demselben Pfad geholt, in dem sich unsere Beispieldatei befindet: `ThisWorkbook.Path`. Wenn sich Ihr Bild nicht im selben Verzeichnis wie die Mappe befindet, dann lassen Sie das `ThisWorkbook.Path` und das `&` weg und geben in Anführungs- und Schlusszeichen den gesamten Pfad ein. Zum Beispiel:

`"C:\Bilder\Meine Bilder\Idylle.bmp"`

Grundlagen

Allgemein

Datum
Zeit

ungen

Steuerelemente

Befehle
leisten

Objekte

Diagramm

Ereignisse

User-Form

Web/
Mail

Extern

Gemischtes

Spezial

```
'=====
' Auf CD      Buchdaten\Beispiele\01_Einstieg\
' Dateiname   01_03_HeaderFooter.xls
' Modul       mdl_02_HeaderFooterPicture
'=====

Sub FooterAndHeaderWithPicture()
    With ActiveSheet.PageSetup
        .LeftHeaderPicture.FileName = _
            ThisWorkbook.Path & "\Idylle.bmp"
        .LeftHeader = "&G"

        .RightFooterPicture.FileName _
            = ThisWorkbook.Path & "\Idylle.bmp"
        .RightFooter = "&G"
    End With
End Sub
```

Tip

Falls Sie mit der Office-Version 2000 oder niedriger arbeiten, haben Sie zwar nicht die Möglichkeit, Bilder in die Kopfzeile einzufügen, Sie können sich jedoch der Wiederholungszeilen bedienen.

- ▶ Fügen Sie eine Grafik ganz oben in Ihre Tabelle ein
- ▶ Wählen Sie den Menüpunkt DATEI | SEITE EINRICHTEN
- ▶ Aktivieren Sie die Registerkarte TABELLE
- ▶ Klicken Sie bei WIEDERHOLUNGSZEILEN OBEN die Eingabezeile und markieren Sie, bei geöffnetem Dialogfenster, so viele Zeilen der Tabelle, bis das gesamte Bild in der Höhe abgedeckt ist
- ▶ Sie können nun alle Dialogfenster schließen, indem Sie jeweils auf die Schaltfläche OK klicken

13 Kopf- und Fußzeilen auf jedem Tabellenblatt

Wenn Kopf- und Fußzeilen eingesetzt werden, dann kommen sie in der Regel auf jedem Tabellenblatt vor und müssen zudem immer gleich aussehen. Es wäre daher natürlich recht umständlich, eine entsprechende Prozedur für jedes Blatt ausführen zu müssen. Mittels einer `For...Each`-Schleife, die das Einfügen der Kopf- und Fußzeilen umgibt, können Sie in einem Arbeitsgang allen Tabellenblättern der Arbeitsmappe die Kopf- und Fußzeilen zuweisen.

```
'=====
' Auf CD      Buchdaten\Beispiele\01_Einstieg\
' Dateiname   01_03_HeaderFooter.xls
' Modul       mdl_03_HeaderFooterSheets
'=====

Sub HeaderAndFooterForAllSheets()
    Dim ws As Worksheet
```

```
For Each ws In Worksheets
    With ws.PageSetup
        .LeftHeader = "Das"
        .CenterHeader = "Excel-VBA"
        .RightHeader = "Codebook"
        .LeftFooter = "Monika Weber"
        .CenterFooter = "Melanie Breden"
        .RightFooter = "&D"
    End With
Next ws
End Sub
```

Um die Kopf- und Fußzeilen wieder zu entfernen, setzen Sie einfach die Felder zurück, indem Sie einen Leerstring (" ") übergeben.

```
' =====
' Auf CD      Buchdaten\Beispiele\01_Einstieg\
' Dateiname  01_03_HeaderFooter.xls
' Modul      mdl_04_UndoHeaderFooter
' =====

Sub UndoHeaderAndFooter()
    Dim ws As Worksheet

    For Each ws In Worksheets
        With ws.PageSetup
            .LeftHeader = " "
            .CenterHeader = " "
            .RightHeader = " "
            .LeftFooter = " "
            .CenterFooter = " "
            .RightFooter = " "
        End With
    Next ws
End Sub
```

14 Eine Farbtabelle erstellen

In Excel stehen insgesamt 56 Farben zur Verfügung. In der Regel werden die Farben in VBA über den Index angesprochen. Nur acht davon haben sprechende Namen in Form von Konstanten. Welche das sind, haben Sie bereits in der Tabelle 1.1 erfahren. Beim Arbeiten mit dem Index muss man wissen, welcher Index zu welcher Farbe gehört. In der Regel kennt man diese Zuordnung nicht auswendig. Was liegt deshalb näher, als eine kurze VBA-Prozedur zu schreiben, die beides zusammenbringt? In diesem Rezept werden wir eine Farbtabelle erstellen.

Um das Ganze etwas kompakt zu gestalten, werden wir die Farben und deren Index über verschiedene Zeilen verteilen. Pro Zeile sind 14 Farben enthalten. Dazu verwenden wir eine For-Schleife, die von 1 bis 14 zählt. Die Variable i zählt bis 14. In den ersten beiden Zeilen reicht es, lediglich die Variable i zu übergeben. In den weiteren Zeilen werden die Werte 14, 28 und 42 addiert. Damit erhalten wir die insgesamt 56 Farben (siehe Abbildung 5).

Grundlagen

Allgemein

Datum
Zeit

Diagramme

Steuerelemente

Befehle
leisten

Objekte

Diagramm

Ereignisse

User-Form

Web/
Mail

Extern

Gemischtes

Spezial

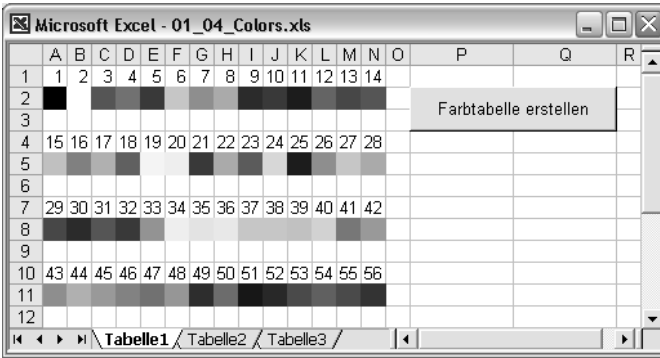


Abbildung 5: Eine Farbtabelle erstellen

```
'=====
' Auf CD      Buchdaten\Beispiele\01_Einstieg\
' Dateiname   01_04_Colors.xls
' Tabelle     Index
' Modul       mdl_01_AllColors
'=====
```

```
Sub AllColors()
    Dim i As Integer

    For i = 1 To 14
        ' Farbindex: 1-14
        Cells(1, i).Value = i
        Cells(2, i).Interior.ColorIndex = i

        ' Farbindex: 15-28
        Cells(4, i).Value = i + 14
        Cells(5, i).Interior.ColorIndex = i + 14

        ' Farbindex: 29-42
        Cells(7, i).Value = i + 28
        Cells(8, i).Interior.ColorIndex = i + 28

        ' Farbindex: 43-56
        Cells(10, i).Value = i + 42
        Cells(11, i).Interior.ColorIndex = i + 42
    Next i

    ' Spaltenbreite verkleinern auf 2 Pixel
    Columns("A:N").ColumnWidth = 2
End Sub
```

15 RGB-Farben

Diesmal arbeiten wir nicht mit dem Farbindex von Excel, sondern mit RGB-Farbwerten. Insgesamt können Sie ganze 16,77 Millionen Farben mischen, wie es Ihnen beliebt. Der Haken bei Excel ist, dass letztlich doch nur 56 Farben (inkl. Schwarz und Weiß) dargestellt werden können.

Die Mischung von RGB-Farben besteht aus drei Farbanteilen: Rot, Grün und Blau. Die einzelnen Farbanteile bewegen sich zwischen 0 und 255. Damit ergeben sich die 16,7 Millionen Farben: 256^3 . Wie ist denn nun der Code für den Farbwert zusammengesetzt? Das Prinzip baut auf den drei RGB-Grundfarben Rot, Grün und Blau auf (RRR, GGG, BBB). In Dreiergruppen können Sie die Farben Rot, Grün und Blau mischen. Wenn Sie beispielsweise die Farbe Rot erhalten möchten, dann verwenden Sie volle Farbdichte 255 für Rot, für Grün 0, also nichts, und ebenso 0 für Blau (255, 0, 0). Vor die Farbmischung wird jeweils »RGB« geschrieben.

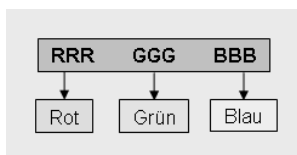


Abbildung 6: RGB-Farbzusammenstellung

Wie kann der RGB-Wert einer Farbe ermittelt werden? Sie können dazu eine VBA-Prozedur verwenden, die den Farbwert aufschlüsselt. Um den reinen Farbwert (nicht Index) einer Farbe in Excel zu ermitteln, verwenden Sie an Stelle von `ColorIndex` lediglich `Color: Range("B1").Interior.Color`. Die Farbe Gelbbraun würde demnach einen Wert von 10079487 zurückgeben. Aus diesem Farbwert wollen wir die RGB-Farbe entschlüsseln. Wir verwenden dazu den Operator `Mod` (Modulo). Der Operator wird dazu verwendet, zwei Werte zu dividieren und daraus den Restwert als Ergebnis zurückzuliefern. Falls Sie damit noch nicht vertraut sein sollten, beachten Sie bitte die nachfolgende Berechnung.

Die Farbe Gelbbraun hat wie gesagt den Farbwert 10079487. Um die drei RGB-Farben Rot, Grün und Blau daraus zu ermitteln, müssen entsprechend drei Berechnungen angestellt werden:

$10079487 / 256$ ergibt 39372,99609375. Dieser Wert wird abgerundet und ergibt somit 39372.

Das Ergebnis wird mit 256 multipliziert: $39372 * 256$ und ergibt dann den Wert: 10079232.

Der Wert 10079232 wird vom Wert 10079487 subtrahiert und liefert als Ergebnis: 255.

Damit haben wir den ersten RGB-Wert, also **Rot**.

Um die Farbe **Grün** zu erhalten, müssen wir das erste Ergebnis der Farbe Rot, also 39372, weiterrechnen:

$$39372 / 256 = 153,796875$$

$$153 * 256 = 39168$$

$$39372 - 39168 = 204$$

Die Farbe **Blau** ergibt sich aus dem Restwert der obigen Berechnung $39372 / 256 = 153,796875$ und ist somit 153.

Genau diese Berechnung nehmen wir in unserer VBA-Prozedur nach der Dimensionierung vor. Im letzten Codeblock werden die einzelnen RGB-Farben an die Zellen B3 bis B5 übergeben. Den Zellen B7 bis B9 sind der RGB-Wert, der Farbwert und der Index zu entnehmen (siehe Abbildung 7).

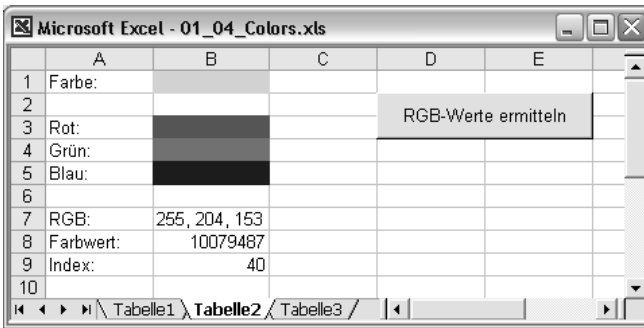


Abbildung 7: RGB-Werte ermitteln

```
'=====
' Auf CD      Buchdaten\Beispiele\01_Einstieg\
' Dateiname  01_04_Colors.xls
' Tabelle     RGB
' Modul      mdl_01_AllColors
'=====
```

```
Sub RGBColors()
    Dim lngRed As Long
    Dim lngGreen As Long
    Dim lngBlue As Long
    Dim lngDummy As Long

    ' Berechnungen für Rot, Grün und Blau
    lngDummy = Range("B1").Interior.Color ' 10079487
    lngRed = lngDummy Mod 256 ' 255 <--- Rot
    lngDummy = (lngDummy - lngRed) / 256 ' 39732
    lngGreen = lngDummy Mod 256 ' 204 <--- Grün
    lngBlue = (lngDummy - lngGreen) / 256 ' 153 <--- Blau

    ' Farben und Werte an Zellen übergeben
    Range("B3").Interior.Color = RGB(lngRed, 0, 0)
    Range("B4").Interior.Color = RGB(0, lngGreen, 0)
    Range("B5").Interior.Color = RGB(0, 0, lngBlue)
    Range("B7").Value = lngRed & ", " & lngGreen & ", " & lngBlue
    Range("B8").Value = Range("B1").Interior.Color
    Range("B9").Value = Range("B1").Interior.ColorIndex
End Sub
```

16 Kommentare einfügen

Sie können einer Zelle manuell einen Kommentar hinzufügen, indem Sie die entsprechende Zelle mit der rechten Maustaste anklicken. Sie öffnen so ein Kontextmenü. Wählen Sie den Eintrag KOMMENTAR EINFÜGEN. Wenn Sie dies per VBA tun möchten, müssen Sie beachten, dass der Code zu einem Fehler führen würde, wenn bereits ein Kommentar in der Zelle vorhanden wäre. Um dies zu verhindern, arbeiten wir mit einer If-Entscheidung, die erst eine notwendige Prüfung vornimmt. Wenn kein Kommentar vorhanden ist (Is Nothing), wird unser Kommentar eingefügt (AddComment). Wenn bereits ein Kommentar vorhanden ist, bleibt dieser erhalten.

```
'=====
' Auf CD      Buchdaten\Beispiele\01_Einstieg\
' Dateiname   01_05_Comments.xls
' Tabelle     Comment
' Modul       mdl_01_NewComment
'=====

Sub NewCommentCell()
    If ActiveCell.Comment Is Nothing Then
        ActiveCell.AddComment "Mein Kommentar"
    End If
End Sub
```

Nachfolgend wollen wir das Codebeispiel etwas erweitern. Falls bereits ein Kommentar in der Zelle vorhanden ist, wird ein Nachrichtenfenster MsgBox angezeigt. Das Meldungsfenster enthält die Schaltflächen JA und NEIN (vbYesNo). Wenn die Schaltfläche Ja gedrückt wird, wird der alte Kommentar gelöscht und der neue eingefügt. Um dies zu verdeutlichen, schreiben wir im Kommentarfenster das Datum und die Uhrzeit, welche angibt, wann der Kommentar eingefügt wurde.



Abbildung 8: Kommentar mit Rückfrage

```
'=====
' Auf CD      Buchdaten\Beispiele\01_Einstieg\
' Dateiname   01_05_Comments
' Tabelle     Comment
' Modul       mdl_01_NewComment
'=====

Sub NewCommentCellWithRequest()
    Dim str As String
    Dim strComment As String

    strComment = Date & Chr(10) & Time & Chr(10) & "Mein Kommentar"

    If ActiveCell.Comment Is Nothing Then
        ActiveCell.AddComment strComment
    Else

```

Grundlagen

Allgemein

Datum
Zeit

Funktionen

Steuerelemente

Befehle
Anweisungen

Objekte

Diagramme

Ereignisse

User-Formen

Web/
Mail

Extern

Gemischtes

Spezial

```

str = MsgBox("Es ist bereits ein Kommentar vorhanden. " _
            & "Möchten Sie ihn löschen und den Neuen " _
            & "einfügen?", vbYesNo)
If str = vbYes Then
    With ActiveCell
        .Comment.Delete
        .AddComment strComment
    End With
Else
    Exit Sub
End If
End If
End Sub

```

17 Kommentare ergänzen

Mittels Kommentaren haben Sie die Möglichkeit, laufend zu dokumentieren, wer welche Änderungen in welcher Zelle vorgenommen hat. Damit die Änderungen laufend vorgenommen werden, verwenden wir die Ereignis-Prozedur `Worksheet_Change`, die an das entsprechende Tabellenblatt gebunden ist.

Hinweis

Mehr zum Thema Ereignis-Prozeduren erfahren Sie in der Kategorie »Ereignisse«.

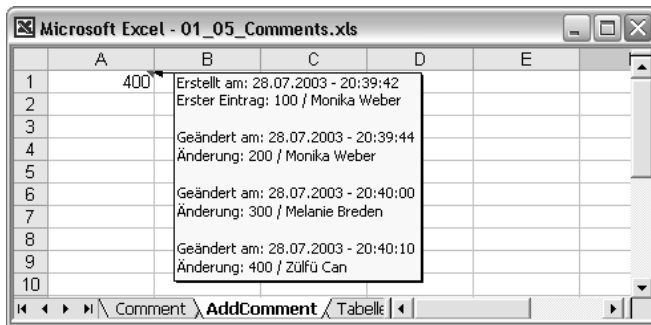


Abbildung 9: Änderungsverfolgung innerhalb eines Kommentars

Damit nicht der Debugger gestartet wird, wenn beispielsweise eine Löschung mehrerer Zellen erfolgt, verwenden wir zu Beginn der Prozedur die Anweisung `If Target.Column > 1`. Das heißt, wenn mehr als eine Zelle markiert wurde, wird die Prozedur verlassen.

`Target` bedeutet übersetzt »Ziel«. Hier in unserem Beispiel ist damit die aktive Zelle gemeint. Da sich alles um die aktive Zelle dreht, können wir die darauf folgende Prozedur in einer `With`-Anweisung unterbringen. Damit ersparen wir uns einerseits das vermehrte Ansprechen von `Target` und erhöhen andererseits die Geschwindigkeit der Prozedur.

Innerhalb der `With`-Anweisung verwenden wir eine `If`-Entscheidung. Diese prüft, ob bereits ein Kommentar in der aktiven Zelle enthalten ist. Trifft dies nicht zu, wird ein entsprechender Kommentar eingefügt (`AddComment`).

Sollte in der Zelle bereits ein Kommentar vorhanden sein (Else), wird dieser ergänzt. Die Variable `str` wird benötigt, um den bereits vorhandenen Text zu speichern, denn die Anweisung `Comment.Text` baut den Inhalt des Kommentars neu auf. Für den neuen Aufbau des Kommentars möchten wir den alten Text nicht verlieren und setzen die Variable `str` beim Neuaufbau wieder ein.

```
'=====
' Auf CD      Buchdaten\Beispiele\01_Einstieg\
' Dateiname  01_05_Comments
' Tabelle     AddComment
' Ereignis    Tabelle2(AddComment)
'=====

Private Sub Worksheet_Change(ByVal Target As Range)
    Dim str As String

    If Target.Count > 1 Then Exit Sub

    With Target
        If .Comment Is Nothing Then
            ' Wenn noch kein Kommentar besteht, einen Neuen einfügen
            .AddComment "Erstellt am: " & Date & " - " & Time & _
                Chr(10) & "Erster Eintrag: " & .Value & _
                " / " & Application.UserName
        Else
            ' Wenn bereits ein Kommentar besteht, diesen ergänzen
            str = .Comment.Text & Chr(10)
            .Comment.Text str & Chr(10) & "Geändert am: " & _
                Date & " - " & Time & Chr(10) & _
                "Änderung: " & .Value & " / " & _
                Application.UserName
        End If

        ' Automatische Größenanpassung des Kommentars
        .Comment.Shape.TextFrame.AutoSize = True
    End With
End Sub
```

Tipp

Application.UserName gibt den Namen zurück, der unter dem Menüpunkt EXTRAS | OPTIONEN, Registerkarte ALLGEMEIN festgelegt wurde. Bei dieser Einstellung ist es für den Benutzer natürlich ein Leichtes, den Namen selbst zu ändern.

Sicherer ist, wenn Sie den Namen des am System angemeldeten Benutzers verwenden, die Anweisung dazu lautet: Environ("UserName").

18 Kommentare auslesen

Kommentare können am Ende eines Tabellenblattes ausgegeben werden. Verwenden Sie dazu den Menüpunkt DATEI | SEITE EINRICHTEN, Registerkarte TABELLE. Im DropDown-Feld KOMMENTARE wählen Sie den Eintrag AM ENDE DES BLATTES.

Grundlagen

Allgemein

Datum
Zeit

tungen

Steuer
elemente

Befehl
leisten

Objekte

Diagramm

Ereignisse

User-
Form

Web/
Mail

Extern

Gemischtes

Spezial

Alternativ zur oben genannten Excel-Einstellung können Sie auch eine VBA-Prozedur verwenden, die den Inhalt der Kommentare der gesamten Mappe auf einem separaten Tabellenblatt ausgibt.

Damit keine Informationen in bestehenden Tabellenblättern überschrieben werden, legen wir im ersten Schritt ein neues Tabellenblatt an. Das Tabellenblatt wird an letzter Stelle in der Mappe eingefügt. Die Variable `intWS` verwenden wir, um die Anzahl der bereits vorhandenen Tabellenblätter zu ermitteln.

Zu Beginn der `With`-Anweisung wird die Überschrift für das neue Tabellenblatt aufbereitet und fett formatiert.

In der ersten `For`-Schleife werden alle Tabellenblätter abgearbeitet, denn wir wollen im neu erstellten Tabellenblatt jeden in der Mappe enthaltenen Kommentar ausgeben. In der zweiten `For`-Schleife wird innerhalb jedes Tabellenblattes jede Zelle im benutzten Bereich `UsedRange` auf einen möglichen Kommentar geprüft.

Die `If`-Entscheidung prüft, ob ein Kommentar in der Zelle vorhanden ist. Wenn dies nicht zutrifft, geschieht nichts. Wenn ein Kommentar gefunden wird, dann wird dessen Inhalt ins neu erstellte Tabellenblatt geschrieben. In der zweiten Codezeile wird der Name des Tabellenblattes in die Spalte B geschrieben. In der dritten Codezeile wird die Zelladresse `Address` in die Spalte C geschrieben. Die beiden `False` in Klammern bedeuten, dass das Dollarzeichen \$, welches für eine absolute Zelladressierung steht, weggelassen werden soll.

Am Ende der Prozedur wird die Breite der drei Spalten automatisch auf deren längsten Inhalt angepasst (`AutoFit`). Die Höhe der Zeilen im benutzten Bereich wird ebenfalls angepasst.

```
' =====
' Auf CD      Buchdaten\Beispiele\01_Einstieg\
' Dateiname   01_05_Comments
' Tabelle     GetAllComments
' Modul       mdl_02_GetAllComments
' =====
```

```
Sub GetAllComments()
    Dim intWS As Integer
    Dim i As Integer
    Dim c As Range

    intWS = Worksheets.Count
    Worksheets.Add After:=Worksheets(intWS)

    ' Spaltenüberschrift für das neue Tabellenblatt
    With Worksheets(intWS + 1)
        .Range("A1").Value = "Kommentar"
        .Range("B1").Value = "Tabellenblatt"
        .Range("C1").Value = "Zelle"
        .Range("A1:C1").Font.Bold = True

        For i = 1 To intWS
            For Each c In Worksheets(i).UsedRange
                If c.Comment Is Nothing Then
                Else
                    ' Spalte A: Inhalt des Kommentars
                    .Cells(65536, 1).End(xlUp).Offset(1, 0).Value = _
```

```
        c.Comment.Text
        ' Spalte B: Name des Tabellenblatts
        .Cells(65536, 2).End(xlUp).Offset(1, 0).Value = _
            Worksheets(i).Name
        ' Spalte C: Zelle in der sich der Kommentar befindet
        .Cells(65536, 3).End(xlUp).Offset(1, 0).Value = _
            c.Address(False, False)
    End If
Next c
Next i

' Automatisches Anpassen der Spaltenbreite und Zeilenhöhe
.Columns("A:C").AutoFit
.UsedRange.Rows.AutoFit
End With
End Sub
```

19 Kommentare formatieren

Kommentare und deren Inhalt lassen sich ebenso wie Zellinhalte formatieren. Bei dem Rahmen, der den Kommentar darstellt, handelt es sich im Grunde genommen um nichts anderes als um eine Art Textfeld und kann somit gleich behandelt werden. Um einen Kommentar manuell zu formatieren, klicken Sie ihn mit der rechten Maustaste an. Wählen Sie aus dem Kontextmenü den Eintrag KOMMENTAR FORMATIEREN. In dem Dialogfenster, das sich nun öffnet, können die gewünschten Einstellungen vorgenommen werden.

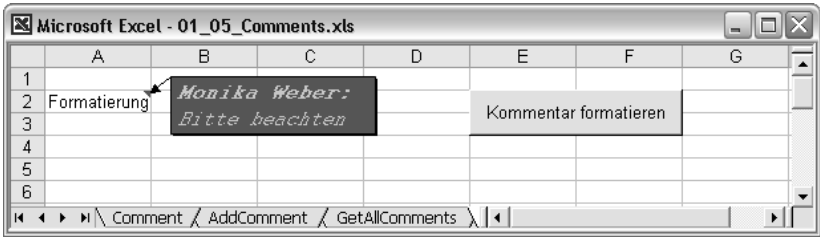


Abbildung 10: Kommentare formatieren

In unserem Code wird zuerst geprüft, ob in der aktiven Zelle ein Kommentar vorhanden ist. Wenn nicht, wird eine entsprechende Meldung `MsgBox` angezeigt. Falls in der Zelle ein Kommentar vorhanden ist, wird ihr eine rote Hintergrundfarbe zugewiesen. Es wird hier ein RGB-Farbwert verwendet. Mehr zum Thema RGB-Farben können Sie im Rezept 16 nachlesen.

In der `With`-Anweisung wird der Kommentartext formatiert. Zuerst wird die Schriftart `Courier` eingestellt, danach die Schriftgröße 14, die Schriftfarbe 6 und schließlich der Schriftschnitt `Kursiv Italic`. Es können noch weitere Formatierungen eingesetzt werden.

Nach der `With`-Anweisung wird die Größe des Kommentar-Rahmens angepasst.

```
' =====
' Auf CD      Buchdaten\Beispiele\01_Einstieg\
' Dateiname  01_05_Comments
' Tabelle     FormatComment
```

Grundlagen

Allgemein

Datum
Zeit

Formeln

Steuerelemente

Befehle
leisten

Objekte

Diagramm

Ereignisse

User-Form

Web/
Mail

Extern

Gemischtes

Spezial

```
' Modul      mdl_03_FormatComment
' =====

Sub FormatComment()
    With ActiveCell
        If .Comment Is Nothing Then
            MsgBox "Es ist kein Kommentar in dieser Zelle vorhanden."
        Else
            ' Die Farbe der Kommentarbox verändern
            .Comment.Shape.Fill.ForeColor.RGB = RGB(255, 0, 0)

            ' Die Schrift des Kommentarboxinhaltes verändern
            With .Comment.Shape.TextFrame.Characters.Font
                .Name = "Courier"
                .Size = 14
                .ColorIndex = 6
                .Italic = True
            End With

            ' Automatisches Anpassen der Kommentarboxgröße
            .Comment.Shape.TextFrame.AutoSize = True
        End If
    End With
End Sub
```

20 Kommentare löschen

Es gibt zwei Möglichkeiten, Kommentare zu löschen. Sie können einerseits die Anweisung `Comment.Delete` oder `ClearComments` verwenden. Der Vorteil bei `ClearComment` ist, dass kein Fehler verursacht wird, wenn in der Zelle kein Kommentar vorhanden ist. Der ersten Prozedur in diesem Rezept können Sie beide Möglichkeiten entnehmen.

```
' =====
' Auf CD      Buchdaten\Beispiele\01_Einstieg\
' Dateiname   01_05_Comments
' Tabelle     DeleteComments
' Modul      mdl_04_DeleteComments
' =====

Sub DeleteOneComment()
    If Range("B2").Comment Is Nothing Then
        MsgBox "Es ist kein Kommentar in dieser Zelle vorhanden."
    Else
        Range("B2").Comment.Delete
    End If

    ' Alternative:
    Range("B2").ClearComments
End Sub
```

Falls Sie sämtliche Kommentare in einem Tabellenblatt entfernen möchten, verwenden Sie entweder eine Schleife, die den Befehl `Comment.Delete` enthält, oder – wie im folgenden Beispiel – die

Methode `SpecialCells`. Um zu verhindern, dass der Debugger gestartet wird, wenn in dem Tabellenblatt keine Kommentare enthalten sind, zählen wir in einer `If`-Entscheidung die Anzahl der Kommentare. Der Befehl zum Entfernen der Kommentare wird nur ausgelöst, wenn die Zahl der gezählten Kommentare größer ist.

```
'=====
' Auf CD      Buchdaten\Beispiele\01_Einstieg\
' Dateiname   01_05_Comments
' Tabelle     DeleteComments
' Modul       mdl_04_DeleteComments
'=====

Sub DeleteAllCommentsSheet()
    With ActiveSheet
        If .Comments.Count > 0 Then
            .Cells.SpecialCells(xlCellTypeComments).ClearComments
        End If
    End With
End Sub
```

Um sämtliche Kommentare aus einer Excel-Arbeitsmappe zu entfernen, können Sie ebenfalls die Methode `ClearComments` verwenden. Die Anweisung wird durch eine `For...Each`-Schleife umgeben, welche alle Tabellenblätter abarbeitet. In der `If`-Entscheidung wird geprüft, ob in der Mappe ein Kommentar vorhanden ist. Diese Prüfung ist erforderlich, da sonst beim Nichtvorhandensein eines Kommentars der Debugger gestartet würde.

```
'=====
' Auf CD      Buchdaten\Beispiele\01_Einstieg\
' Dateiname   01_05_Comments
' Tabelle     DeleteComments
' Modul       mdl_04_DeleteComments
'=====

Sub DeleteAllCommentsWorkbook()
    Dim ws As Worksheet

    For Each ws In Worksheets
        With ws
            If .Comments.Count > 0 Then
                .Cells.SpecialCells(xlCellTypeComments). _
                    ClearComments
            End If
        End With
    Next ws
End Sub
```

21 Verfügbare Systeminformationen auslesen (Environ)

Bereits im Rezept 17 haben wir auf die Funktion `Environ` hingewiesen. Man hat damit unter anderem die Möglichkeit, den zurzeit am System angemeldeten Benutzer zu ermitteln. Der englische Begriff `Environ` bedeutet übersetzt »Umgebung«.

Grundlagen

Allgemein

Datum
Zeit

Umgang

Steuerelemente

Befehle
leisten

Objekte

Diagramm

Ereignisse

User-Form

Web/
Mail

Extern

Gemischtes

Spezial

Je nach verwendetem Betriebssystem können mit dieser Funktion unterschiedliche System-Informationen ausgelesen werden. Wir verwenden dazu eine Do...Loop-Schleife, die alle verfügbaren Informationen im aktiven Tabellenblatt ausgibt.

```
'=====
' Auf CD      Buchdaten\Beispiele\01_Einstieg\
' Dateiname   01_06_ReadSystemInfos
' Modul       mdl_01_ReadSystemInfos
'=====

Sub ReadSystemInformations()
    Dim i As Integer
    i = 1
    Do While Environ(i) <> ""
        Cells(i, 1) = Environ(i)
        i = i + 1
    Loop
End Sub
```

Nun fragen Sie sich vielleicht, welche Informationen in die runden Klammern nach der Environ-Funktion angegeben werden müssen? Sie erhalten diese, indem Sie die obige Prozedur ausführen. In Spalte A der Beispielmappe sind nun die möglichen Werte zu finden. Die Information, die jeweils vor dem Gleichheitszeichen (=) steht, kann in die runden Klammern geschrieben werden. Um beispielsweise den angemeldeten Benutzer sowie das angewandte Betriebssystem zu ermitteln, verwenden Sie folgende Prozedur. Achten Sie dabei auf den Eintrag in den runden Klammern. Die durch die Prozedur ermittelten Informationen werden in einem Meldungsfenster MsgBox angezeigt.

```
'=====
' Auf CD      Buchdaten\Beispiele\01_Einstieg\
' Dateiname   01_06_ReadSystemInfos
' Modul       mdl_01_ReadSystemInfos
'=====

Sub GetTwoEnvironInfos()
    MsgBox Environ("UserName") & Chr(10) & _
           Environ("OS")
End Sub
```

22 Eigenschaften auslesen

Bei den Eigenschaften unterscheidet man zwischen integrierten und benutzerdefinierten Eigenschaften. Beide sind unter dem Menüpunkt DATEI | EIGENSCHAFTEN zu finden. Es sind insgesamt 30 integrierte Dokumenteigenschaften in einer Mappe vorhanden. Diese sind auf den ersten vier Registerkarten nach Thema gruppiert zu finden.

Falls Ihnen die integrierten Dokumenteigenschaften nicht ausreichen sollten, können Sie zusätzlich eigene, benutzerdefinierte Eigenschaften erstellen. Verwenden Sie dazu das Register ANPASSEN, füllen Sie die Felder NAME und WERT aus und bestätigen Sie die Eingabe, indem Sie auf die Schaltfläche HINZUFÜGEN klicken.



Abbildung 11: Integrierte Dokumenten-Eigenschaften auslesen

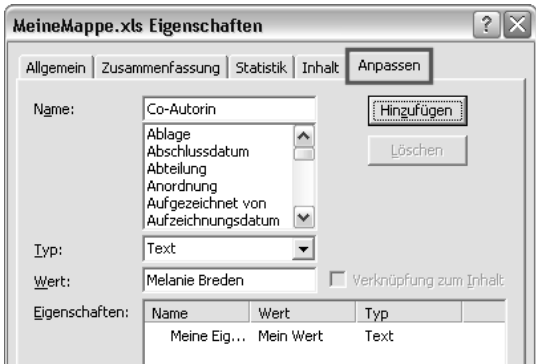


Abbildung 12: Benutzerdefinierte Dokumenteigenschaften

Beide Eigenschaften-Typen, sowohl integriert als auch benutzerdefiniert, können per VBA ausgelesen werden.

Um den Code etwas übersichtlicher zu gestalten, werden wir zu Beginn der Prozedur `GetProperties` die Objekte mit der `Set`-Anweisung referenzieren. Am Ende der Prozedur geben wir die Objekte sicherheitshalber wieder frei mit `Set ... = Nothing`. Damit werden die Objekte zurückgesetzt und der Speicher geleert.

In der ersten `For...Next`-Schleife lesen wir die integrierten Dokumenteigenschaften `BuiltinDocumentProperties` der Beispieldatei `ThisWorkbook` aus. In unserem Tabellenblatt werden die Namen der Eigenschaften in die Spalte A geschrieben. Die Werte der Felder werden in die Spalte B geschrieben.

In der zweiten `For...Next`-Schleife lesen wir sämtliche benutzerdefinierten Eigenschaften aus. Dazu verwenden wir die Eigenschaft `CustomDocumentProperties`. Die Namen werden in die Spalte D und die Werte in die Spalte E geschrieben.

```

'=====
' Auf CD      Buchdaten\Beispiele\01_Einstieg\
' Dateiname  01_07_Properties
' Modul      mdl_01_GetProperties
'=====

Sub GetProperties()
    Dim i As Integer
    Dim objBDP As Object
    Dim objCDP As Object

    Set objBDP = ThisWorkbook.BuiltinDocumentProperties
    Set objCDP = ThisWorkbook.CustomDocumentProperties
    On Error Resume Next

    Cells(1, 1) = "Integrierte Dokumenteigenschaften"
    For i = 1 To objBDP.Count
        Cells(i, 1) = objBDP(i).Name
        Cells(i, 2) = objBDP(i).Value
    Next i

    Cells(1, 4) = "Benutzerdefinierte Dokumenteigenschaften"
    For i = 1 To objCDP.Count
        Cells(i, 4) = objCDP(i).Name
        Cells(i, 5) = objCDP(i).Value
    Next i

    Columns("A:E").AutoFit

    Set objBDP = Nothing
    Set objCDP = Nothing
End Sub

```

23 Eigenschaften ausfüllen bzw. erstellen

Im vorangegangenen Rezept haben wir die Eigenschaften ausgelesen. Natürlich geht das auch umgekehrt. Felder von Eigenschaften können beschrieben werden. Es handelt sich dabei vor allem um die befüllbaren Eigenschaften der Registerkarte ZUSAMMENFASSUNG.

Nach der Variablendeklaration und Referenzierung werden drei der Eigenschaften-Felder ausgefüllt. Wenn Sie die Namen der Felder nicht kennen, führen Sie am einfachsten die Prozedur aus dem vorangegangenen Rezept aus. In der Spalte A finden Sie die Definitionen, die Sie verwenden können. Denken Sie dabei jedoch daran, dass nicht alle Felder zur Verfügung stehen, sondern nur jene, welche im Eigenschaften-Fenster manuell ausgefüllt werden können.

Der zweite Codeblock zeigt Ihnen, wie Sie eine benutzerdefinierte Dokumenteigenschaft mittels VBA erstellen können. Verwenden Sie dazu die Methode `Add`. Es stehen die Argumente `Name`, `LinkToContent`, `Value` und `Type` zur Verfügung. Bei `Name` und `Value` können Sie selbst deren Benennung festlegen. `LinkToContent` bedeutet: Link zu einem Zelleninhalt. Diesem Argument wird in der Regel `False` zugewiesen. Ein Link ist nur dann möglich, wenn Sie sich auf einen benannten Bereich eines Tabellenblattes beziehen. Für das Argument `Type` stehen vier Konstanten zur Verfügung:

Konstante	Beschreibung
msoPropertyTypeString	Text
msoPropertyTypeDate	Datum
msoPropertyTypeNumber	Zahl
msoPropertyTypeBoolean	Wahr oder Falsch

Tabelle 4: Konstanten zum Argument Type

```
'=====
' Auf CD      Buchdaten\Beispiele\01_Einstieg\
' Dateiname  01_07_Properties
' Modul      mdl_02_SetProperties
'=====

Sub SetProperties()
    Dim objBDP As Object
    Dim objCDP As Object

    Set objBDP = ThisWorkbook.BuiltinDocumentProperties
    Set objCDP = ThisWorkbook.CustomDocumentProperties

    ' Integrierte Dokumenteigenschaftenfelder ausfüllen
    objBDP("Title") = ("Das Excel-VBA-Codebook, 2. Auflage")
    objBDP("Subject") = ("Das Eigenschaften-Fenster")
    objBDP("Author") = ("Monika Weber")

    ' Benutzerdefinierte Dokumenteigenschaft erstellen
    On Error GoTo ErrorHandler
    objCDP.Add _
        Name:="Meine Eigenschaft", _
        LinkToContent:=False, _
        Value:="Mein Wert", _
        Type:=msoPropertyTypeString

    Set objBDP = Nothing
    Set objCDP = Nothing
    Exit Sub

ErrorHandler:
    MsgBox "Eine benutzerdef. Eigenschaft mit diesem Namen " & _
        "besteht bereits."

End Sub
```

24 Funktionen

In fast jeder Programmiersprache sind Funktionen als feste Bestandteile integriert. In Microsoft Excel unterscheidet man zwischen drei Gruppen von Funktionen:

- ▶ Excel-Funktionen
- ▶ VBA-Funktionen
- ▶ Benutzerdefinierte Funktionen

Grundlagen
Allgemein
Datum
Zeit
tungen
Steuer
elemen
Befehl
leisten
Objekte
Diagramm
Ereignisse
User-Form
Web/Mail
Extern
Ge-mischtes
Specia

Eine Funktion ist dazu vorgesehen, das Ergebnis einer Berechnung oder einen anderen Rückgabewert (Zeichenkette, Wahrheitswert oder Fehlerwert) auszugeben. Im Tabellenblatt liefert eine Funktion (Formel) nur ein Ergebnis in der Zelle, aus der die Funktion aufgerufen wird. Es ist nicht möglich, Werte durch eine Funktion in eine andere als die aufrufende Zelle zu schreiben. Ebenso ist es mit VBA-Funktionen nicht möglich, den Inhalt oder Formatierung von Zellen zu verändern.

Grundsätzlich gilt für benutzerdefinierte Tabellenfunktionen, dass sie keine Formatierungen transportieren können. Über Funktionen, welche aus einer Zelle aufgerufen werden, können Sie beispielsweise keine Hintergrundformate oder Schriftattribute festlegen, dazu benötigen Sie eine Sub-Prozedur.

Funktionen haben in VBA einen besonderen Status, weil sie in die ereignisgesteuerte Berechnungshierarchie eines Tabellenblattes eingebunden werden. Das heißt, es darf kein schreibender Zugriff auf einen Bereich erfolgen. Wenn durch eine Funktion ein Wert in eine Zelle geschrieben oder geändert würde, hätte dieser Vorgang eine Neuberechnung der gesamten Excel-Arbeitsmappe zur Folge. Die auslösende Funktion würde wieder und wieder aufgerufen, was letztendlich in einer Endlosschleife oder einer Fehlermeldung enden könnte.

Wenn Sie in einer Zelle den Namen einer integrierten Excel-Funktion in Kleinbuchstaben eingeben, wandelt Excel die Schreibweise der Funktion in Großbuchstaben um. Geben Sie zum Beispiel die Formel `=summe(a1:a10)` in eine Zelle ein, ändert Excel diese Formel in `=SUMME(A1:A10)`. Diesen Effekt können Sie auch bei Ihren benutzerdefinierten Tabellenblatffunktionen erzielen. Die Schreibweise im Funktionskopf der VBA-Umgebung spielt dabei keine Rolle. Wichtig ist, in welcher Schreibweise Sie die Funktion in einer Zelle das *erste* Mal in der Arbeitsmappe angeben. Dies gilt jedoch nur bei der Verwendung benutzerdefinierter Funktionen, welche aus einer Zelle aufgerufen werden.

Die Funktionen *Brutto* und *Netto* aus Rezept 27 haben wir bei der ersten Verwendung in der Beispieldatei in *Großbuchstaben* (*BRUTTO*, *NETTO*) geschrieben. Wenn Sie diese Funktionen im Tabellenblatt der Beispieldatei in der Kleinschreib- oder gemischten Schreibweise verwenden, wandelt Excel sie nach Verlassen der Zelle in Großbuchstaben um.

Verwenden Sie benutzerdefinierte Funktionen innerhalb von VBA-Prozeduren, ändert Excel die Schreibweise des Funktionsnamens entsprechend der angegebenen Schreibweise im Funktionskopf um.

25 Excel-Tabellenfunktionen in Visual Basic verwenden

Die erste der drei genannten Funktionsgruppen *Excel-Funktionen* beinhaltet alle Formeln und verschachtelte Funktionen innerhalb einer Formel, welche in den Zellen des Tabellenblattes angewandt werden. Eine Formel erkennen Sie grundsätzlich daran, dass Sie mit einem Gleichheitszeichen `=` beginnt.

Eine Liste aller integrierter und benutzerdefinierter Tabellenfunktionen finden Sie im Dialog *FUNKTION EINFÜGEN* (siehe Abbildung 13). Klicken Sie in der Bearbeitungszeile auf das Symbol *fx*, um den Dialog aufzurufen, oder drücken die Tastenkombination `[F4] + [F3]`. Ab Excel 2002 gelangen Sie über den Hyperlink *HILFE FÜR DIESE FUNKTION* zum Hilfethema der markierten Funktion. Geben Sie in älteren Versionen den Namen der Funktion im Hilfe-Assistenten ein, um zum entsprechenden Hilfethema der Online-Hilfe zu gelangen.

In der Excel-Hilfe finden Sie unter dem Stichwort *Tabellenfunktionen/Liste der in Visual Basic verfügbaren Arbeitsblatffunktionen* ebenfalls eine Liste aller integrierten Tabellenfunktionen.



Abbildung 13: Liste aller zur Verfügung stehenden Tabellenfunktionen

In der VBA-Programmierung können Sie alle Tabellenblatt-Funktionen über das Worksheet-Function-Objekt anwenden. Zur Benutzung muss der englische Funktionsname verwendet werden.

Hinweis

Bei der Installation von Excel wird die Datei *VBALISTE.xls* im Ordner *C:\Programme\Microsoft Office\Office\1031* gespeichert. Sie beinhaltet fast alle Tabellenblattfunktionen in deutscher und englischer Schreibweise. Zusätzlich finden Sie in der Datei die gängigsten Excel- und VBA-Schlüsselwörter in beiden Sprachen. Diese Liste wurde seit 1997 von Microsoft nicht mehr aktualisiert. Aus diesem Grund sind neuere Funktionen und Schlüsselwörter darin nicht enthalten.

Da der Objektkatalog ebenfalls alle Funktionen aufführt, hilft Ihnen die Intellisense-Liste bei der Auswahl einer Tabellenfunktion. Wenn Sie im Anschluss an den Ausdruck `Application.WorksheetFunction` einen Punkt (.) eingeben, können Sie in der aufgeklappten Intellisense-Liste eine Tabellenblattfunktion auswählen (siehe Abbildung 14).

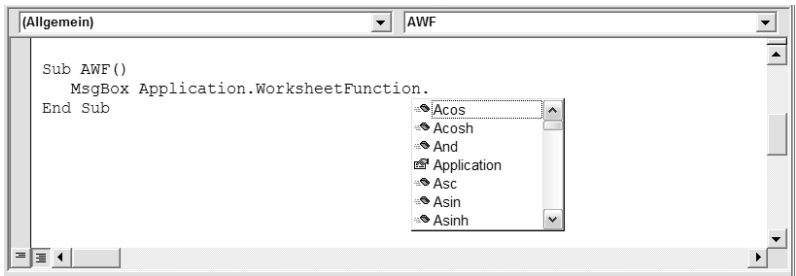


Abbildung 14: Tabellenblattfunktionen des Objektkataloges

Nach Auswahl einer Funktion und Eingabe einer öffnenden runden Klammer () zeigt Ihnen die Quickinfo alle für diese Funktion zur Verfügung stehenden Argumente an.

Grundlagen

Allgemein

Datum und Zeit

Formeln und Berechnungen

Steuerungselemente

Befehlszeile

Objekte

Diagramme

Ereignisse

User-Formen

Web/Mail

Extern

Gemischtes

Spezial

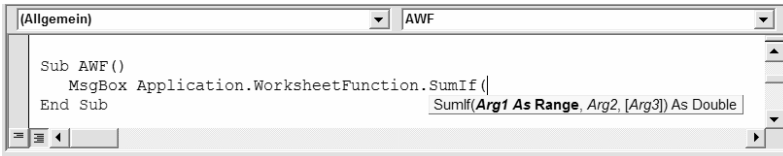


Abbildung 15: Quickinfo mit Argumenten der angegebenen Funktion

In der folgenden Prozedur wird die Tabellenblatfunktion `SUMME()` angewandt, um in einem Meldungsfenster die Summe eines angegebenen Bereichs anzuzeigen. Da der Ausdruck `Application.WorksheetFunction` sehr lang ist und der Code dadurch unübersichtlich wird, deklarieren wir die Variable `AWF` vom Typ `WorksheetFunction`. Diese wird in der Prozedur anstelle des ganzen Ausdrucks vor eine Tabellenblatfunktion geschrieben.

```
'=====
' Auf CD      Buchdaten\Beispiele\01_Einstieg\
' Dateiname   01_08_Functions
' Tabelle     AWF
' Modul       mdl_01_WorksheetFunction
'=====
```

```
Sub SumMessage()
    Dim AWF As WorksheetFunction
    Dim rngArea As Range

    Set AWF = Application.WorksheetFunction
    Set rngArea = Worksheets("AWF").Range("A1:A10")

    MsgBox AWF.Sum(rngArea)
End Sub
```

26 VBA-Funktionen

Neben den Tabellenfunktionen gibt es in der VBA-Programmierung unzählige integrierte VBA-Funktionen. Sie werden in vielen Rezepten dieses Buches wie selbstverständlich verwendet. Um zum Beispiel das aktuelle Datum der Systemeinstellung zu ermitteln, verwenden Sie die `Date`-Funktion.

```
'=====
' Auf CD      Buchdaten\Beispiele\01_Einstieg\
' Dateiname   01_08_Functions
' Tabelle     VBA-Funktion
' Modul       mdl_02_VBAFunction
'=====
```

```
Sub DateMessage()
    MsgBox "Heute ist der " & Date
End Sub
```

27 Benutzerdefinierte Tabellenfunktionen

Tabellen- und VBA-Funktionen erwarten in der Regel Übergabewerte, auf deren Grundlage sie ihre Berechnungen durchführen. Sie geben die Berechnungsergebnisse an die aufrufende Zelle zurück. Je nachdem, wie Sie benutzerdefinierte Funktionen programmieren, können Sie sowohl im Tabellenblatt eingesetzt als auch aus anderen Prozeduren aufgerufen werden.

Um in einer Zelle zu dem Betrag 100,00 +E+U+R+O+ die Mehrwertsteuer von derzeit 16% zu addieren, verwenden Sie beispielsweise die Formel =100*1,16. Anstelle des festen Wertes können Sie auch einen Zellbezug angeben. Das Ergebnis lautet 116.

Für diese Berechnung haben wir die benutzerdefinierte Funktion `Brutto` programmiert. In der Argumentenliste der Funktion wurden zwei Variablen vom Datentyp `Double` deklariert. Der erste Parameter `dblValue` steht für die Angabe des Grundwertes und erwartet einen numerischen Wert, Zellbezug oder Variable, welche auf einen Zahlenwert verweist. Der zweite Parameter `dblPercent` gibt den Prozentwert an, welcher zu dem Grundwert `dblValue` addiert werden soll.

Hinweis

Beachten Sie, dass die Angabe des Prozentwertes im Zahlenformat erfolgen muss. Bei Angaben im %-Format würden falsche Ergebnisse geliefert.

Die Funktion `Brutto` wird vom Datentyp `Currency` deklariert. Das bedeutet, dass der Rückgabewert im Währungsformat erfolgt. In einer Zelle wird die Funktion wie folgt aufgerufen: =Brutto(Wert/Bezug;16).

```
'=====
' Auf CD      Buchdaten\Beispiele\01_Einstieg\
' Dateiname   01_08_Functions
' Tabelle     MwSt
' Modul       mdl_03_MwSt
'=====

Function Brutto(dblValue As Double, dblPercent As Double) _
    As Currency
    Brutto = dblValue * (dblPercent + 100) / 100
End Function
```

Umgekehrt werden mit der Formel =107/1,07 7% aus dem Wert 107 herausgerechnet, so dass das Ergebnis 100 lautet. Mit der benutzerdefinierten Funktion `Netto` lautet die Zellen-Formel: =NETTO(107;7).

```
'=====
' Auf CD      Buchdaten\Beispiele\01_Einstieg\
' Dateiname   01_08_Functions
' Tabelle     MwSt
' Modul       mdl_03_MwSt
'=====

Function Netto(dblValue As Double, dblPercent As Double) _
    As Currency
    Netto = dblValue / (dblPercent + 100) * 100
End Function
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Obj
ekte

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

Diese beiden benutzerdefinierten Funktionen können sowohl aus einer Zelle im Tabellenblatt als auch aus einer VBA-Prozedur heraus aufgerufen werden.

Nach Aufruf der Prozedur `MwSt` wird ein Meldungsfenster mit den berechneten Werten angezeigt.



Abbildung 16: Funktionsergebnisse in Meldungsfenster anzeigen

```
'=====
' Auf CD      Buchdaten\Beispiele\01_Einstieg\
' Dateiname  01_08_Functions
' Tabelle     MwSt
' Modul      mdl_03_MwSt
'=====
```

```
Sub MwSt()
    MsgBox "Wert: " & vbTab & "100" & vbCrLf & _
        "Brutto: " & vbTab & Brutto(100, 16) & vbCrLf & _
        "Netto: " & vbTab & Netto(100, 16)
End Sub
```

28 Funktion einer Kategorie zuweisen

Über den Funktionsassistenten können Sie im Tabellenblatt eine beliebige verfügbare Funktion in eine Zelle einfügen. Ab Excel 2002 rufen Sie den Funktionsassistenten über das Symbol *fx* in der Bearbeitungszeile von Excel auf. In vorherigen Versionen ist dieses Symbol in der Grundeinstellung in der STANDARD-Symbolleiste integriert.

Tipp

Über den ANPASSEN-Dialog im Menü ANSICHT | SYMBOLLEISTEN | ANPASSEN Register BEFEHLE, Kategorie EINFÜGEN können Sie aus der rechten Fensterhälfte das Symbol *fx* mit gedrückter linker Maustaste in eine Symbolleiste ziehen.

Sobald Sie in irgendeiner geöffneten Excel-Arbeitsmappe oder einem AddIn eine benutzerdefinierte Funktionen in einem Standardmodul hinterlegt haben, wird in der Kategorien-Liste des Funktionsassistenten die Kategorie BENUTZERDEFINIERT angezeigt (siehe Abbildung 17).

Die Kategorie BENUTZERDEFINIERT enthält alle Prozeduren der aktiven Excel-Anwendung, die als Function definiert sind. Sie unterscheidet nicht zwischen Funktionen, welche Sie zur Anwendung im Tabellenblatt oder zur Verwendung als VBA-Funktion programmiert haben.

Standardmäßig werden benutzerdefinierte Funktionen, wie der Name schon sagt, in der Kategorie BENUTZERDEFINIERT aufgeführt. Über die `MacroOptions`-Methode können Sie benutzerdefinierte Funktionen über den Kategorie-Index einer anderen Kategorie zuweisen. Dieser Vorgang muss nur einmalig je Funktion ausgeführt werden. Die Zuweisung wird mit der Arbeitsmappe gespeichert.



Abbildung 17: Liste der Funktionskategorien

Index	Kategorie
0	Nur in Kategorie ALLE anzeigen (entfernt Funktion aus anderen Kategorien)
1	Finanzmathematik
2	Datum & Zeit
3	Math. & Trigonom
4	Statistik
5	Matrix
6	Datenbank
7	Text
8	Logik
9	Information
10	Menübefehle (ausgeblendet)
11	Benutzerorientiert (ausgeblendet)
12	Makrosteuerung (ausgeblendet)
13	DDE/Extern (ausgeblendet)
14	Benutzerdefiniert
15	Technisch (nur bei installiertem <i>Analyse-Funktionen</i> Add-In)

Tabelle 5: Funktions-Kategorien

Die folgende Funktion `ExcelUserName` ist in einem Standardmodul der Beispieldatei hinterlegt. Sie liefert den im Menü EXTRAS | OPTIONEN Register ALLGEMEIN aufgeführten BENUTZERNAMEN. Die Funktion ist sowohl aus einer Zelle im Tabellenblatt `=ExcelUserName()` als auch aus einer VBA-Prozedur heraus aufrufbar: `MsgBox ExcelUserName`.

Grundlagen

Allgemein

Datum und Zeit

Formeln

Steuerungselemente

Befehle

Objekte

Diagramme

Ereignisse

User-Formen

Web/Mail

Extern

Gemischtes

Spezial

```
' =====
' Auf CD      Buchdaten\Beispiele\01_Einstieg\
' Dateiname   01_08_Functions
' Tabelle     Funktions-Kategorie
' Modul       mdl_04_ChangeCategory
' =====
```

```
Function ExcelUserName()
    ExcelUserName = Application.UserName
End Function
```

Standardmäßig wird die Funktion `ExcelUserName` in der Liste der Kategorie **BENUTZERDEFINIERT** aufgeführt. In der Beispieldatei haben wir die Prozedur `ChangeCategory` einer Schaltfläche zugewiesen. Sie verschiebt die Funktion `ExcelUserName` von der Kategorie **BENUTZERDEFINIERT** in die Kategorie **INFORMATION** (siehe Abbildung 18). Geben Sie für die Kategorie den entsprechenden Index aus Tabelle 5 an.

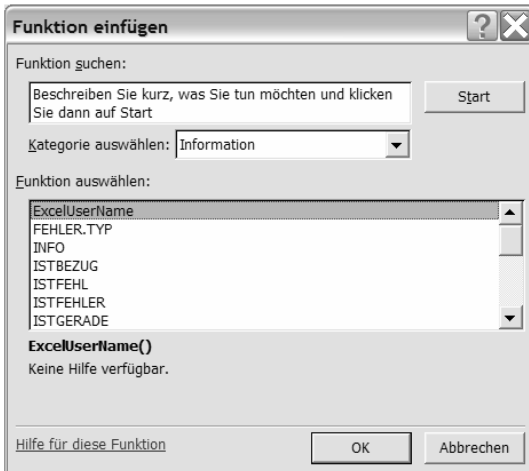


Abbildung 18: Benutzerdefinierte Funktion einer Kategorie zuweisen

```
' =====
' Auf CD      Buchdaten\Beispiele\01_Einstieg\
' Dateiname   01_08_Functions
' Tabelle     Funktions-Kategorie
' Modul       mdl_04_ChangeCategory
' =====
```

```
Sub ChangeCategory()
    Application.MacroOptions _
        Macro:="ExcelUserName", _
        Category:=9 ' Information
End Sub
```

29 Beschreibung einer Funktion hinzufügen

Im Dialog FUNKTION EINFÜGEN (siehe Abbildung 18) sowie im FUNKTIONSSASSISTENTEN wird bei den integrierten Excel-Funktionen ein Hilfetext zu der markierten Funktion angezeigt. Bei benutzerdefinierten Funktionen lautet der Standardtext KEINE HILFE VERFÜGBAR.

Es gibt verschiedene Möglichkeiten, Ihren benutzerdefinierten Funktionen, auch *UDF* (User Defined Functions) genannt, einen aussagekräftigen Hilfetext mitzugeben. Eine weitestgehend unbekannte Methode stellen wir Ihnen als erste Möglichkeit vor.

1. Öffnen Sie über das Menü EXTRAS | MAKRO | AUFZEICHNEN den Dialog MAKRO AUFZEICHNEN (Abbildung 19).
2. Geben Sie in der Zeile MAKRONAME den Funktionsnamen an und notieren Sie im Textfeld BESCHREIBUNG den gewünschten Hilfetext zu der Funktion (Abbildung 19).
3. Klicken Sie auf die Schaltfläche OK (Abbildung 19).
4. Beenden Sie anschließend die Aufzeichnung des Makros sofort wieder.
5. Wechseln Sie in die VBA-Umgebung zum aufgezeichneten Code (Abbildung 19).

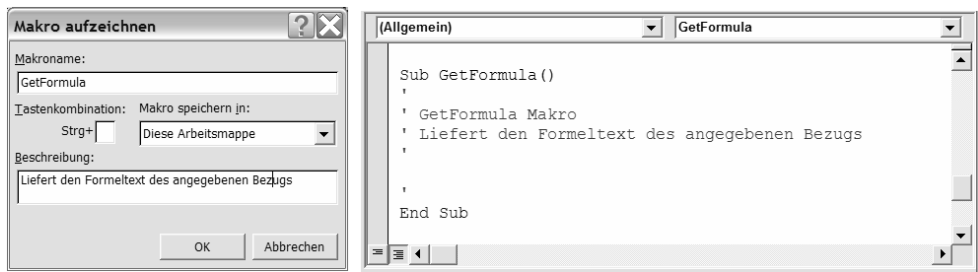


Abbildung 19: Beschreibung einem Makro/Funktion hinzufügen

6. Ersetzen Sie in der Aufrufzeile des Makros das Wort Sub durch Function (Abbildung 20).
7. Ergänzen Sie die Funktion um die erforderlichen Parameter, Deklarierungen und den Funktions-Code (Abbildung 20).

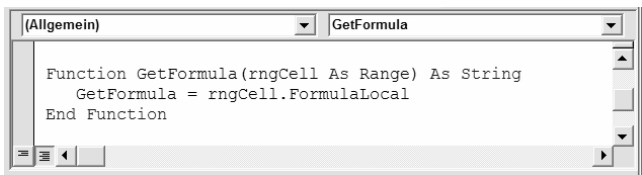


Abbildung 20: Makro in Funktion ändern

8. Im Dialog FUNKTION EINFÜGEN wird der über die Makroaufzeichnung eingegebene Hilfetext zu der Funktion angezeigt (Abbildung 21).

Die in diesem Beispiel angewandte Funktion *GetFormula* liefert, wie im Hilfetext der Funktion bereits angekündigt, den Formeltext des im Klammerpaar angegebenen Bezugs: =GetFormula(A10).

Grundlagen

Allgemein

Datum
Zeit

Formeln

Steuerelemente

Befehle
leisten

Objekte

Diagramme

Ereignisse

User-Form

Web/
Mail

Extern

Gemischtes

Spezial



Abbildung 21: Benutzerdefinierte Funktion mit Beschreibung

```
' =====
' Auf CD      Buchdaten\Beispiele\01_Einstieg\
' Dateiname   01_08_Functions
' Tabelle     Funktions-Beschreibung
' Modul       mdl_05_FunctionDescription
' =====
```

```
Function GetFormula(rngCell As Range) As String
    GetFormula = rngCell.FormulaLocal
End Function
```

Die beschriebene manuelle Vorgehensweise zur Erstellung eines Beschreibungstextes über die Makroaufzeichnung muss ausgeführt werden, bevor Sie Ihrem VBA-Projekt eine neue benutzerdefinierte Funktion hinzufügen.

Um den Hilfetext einer bestehenden Funktion zu erstellen, oder abzuwandeln, ändern Sie das Wort **Function** in **Sub** und entfernen alle Parameter und Deklarierungen der Funktion. Über das Menü **EXTRAS | MAKRO | MAKROS Schaltfläche OPTIONEN** ändern Sie dann den Hilfetext der Funktion, welche zu diesem Zeitpunkt als Makro angesehen wird. Anschließend benennen Sie die Prozedur wieder als **Function** und fügen die zuvor entfernten Parameter wieder hinzu.

Eine weitaus einfachere Methode, einen Hilfetext für Ihre benutzerdefinierten Funktionen zu verfassen oder zu ändern, stellt die in Rezept 28 bereits vorgestellte **MacroOptions**-Methode dar.

```
' =====
' Auf CD      Buchdaten\Beispiele\01_Einstieg\
' Dateiname   01_08_Functions
' Tabelle     Funktions-Beschreibung
' Modul       mdl_05_FunctionDescription
' =====
```

```
Sub FunctionDescription()
    Application.MacroOptions _
        Macro:="GetFormula", _
        Description:="Neue Funktionsbeschreibung"
End Sub
```

Sie können den Kategoriwechsel sowie die Änderung des Beschreibungstextes einer benutzerdefinierten Funktion zusammen in einer Prozedur ausführen. In der folgenden Prozedur wird neben einem festen Text der eingestellte Benutzername über `Application.UserName` im Hilfetext integriert. Die angegebene Funktion wird in die Kategorie `INFORMATION` verschoben.

```
'=====
' Auf CD      Buchdaten\Beispiele\01_Einstieg\
' Dateiname  01_08_Functions
' Tabelle     Funktions-Beschreibung
' Modul      mdl_05_FunctionDescription
'=====

Sub CategoryAndDescription()
    Application.MacroOptions _
        Macro:="GetFormula", _
        Description:="Beschreibung durch " & _
            Application.UserName, _
        Category:=9 ' Information
End Sub
```

30 Pfad, Dateiname und Blattname ermitteln

Über die Informationsfunktion `Zelle` mit dem Infotyp `Dateiname` erhalten Sie den vollständigen Pfad einer Datei sowie den Datei- und Blattnamen des angegebenen Bezugs in einer Zeichenkette. Die Funktion liefert eine leere Textzeichenfolge `" "` zurück, wenn das Tabellenblatt noch nicht gespeichert wurde.

Die Formel `=ZELLE("Dateiname";$A$1)` liefert im Tabellenblatt `TESTBLATT` als Ergebnis zum Beispiel: `C:\Eigene Dateien\Excel\Testmappe.xls\TestBlatt`

Wenn Sie lediglich den Pfad, oder den Datei- bzw. Blattnamen in einer Zelle anzeigen möchten, muss die gewünschte Information durch verschachtelte Funktionen aus dieser Zeichenkette extrahiert werden.

Geben Sie die Formeln aus Tabelle 6 jeweils in die Bearbeitungszeile einer Zelle ein. Die angewandten Zeilenumbrüche sind beabsichtigt und müssen nicht entfernt werden. Sie beeinträchtigen nicht die Funktionsweise und das Ergebnis der Berechnung.

Information	Formel	Ergebnis
Pfad	=LINKS(ZELLE("Dateiname";\$A\$1); FINDEN("[";ZELLE("Dateiname";\$A\$1))-2)	C:\Eigene Dateien\Excel
Dateiname	=TEIL(ZELLE("Dateiname";\$A\$1);FINDEN("["; ZELLE("Dateiname";\$A\$1))+1; FINDEN("]";ZELLE("Dateiname";\$A\$1))- FINDEN("[";ZELLE("Dateiname";\$A\$1))-1)	Testmappe.xls
Blattname	=ERSETZEN(ZELLE("Dateiname";\$A\$1);1; FINDEN("]";ZELLE("Dateiname";\$A\$1));" ")	TestBlatt

Tabelle 6: Verschachtelte Funktionen

Mit VBA ist es wesentlich einfacher, an obige Datei-Informationen zu gelangen. Über die `Caller`-Eigenschaft erhalten Sie Informationen darüber, wie Visual Basic aufgerufen wurde. In einer

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Obj
ekte

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

benutzerdefinierten Tabellenblatffunktion einer einzelnen Zelle liefert sie ein Range-Objekt, das die Zelle angibt, in welcher die Funktion hinterlegt ist.

Analog zu der Funktion =Zelle("Dateiname";\$A\$1") und den herausgefilterten Informationen laut Tabelle 6 haben wir entsprechende benutzerdefinierte Tabellenblatffunktionen programmiert. Die Funktionen können nur in einer Zelle und nicht aus VBA-Prozeduren aufgerufen werden. Geben Sie in einer Zelle ein = gefolgt von dem Funktionsnamen und einem leeren runden Klammerpaar an. Für die Funktion PathAndName schreiben Sie beispielsweise: =PathAndName().

```
' =====
' Auf CD      Buchdaten\Beispiele\01_Einstieg\
' Dateiname  01_08_Functions
' Tabelle     FileInfos
Modul        mdl_06_FileInfos
' =====

' Dateipfad und -name
Function PathAndName() As String
    PathAndName = Application.Caller.Parent.Parent.FullName
End Function

' Dateipfad
Function FilePath() As String
    FilePath = Application.Caller.Parent.Parent.Path
End Function

' Dateiname
Function FileName() As String
    FileName = Application.Caller.Parent.Parent.Name
End Function

' Blattname
Function SheetName() As String
    SheetName = Application.Caller.Parent.Name
End Function
```

Achtung

Wenn Sie eine Excel-Arbeitsmappe mit diesen Funktionen über das Menü DATEI | SPEICHERN UNTER in einen anderen Pfad, oder unter einem anderen Namen speichern, werden die Ergebnisse der Funktionen nicht automatisch aktualisiert, da die Funktionen keine Bezugsangaben enthalten.

Diese Verhaltensweise bezeichnet man als nicht veränderliche Funktionen. Sie werden nur dann neu berechnet, wenn sich die Eingabevariablen ändern.

Gravierend wirkt sich die nicht automatische Neuberechnung aus, wenn der Blattname über die Registerbeschriftung geändert wird. Die Funktion SheetName wird nicht automatisch aktualisiert. Sie müssten mittels eines Doppelklicks auf die Zelle mit der angewandten Funktion in den Bearbeitungsmodus der Zelle wechseln. Erst nachdem Sie die Zelle wieder verlassen haben, wird die Funktion neu berechnet.

Im nachfolgendem Rezept 31 erfahren Sie, wie diese Funktionen dennoch automatisch aktualisiert werden können.

31 Flüchtige Tabellenblattfunktionen

Nach Möglichkeit sollten in einer benutzerdefinierten Tabellenblattfunktion alle veränderbaren Werte und Bezüge über Parameter im Funktionskopf der Funktion übergeben werden. Bei einer Änderung eines Bezugswertes werden die veränderlichen Funktionen automatisch neu berechnet. Werden keine Argumente verwendet, kann die Funktion nicht automatisch neu berechnet werden.

In Microsoft Excel gibt es eine Reihe von integrierten, argumentlosen, aber dennoch flüchtigen Tabellenblattfunktionen. Dazu zählen alle Funktionen, die im Zusammenhang mit Datum `HEUTE()` und Zeit `JETZT()` stehen. Zu nennen ist auch die Funktion `ZUFALLSZAHL()`. Flüchtige Funktionen werden jedes Mal neu berechnet, sobald in der Excel-Anwendung in irgendeinem Blatt irgendeiner Arbeitsmappe irgendein Wert eingegeben oder durch VBA-Code geändert wurde.

Es liegt in der Hand des Anwenders, ebenfalls argumentlose Funktionen zu erstellen und einzusetzen, mit dem kleinen Unterschied, dass sie standardmäßig eben nicht bei der Neuberechnung des Tabellenblattes aktualisiert werden.

Für diese Funktionen ist die `Volatile`-Methode vorgesehen. Schreiben Sie den Befehl `Application.Volatile` in die erste Zeile nach dem Funktionsaufruf. Die Methode löst eine Neuberechnung aller Formeln der Funktion aus, wenn in irgendeiner Zelle der Excel-Anwendung ein Wert geändert wurde. Hat man eine ganze Reihe von flüchtigen Formeln, wird die benutzerdefinierte Funktion für jede Formel einmal durchlaufen. Wenn in einer anderen geöffneten Arbeitsmappe eine Neuberechnung angestoßen wird, werden ebenfalls alle Formeln aus flüchtigen Funktionen in allen offenen Excel-Arbeitsmappen neu berechnet.

Der Umstand der Neuberechnung kann sich bei einer umfangreichen Programmierung oder häufig eingesetzten flüchtigen Funktionen störend auswirken. Excel steht für die Dauer der Neuberechnung sozusagen still.

Achtung

Sie sollten auf keinen Fall benutzerdefinierte Funktionen mit `Application.Volatile` in die Datei `PERSONL.xls` einfügen, sie werden bei jeder Neuberechnung in der gesamten Excel-Anwendung aufgerufen.

Tipp

Stellen Sie die Berechnungsart im Menü `EXTRAS | OPTIONEN Register BERECHNUNG` auf `MANUELL`. Eine Neuberechnung des Tabellenblattes inklusive der mit `Volatile` gekennzeichneten Funktionen erreichen Sie dann nur noch über die Taste `F9`

Am Beispiel der `SheetName`-Funktion aus dem Rezept 28 sieht eine mit der `Volatile`-Methode als flüchtig gekennzeichnete Funktion folgendermaßen aus:

```
'=====
' Auf CD      Buchdaten\Beispiele\01_Einstieg\
' Dateiname   01_08_Functions
' Tabelle     FileInfos
' Modul       mdl_06_FileInfos
'=====

Function SheetNameVolatile() As String
    Application.Volatile
    SheetNameVolatile = Application.Caller.Parent.Name
End Function
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Obj
ekte

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

Anstatt über die `Volatile`-Methode können Sie eine benutzerdefinierte *nicht flüchtige* Funktion über eine integrierte *flüchtige* Excel-Funktion zur Neuberechnung veranlassen. Bei Funktionen, die einen Zahlenwert als Ergebnis liefern, wird die `Jetzt`-Funktion eingesetzt. Multipliziert mit der Zahl 0 addiert sie zu dem Funktionsergebnis den Wert 0: `=Funktion()+0*JETZT()`.

Sobald im Tabellenblatt eine Neuberechnung stattfindet, wird auch die `Jetzt`-Funktion mitsamt der benutzerdefinierten Funktion neu berechnet.

Für die `SheetName`-Funktion (ohne `Volatile`-Methode), welche einen Textwert als Ergebnis liefert, setzen Sie die `Jetzt`-Funktion innerhalb der `Text`-Funktion ein. Durch das angegebene Textformat "" verändert sich das Funktionsergebnis nicht. Verwenden Sie die Formel: `=SheetName()&TEXT(0*JETZT();"").` Änderungen der Registerbeschriftungen werden dadurch automatisch in der Formelzelle aktualisiert.

Diese Methode hat den großen Vorteil, dass sie selektiv eingesetzt werden kann. Schreiben Sie den Formelzusatz hinter den Funktionsnamen nur in die Formelzellen, welche als flüchtig gekennzeichnet werden sollen.

32 Alle Blattnamen einer Arbeitsmappe ermitteln

Die bereits vorgestellte `SheetName`-Funktion liefert nur den Namen des Tabellenblattes, in welchem die Formel hinterlegt ist. Eine Auflistung aller Blattnamen der aktiven Arbeitsmappe erreichen Sie üblicherweise über eine Prozedur. Dabei müssen alle Blätter der Arbeitsmappe durchlaufen und die Blattnamen in die Zellen geschrieben werden.

Eine Übersicht aller Blattnamen können Sie auch mit der folgenden Funktion `FileSheets` erreichen. Übergeben Sie in der Formel den Index des Tabellenblattes, dessen Registerbeschriftung Sie ermitteln möchten. Die Formel `=FileSheets(1)` liefert den Namen des ersten Arbeitsblattes. Wird ein Blattname geändert, wirkt sich die Änderung nicht automatisch auf das Formelergebnis aus. Um die Funktion als flüchtig zu kennzeichnen, setzen Sie entweder die `Volatile`-Methode oder die flüchtige `Jetzt`-Funktion wie in unserer Beispieldatei ein `=FileSheets()&TEXT(0*JETZT();"").`

Im Kopf der Funktion `FileSheets` wird der optionale Parameter `intSheet` vom Datentyp Integer (Ganzzahl) angegeben. Beim Aufruf in einer Formel können Sie stellvertretend für den gewünschten Index eine Zahl eingeben, welche von links die Nummer des Tabellenblattes der Tabellenregister darstellt.

Hinweis

Wenn Sie anstelle einer Zahl einen Textwert als Argument für `intSheet` angeben, liefert die Funktion den Fehler `#WERT!` als Ergebnis, da die Funktion einen Zahlenwert erwartet.

Lassen Sie das Argument `intSheet` zwischen dem Klammerpaar nach dem Funktionsnamen leer `=FileSheets()`, dann wird der Name des Blattes, in welchem die Formel aufgeführt ist, als Ergebnis zurückgegeben. Die Funktion selbst wurde mit dem Datentyp `String` deklariert, da sie als Ergebnis einen Textwert liefert.

```
'=====
' Auf CD      Buchdaten\Beispiele\01_Einstieg\
' Dateiname  01_08_Functions
' Tabelle    FileInfos
' Modul      mdl_06_FileInfos
'=====
```

```
' Blattname des angegebenen Blatt-Index
Public Function FileSheets(Optional intSheet As Integer) As String
    If intSheet = 0 Then
        FileSheets = Application.Caller.Parent.Name
        Exit Function
    End If

    FileSheets = Workbooks(Application.Caller.Parent.Parent.Name) _
        .Sheets(intSheet).Name
End Function
```

33 Existiert ein bestimmtes Tabellenblatt?

Im vorherigen Rezept haben Sie die Blattnamen einer Excel-Arbeitsmappe über eine benutzerdefinierte Funktion ermittelt. Es ist ebenso möglich, das Vorhandensein eines Arbeitsblattes durch einen booleschen Wert Wahr oder Falsch anzeigen zu lassen.

In der Funktion `SheetExists` wird der übergebene Blattnamen mit der `Sheets`-Eigenschaft überprüft. Sie gibt eine `Sheets`-Auflistung zurück, die alle Blätter in der angegebenen oder aktiven Arbeitsmappe darstellt. Die `Sheets`-Auflistung kann `Chart`- oder `Worksheet`-Objekte enthalten. Da der Ausdruck `Sheets(strShName)` ein Objekt darstellt, gibt eine Überprüfung mit `Is Nothing` einen booleschen Wert zurück, der besagt, ob das Objekt in der `Sheets`-Auflistung vorhanden ist.

Wenn das gesuchte Blatt vorhanden ist, soll die Funktion den Wert `Wahr` liefern. Dementsprechend wird bei der Zuweisung eines booleschen Wertes an die Funktion der `Not`-Operator eingesetzt. `Not Is Nothing` liefert die Negation (Umkehrung) des Ausdrucks und bedeutet soviel wie: Ja, das Blatt ist vorhanden. Trifft dies zu, ist das Funktionsergebnis `Wahr`, ansonsten `Falsch`. Der Wahrheitswert des Ausdrucks `Not Worksheets(strShName) Is Nothing` wird also nahtlos als Funktionsergebnis verwendet.

Falls das angegebene Blatt nicht in der `Sheets`-Auflistung vorhanden ist, würde Excel eine Fehlermeldung ausgeben. Dieser Fehler würde durch die vorangestellte `On Error Resume Next`-Fehlerroutine abgefangen.

Tipp

Die Verwendung von Fehler Routinen wird in benutzerdefinierten Funktionen häufig angewandt, da Funktionen oft dazu dienen, Fehler Routinen aus Sub-Prozeduren auszulagern. Zudem kann, wie in unserem Beispiel, der Berechnungsvorgang beschleunigt werden.

Um die Fehler routine zu umgehen, könnten alle Blätter der Arbeitsmappe mittels einer `For Each...Next`-Anweisung durchlaufen werden, um die Existenz eines Blattes zu ermitteln. Stellen Sie sich vor, eine Arbeitsmappe hätte 1000 Blätter und das letzte Blatt entspräche der gesuchten Beschriftung. Die Schleife würde 999 Blätter umsonst durchlaufen, um das letzte Blatt zu finden.

Die Funktion `SheetExists` kann entweder als Tabellenblatt- oder VBA-Funktion eingesetzt werden. Da der `Sheets`-Auflistung kein `Workbook`-Objekt vorangestellt wird, bezieht sich die `Sheets`-Auflistung dieser Funktion immer auf die aktive Arbeitsmappe.

```
'=====
' Auf CD      Buchdaten\Beispiele\01_Einstieg\
' Dateiname  01_08_Functions
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Obj
ekte

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

```
' Tabelle      FileInfos
' Modul        mdl_06_FileInfos
' =====
```

```
Function SheetExists(strShName As String) As Boolean
    On Error Resume Next
    SheetExists = Not Sheets(strShName) Is Nothing
End Function
```

Wenn Sie ein Blatt einer anderen geöffneten Excel-Arbeitsmappe auf deren Existenz überprüfen wollen, setzen Sie die Funktion `FileSheetsExists` ein. Als zweites Argument `strWbk` kann optional der Name einer geöffneten Arbeitsmappe angegeben werden. Wird das Argument weggelassen, wird zu Beginn der Funktion der Name der aktiven Arbeitsmappe an die Variable `strWbk` übergeben.

```
' =====
' Auf CD       Buchdaten\Beispiele\01_Einstieg\
' Dateiname    01_08_Functions
' Tabelle      FileInfos
' Modul        mdl_06_FileInfos
' =====
```

```
Function FileSheetsExists(strShName As String, _
                          Optional strWbk As String) As Boolean
    If IsMissing(strWbk) Then strwbk = ThisWorkbook.Name

    On Error Resume Next
    FileSheetsExists = _
        Not Workbooks(strWbk).Sheets(strShName) Is Nothing
End Function
```

34 Ist eine Arbeitsmappe geöffnet?

Mit der gleichen Programmiertechnik des vorangegangenen Beispiels wird geprüft, ob eine Excel-Arbeitsmappe geöffnet, also in der aktiven Excel-Anwendung vorhanden ist. Anstelle der `Sheets`-Auflistung wird dabei die `Workbooks`-Auflistung verwendet. Sie stellt alle geöffneten Arbeitsmappen dar. Die Funktion `IsWorkbookOpen` kann ebenfalls als Tabellenblatt- oder VBA-Funktion eingesetzt werden.

```
' =====
' Auf CD       Buchdaten\Beispiele\01_Einstieg\
' Dateiname    01_08_Functions
' Tabelle      FileInfos
' Modul        mdl_06_FileInfos
' =====
```

```
Function IsWorkbookOpen(strWB As String) As Boolean
    On Error Resume Next
    IsWorkbookOpen = Not Workbooks(strWB) Is Nothing
End Function
```

35 Aufrufende Formelzelle ermitteln

Um die Adresse einer aufrufenden Formelzelle zu ermitteln, steht Ihnen in allen Excel-Versionen die `Caller`-Eigenschaft zur Verfügung. Sie gibt Informationen darüber zurück, wie Visual Basic aufgerufen wurde. Die Formel `=CallerAddress()` liefert als Ergebnis die Zelladresse, aus der die Formel aufgerufen wurde.

```
' =====
' Auf CD      Buchdaten\Beispiele\01_Einstieg\
' Dateiname  01_08_Functions
' Tabelle     FileInfos
' Modul      mdl_06_FileInfos
' =====

' alle Excel-Versionen
Function CallerAddress()
    CallerAddress = Application.Caller.Address(0, 0)
End Function
```

In der Version Excel 2002 wurde dem Objektkatalog die `ThisCell`-Eigenschaft hinzugefügt. Sie gibt die Zelle zurück, von der aus die benutzerdefinierte Funktion als `Range`-Objekt aufgerufen wurde.

Achtung

Da die `ThisCell`-Eigenschaft in Versionen bis einschließlich Excel 2000 nicht zur Verfügung steht, liefert eine Funktion mit deren Benutzung in diesen Versionen den Fehlerwert `#WERT!`.

```
' =====
' Auf CD      Buchdaten\Beispiele\01_Einstieg\
' Dateiname  01_08_Functions
' Tabelle     FileInfos
' Modul      mdl_06_FileInfos
' =====

' ab Excel 2002
Function ThisCellAddress()
    ThisCellAddress = Application.ThisCell.Address(0, 0)
End Function
```

36 Fehlerwerte in Funktionen

Die `CVErr`-Funktion wird angewandt, um einen Wert vom Typ `Variant` mit dem Untertyp `Error` zurückzugeben, der eine vom Benutzer festgelegte Fehlernummer enthält. Die Tabelle 7 zeigt alle verfügbaren Fehlerwerte.

Konstante	Fehlernummer	Zellfehlerwert
<code>xlErrDiv0</code>	2007	<code>#DIV/0!</code>
<code>XlErrNA</code>	2042	<code>#NV</code>

Tabelle 7: Zellfehlerwerte

Grundlagen

Allgemein

Datum
Zeit

Funktionen

Steuerelemente

Befehle
leisten

Objekte

Diagramm

Ereignisse

User-Form

Web/
Mail

Extern

Gemischtes

Spezial

Konstante	Fehlernummer	Zellfehlerwert
xlErrName	2029	#NAME?
XlErrNull	2000	#NULL!
XlErrNum	2036	#ZAHL!
XlErrRef	2023	#BEZUG!
XlErrValue	2015	#WERT!

Tabelle 7: Zellfehlerwerte (Forts.)

In benutzerdefinierten Funktionen können Sie als Ergebnis einen Fehlerwert anhand einer Konstante oder Fehlernummer an die Funktion übergeben:

```
Funktionsname= CVerErr(Konstante)
```

Folgende Prozedur schreibt alle Fehlerwerte ab der Zelle A2 in das aktive Tabellenblatt. Der Schleifenzähler `intI` beginnt bei 0, da das Datenfeld ebenfalls auf 0 basiert.

```
'=====
' Auf CD      Buchdaten\Beispiele\01_Einstieg\
' Dateiname  01_08_Functions
' Tabelle     CVerErr
' Modul      mdl_07_CVerErr
'=====

Sub CVerErrList()
    Dim varErr As Variant
    Dim intI As Integer

    varErr = Array(xlErrDiv0, xlErrNA, xlErrName, xlErrNull, _
                   xlErrNum, xlErrRef, xlErrValue)

    For intI = 0 To 6
        ActiveSheet.Cells(2 + intI, 1).Value = CVerErr(varErr(intI))
    Next intI
End Sub
```

Über die `IsError`-Funktion können Sie ermitteln, ob eine Zelle einen Fehlerwert enthält (`IsError(Ausdruck)`). Es ist aber nicht möglich, die Fehlernummer bzw. die Konstante eines Fehlerwertes einer Zelle direkt auszulesen. Enthält eine Zelle einen Fehlerwert, wird die Zeichenfolge Fehler xxxx zurückgegeben. Der Zellenfehlerwert muss zunächst mit der `CVerErr`-Funktion in eine Fehlernummer xxxx oder Konstante `XlErrxxx` umgewandelt werden. Danach können Sie entsprechende Rückgabewerte bestimmen.

In der folgenden Funktion `CVerErrNumber` wird die Fehlernummer eines Zellbezugs `=CVerErrNumber(A2)` bzw. eines Ausdrucks `=CVerErrNumber(Zeichenkette)` ermittelt (siehe Abbildung 22). Enthält das Argument `varError` keinen Fehlerwert, wird als Ergebnis 0 zurückgegeben, da die Funktion vom Datentyp Integer deklariert wurde.

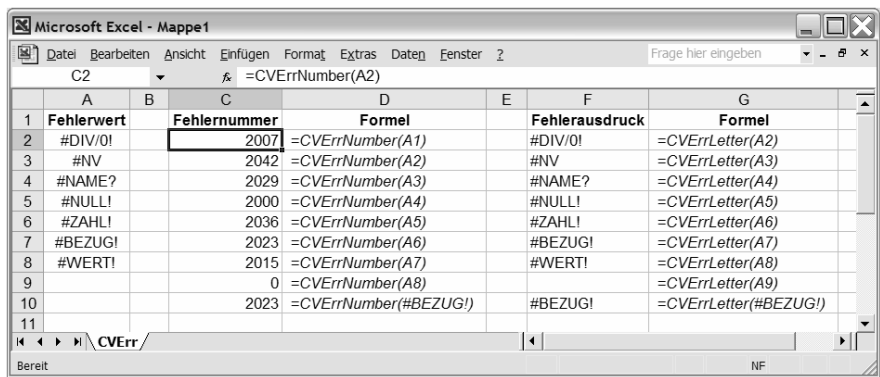


Abbildung 22: Zellfehlerwerte analysieren

In einer For...Next-Anweisung werden alle Elemente des Datenfeldes varErrConst mit der Variablen varError verglichen. Bei Übereinstimmung wird das entsprechende Element des zweiten Arrays varErrNumber als Ergebnis an die Funktion zurückgegeben.

```
'=====
' Auf CD      Buchdaten\Beispiele\01_Einstieg\
' Dateiname  01_08_Functions
' Tabelle     CVerri
' Modul      mdl_07_CVerri
'=====

Function CVerriNumber(varError As Variant) As Integer
    Dim varErrConst As Variant
    Dim varErrNumber As Variant
    Dim intI As Integer

    varErrConst = Array(xlErrDiv0, xlErrNA, xlErrName, xlErrNull, _
                        xlErrNum, xlErrRef, xlErrValue)

    varErrNumber = Array(2007, 2042, 2029, 2000, 2036, 2023, 2015)

    ' Wenn Bezug keinen Fehlerwert enthält, verlasse Funktion
    If Not IsError(varError) Then Exit Function

    For intI = 0 To UBound(varErrConst)
        If varError = CVerri(varErrConst(intI)) Then
            CVerriNumber = varErrNumber(intI)
            Exit Function
        End If
    Next intI
End Function
```

Wenn Sie statt der Fehlernummer den Ausdruck des Fehlerwertes zurückgeben wollen (siehe Abbildung 22), müssen Sie zum einen den Funktionstyp in String ändern, da ein Textwert zu erwarten ist. Anstelle der Fehlernummern vermerken Sie die entsprechenden Fehlerausdrücke im

- Grundlagen
- Allgemein
- Datum
- Zeit
- tungen
- Steuer
- elementen
- Befehl
- leisten
- Objekte
- Diagramm
- Ereignisse
- User-Form
- Web/Mail
- Extern
- Gemischtes
- Spezial

zweiten Datenfeld `varErrLetter`. Wenn `varError` keinen Fehlerwert enthält, wird eine leere Zeichenfolge zurückgegeben.

```
'=====
' Auf CD      Buchdaten\Beispiele\01_Einstieg\
' Dateiname   01_08_Functions
' Tabelle      CVerErr
' Modul       mdl_07_CVerErr
'=====

Function CVerErrLetter(varError As Variant) As String
    Dim varErrConst As Variant
    Dim varErrLetter As Variant
    Dim intI As Integer

    varErrConst = Array(xlErrDiv0, xlErrNA, xlErrName, xlErrNull, _
                        xlErrNum, xlErrRef, xlErrValue)

    varErrLetter = Array("#DIV/0!", "#NV", "#NAME?", "#NULL!", _
                        "#ZÄHL!", "#BEZUG!", "#WERT!")

    If Not IsError(varError) Then Exit Function

    For intI = 0 To UBound(varErrConst)
        If varError = CVerErr(varErrConst(intI)) Then
            CVerErrLetter = varErrLetter(intI)
            Exit Function
        End If
    Next intI
End Function
```

37 Spaltenbuchstabe ermitteln

Im Microsoft Excel gibt die integrierte Tabellenblatffunktion `SPALTE` die Spaltennummer eines angegebenen Bezugs zurück. Fehlt das Argument `Bezug`, wird als Bezug die Zelle angenommen, in der die `SPALTE`-Funktion steht. In der Zelle A1 liefert die Formel `=SPALTE()` den Wert 1, `=SPALTE(D1)` gibt den Wert 4 zurück.

Analog zur Funktion `SPALTE` gibt es keine Funktion, welche zu einem Bezug die Spalte in Ziffern zurückgibt, oder eine Zahl in einen Spaltenbuchstaben umwandelt. Um den Spaltenbuchstaben eines Zellbezugs zu ermitteln, müssten Sie eine umfangreiche, verschachtelte Tabellenfunktion ausarbeiten, oder sich auf einfache Weise mit der folgenden benutzerdefinierten Funktion `ColumnLetter` behelfen.

Da die Funktionsweise von `ColumnLetter` der Funktion `SPALTE` entsprechen soll, haben wir im Funktionskopf den optionalen Parameter `varColNumber` vom Datentyp `Variant` deklariert. Obwohl das Argument `varColNumber` eine Zahl zwischen 1 und 256 für die verfügbaren Spalten erwartet, muss der Parameter `varColNumber` als `Variant` deklariert werden. Würde der Typ `Integer` angewandt, würde die Variable bei fehlender Bezugsangabe den Wert 0 erhalten. Dadurch würde in der `If...Then...Else`-Anweisung die Funktion vorzeitig mit dem Fehlerwert `#WERT!` als Ergebnis beendet.

Wenn Sie keinen Bezug innerhalb der runden Klammern im Formelaufzuruf angeben =ColumnLetter(), wird die Spalte, in welcher die Formel hinterlegt ist, als Grundlage für das Funktionsergebnis angenommen. Geben Sie in der Formel eine Zahl zwischen 1 und 256 an =ColumnLetter(2), wird der entsprechende Spaltenbuchstabe B zurückgegeben.

Die Funktion ColumnLetter muss mit dem Datentyp Variant deklariert werden, da sie entweder einen String-Datentyp, in Form des Spaltenbuchstabens, oder einen Variant-Datentyp durch den Fehlerwert liefern kann.

Über die Address-Eigenschaft der angegebenen Spaltennummer Column erhalten Sie eine Zeichenfolge, welche die gesamte Spalte darstellt. Für Spalte 1 zum Beispiel A:A. Mittels der Split-Funktion teilen Sie diese Zeichenkette in ein nullbasiertes, eindimensionales Datenfeld auf. Als Trennzeichen geben Sie den Doppelpunkt : an. Die Variable varArr(0) enthält den ersten Zeichenfolgenausdruck A. varArr(1) enthält den zweiten Ausdruck, also ebenfalls A. Da beide Variablen den gleichen Wert aufweisen, können Sie eine dieser Variablen als Ergebnis an die Funktion ColumnLetter übergeben.

Hinweis

Die Split-Funktion wurde erst in Excel 2000 eingeführt. Frühere Excel-Versionen unterstützen diese Eigenschaft nicht.

```
'=====
' Auf CD      Buchdaten\Beispiele\01_Einstieg\
' Dateiname   01_08_Functions
' Tabelle      ColumnLetter
' Modul        mdl_08_ColumnLetter
'=====

Function ColumnLetter(Optional varColNumber As Variant) As Variant
    Dim varArr As Variant

    If IsMissing(varColNumber) Then _
        varColNumber = Range(Application.Caller.Address(0, 0)).Column

    If varColNumber < 1 Or varColNumber > 256 Then
        ColumnLetter = CVErr(xlErrValue)
        Exit Function
    End If

    ' liefert für Spalte 1 ="A:A"
    varArr = Split(Columns(varColNumber).Address(0, 0), ":")

    ColumnLetter = varArr(0)
End Function
```

38 Funktion Seitenzahl

In den Kopf- und Fußzeilen können Sie die Seitenzahl im Ausdruck vermerken. Standardmäßig bietet Microsoft Excel keine Möglichkeit, die jeweiligen Seitenzahlen in den Zellen zu hinterlegen. Ebenfalls gibt es keine integrierte VBA-Funktion, um zu einem angegebenen Bezug die Seitenzahl im Ausdruck zu ermitteln. Lediglich über das Menü ANSICHT | SEITENUMBRUCHVORSCHAU werden die Seitenzahlen angezeigt (siehe Abbildung 23).

Grundlagen
Allgemein
Datum und Zeit
Formeln
Diagramme
Ergebnisse
Benutzerdefinierte
Web/Mail
Extern
Gemischtes
Spezial

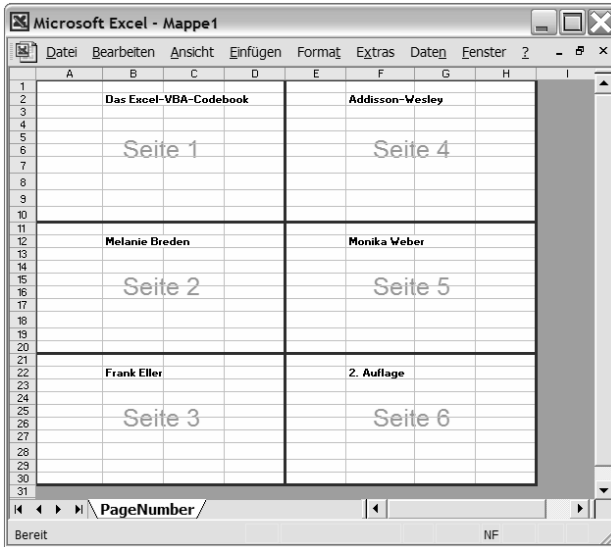


Abbildung 23: Anzeige der Seitenzahlen in der Seitenumbruchvorschau

Die Funktion `PageNumber` können Sie entweder als Formel aus einer Zelle oder über eine VBA-Prozedur aufrufen.

Im Tabellenblatt liefert die Formel `=PAGENUMBER()` jeweils die Seitenzahl der aufrufenden Zelle, in der die Formel steht (siehe Abbildung 24), oder die Seitenzahl der Zelle im angegebenen Bezug `=PAGENUMBER(A1)`. Der Bezug kann auch auf ein anderes Tabellenblatt verweisen. Auf der Seite der Bezugsangabe muss aber mindestens in einer Zelle ein Wert stehen, so dass sie ausgedruckt würde.

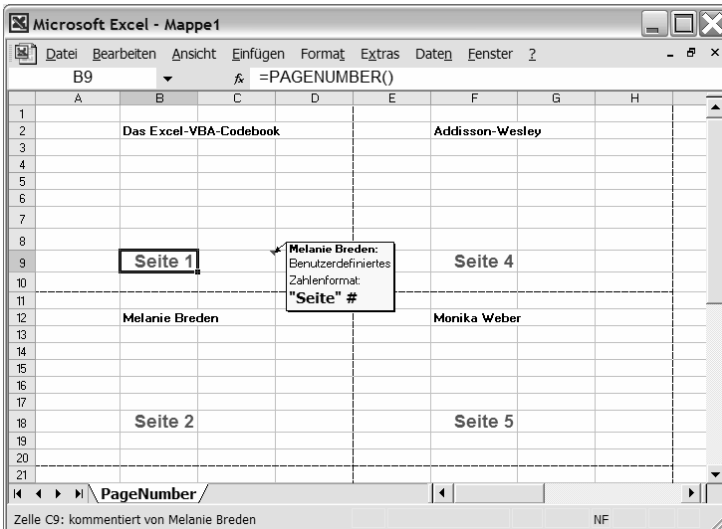


Abbildung 24: Anzeige der Seitenzahlen im Tabellenblatt

Die Funktionsweise der Prozedur entnehmen Sie den im Code hinterlegten Kommentaren.

```
'=====
' Auf CD      Buchdaten\Beispiele\01_Einstieg\
' Dateiname  01_08_Functions
' Tabelle     PageNumber
' Modul      mdl_09_PageNumber
'=====

Function PageNumber(Optional rngCell As Range) As Integer
    Dim wks As Worksheet
    Dim VPB As VPageBreak
    Dim HPB As HPageBreak
    Dim intVPB As Integer
    Dim intHPB As Integer

    ' Bei fehlendem Bezug nehme die aufrufende Zelle als Bezug
    If rngCell Is Nothing Then Set rngCell = Application.Caller

    ' Datei und Blatt des Bezugs ermitteln
    Set wks = Workbooks(rngCell.Parent.Parent.Name). _
        Worksheets(rngCell.Parent.Name)

    ' Reihenfolge der Nummerierung ermitteln
    If wks.PageSetup.Order = xlDownThenOver Then
        intHPB = wks.HPageBreaks.Count + 1
        intVPB = 1
    Else
        intVPB = wks.VPageBreaks.Count + 1
        intHPB = 1
    End If

    ' Vertikalen Seitenumbruch des Bezugs ermitteln
    For Each VPB In wks.VPageBreaks
        If VPB.Location.Column > rngCell.Column Then Exit For
        PageNumber = PageNumber + intHPB
    Next VPB

    ' Horizontalen Seitenumbruch des Bezugs ermitteln
    For Each HPB In wks.HPageBreaks
        If HPB.Location.Row > rngCell.Row Then Exit For
        PageNumber = PageNumber + intVPB
    Next HPB

    ' Rückgabewert an Funktion übergeben
    PageNumber = PageNumber + 1
End Function
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Obj
ekte

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

Nachfolgend zeigen wir Ihnen, wie Sie die Funktion PageNumber aus einer Prozedur heraus aufrufen können.

Wenn Sie die folgende Prozedur GetPageNumber über die Schaltfläche im Tabellenblatt der Beispieldatei aufrufen, wird zunächst ein Eingabe-Dialog InputBox angezeigt. Durch den angegebe-

nen Type-Parameter 8 ist es möglich, eine Zelle im Tabellenblatt anzuklicken. Diese Zelle wird an die Variable `rngCell` übergeben. Im anschließenden Meldungsfenster wird über die benutzerdefinierte Funktion `PageNumber` die Seitenzahl ermittelt und angezeigt (siehe Abbildung 25).

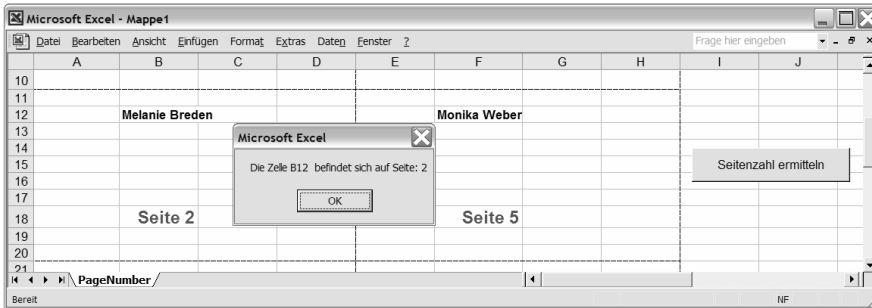


Abbildung 25: Seitenzahl über Prozedur ermitteln

```
' =====
' Auf CD      Buchdaten\Beispiele\01_Einstieg\
' Dateiname   01_08_Functions
' Tabelle     PageNumber
' Modul       mdl_09_PageNumber
' =====
```

```
Sub GetPageNumber()
    Dim rngCell As Range

    Set rngCell = Application.InputBox( _
        Prompt:="Wählen Sie eine Zelle aus.", _
        Title:="Seite ermitteln", _
        Type:=8)

    MsgBox "Die Zelle " & rngCell.Address(0, 0) & _
        " befindet sich auf Seite: " & _
        PageNumber(rngCell)
End Sub
```

39 Formeln mit Zirkelbezügen berechnen

Eine Formel kann Werte oder Zellbezüge zu anderen Zellen enthalten, die nach Berechnung der Formel als Ergebnis in der Formelzelle angezeigt werden. Das folgende Beispiel können Sie in der Beispieldatei im Tabellenblatt ZIRKELBEZUG austesten.

fx =3+10	
B	
Formel-Berechnung	
13	

Abbildung 26: Formel mit Werten

C2		fx =A2+B2	
	A	B	C
1	Eingabezelle	Eingabezelle	Formel-Berechnung
2	3	10	13
3			
4			

Abbildung 27: Formel mit Zellbezügen

Problematisch wird es, wenn sich eine Formel direkt oder indirekt auf ihre eigene Zelle bezieht.

B2=A2+B2

Dies wird als Zirkelbezug bezeichnet und es wird ein Meldungsfenster mit dem Hinweis eingeblendet, dass eine Zirkelbezug vorliegt. Nachdem Sie die Schaltfläche OK gedrückt haben, wird ein Hilfenfenster mit weiteren Informationen zum Thema eingeblendet.

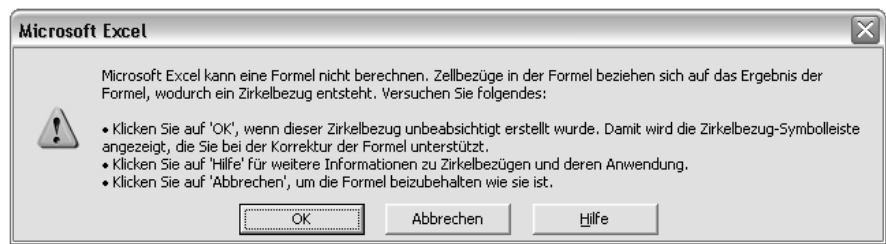


Abbildung 28: Warnhinweis bei Zirkelbezügen

Im Tabellenblatt wird die Symbolleiste ZIRKELVERWEIS angezeigt. Hier können Sie sich die betroffenen Zellbezüge zum Zirkelbezug anzeigen lassen. Durch die Optionen SPUR ZUM VORGÄNGER und SPUR ZUM NACHFOLGER werden durch Pfeilmarkierungen die betroffenen Bezüge auf dem Tabellenblatt dargestellt. Die Symbolleiste ZIRKELVERWEIS wird bei einem Zirkelbezug in allen offenen Excel-Arbeitsmappen angezeigt und listet die festgestellten Zellbezüge in einem Auswahlfeld auf. In der Statuszeile wird ebenfalls die Adresse des Zirkelbezugs des aktiven Tabellenblattes angezeigt.

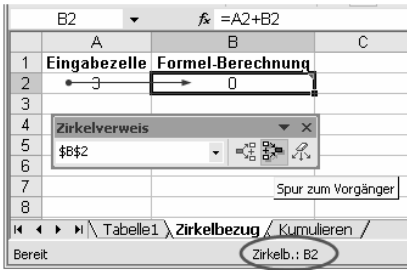


Abbildung 29: Symbolleiste Zirkelverweis

Um diesen Zirkelbezug dennoch zu berechnen, aktivieren Sie im Menü EXTRAS | OPTIONEN | BERECHNUNG die Option ITERATION und stellen den Wert MAXIMALE ITERATIONSZAHL auf den Wert 1. Der Zirkelbezug wird aufgehoben und die Berechnung der Formel durchgeführt. Bei jeder Neuberechnung der Arbeitsmappe wird ebenfalls eine Neuberechnung der Formel vorgenommen, was zu falschen Ergebnissen führt. Deaktivieren Sie deshalb die automatische Neuberechnung, indem Sie die Option MANUELL aktivieren, und entfernen Sie den Haken vor VOR DEM SPEICHERN NEU BERECHNEN. Eine Neuberechnung der gesamten Arbeitsmappe erreichen Sie dann über die Taste **[F9]**.

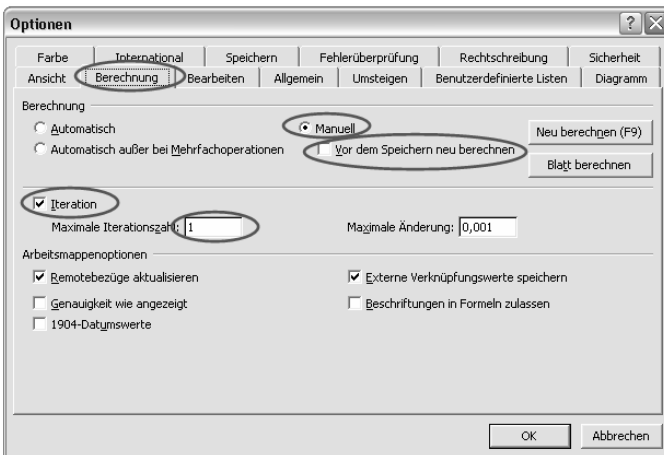


Abbildung 30: Einstellungen zur Zirkelberechnung

Nachdem Sie die Einstellungen laut Abbildung 30 vorgenommen haben, verschwindet die Symbolleiste ZIRKELVERWEIS und die Formel wird berechnet.

	B2		f _x =A2+B2
	A	B	C
1	Eingabezelle	Formel-Berechnung	
2	10	13	
3			
4			
5			

Abbildung 31: Formelberechnung trotz Zirkelbezug

Geben Sie jetzt in die Zelle A2 eine neue Zahl ein, so wird die Formel in Zelle B2 erst durch eine manuelle Neuberechnung F9 aktualisiert. Dieses Verhalten wirkt sich natürlich auf alle weiteren Formeln in der Tabelle aus.

Mit VBA können Sie ebenfalls die ITERATION einstellen. Bei den folgenden Makros ist es wichtig, die Reihenfolge der Befehle einzuhalten, da das Ergebnis der Formel sonst durch Mehrfachberechnungen verfälscht würde.

Im Tabellenblatt ZIRKELBEZUG werden über zwei Schaltflächen die Makros `IterationOn` zur Aktivierung und `IterationOff` zur Deaktivierung der Iteration ausgeführt.

```
'=====
' Auf CD      Buchdaten\Beispiele\02_Allgemein\
' Dateiname  02_01_SumMyself.xls
' Tabelle     Zirkelbezug
' Modul      mdl_01_Iteration
'=====

Sub IterationOn()
    With Application
        .Iteration = True
        .Calculation = xlCalculationManual
        .MaxIterations = 1
        .MaxChange = 0.001
    End With
End Sub

Sub IterationOff()
    With Application
        .Iteration = False
        .Calculation = xlAutomatic
    End With
End Sub
```

40 VBA-Addition mit eigenem Zellenwert

Dass Excel nicht dafür konzipiert ist, mit Zirkelbezügen zu arbeiten, haben Sie in dem vorherigen Beispiel an den vielen Einstellungen, die vorgenommen werden müssen, gesehen. Ebenso sind die weiteren Nachteile, wie manuelle Neuberechnung, oft nicht erwünscht. Mit VBA ist es ohne weiteres möglich, eine Berechnung, welche sich direkt oder indirekt auf ihre eigene Zelle bezieht, ohne die Verwendung der Iteration vorzunehmen. Zum direkten Vergleich verwenden wir das Formelbeispiel aus Rezept 39. Dieses Beispiel befindet sich im Tabellenblatt ZIRKELBEZUG der Beispieldatei. Bei einer Änderung der Eingabezelle E2 soll der Wert der Zelle F2 zu dem neuen Wert aus E2 addiert werden. Dazu setzen wir das `Worksheet_Change`-Ereignis der Tabelle ein.

Da die Prozedur nur ausgeführt werden soll, wenn sich der Wert der Zelle E2 ändert, wird zu Beginn der Prozedur ermittelt, ob das Ereignis durch eine Änderung des Wertes in Zelle E2 ausgelöst wurde. Die Ereignis-Prozedur wird beendet, wenn ein anderer Zellbezug der Auslöser für das Aufrufen des Ereignisses war.

In der Prozedur sollen Zahlenwerte berechnet werden. Bei Texteingaben in Zelle E2 würde Excel versuchen, einen Zahlenwert mit einem Textwert zu berechnen, was zu einer Fehlermeldung führen würde. Mit der `IsNumeric`-Funktion wird überprüft, ob es sich bei der Eingabe um eine Zahl

Grundlagen

Allgemein

Datum
Zeit

Formeln

Steuerelemente

Befehle
leisten

Objekte

Diagramme

Ereignisse

User-Formen

Web/
Mail

Extern

Gemischtes

Spezial

oder Text handelt. Bei einem Textwert erfolgt ein Meldungsfenster und die Eingabezelle wird daraufhin wieder markiert und die Prozedur verlassen.

Ist die Eingabe wie erwartet ein Zahlenwert, wird der Wert der Zelle E2 zu dem Wert der Zelle F2 hinzuaddiert und in die Zelle F2 geschrieben.

Schalten Sie vor dem Schreiben des Wertes in Zelle F2 alle Ereignisse mit der `EnableEvents`-Eigenschaft aus, damit das `Worksheet_Change`-Ereignis nicht nochmals ausgelöst und der Zellenwert aus E2 erneut zu F2 addiert wird. Anschließend muss die `EnableEvents`-Eigenschaft unbedingt wieder eingeschaltet werden, da sonst in der gesamten Excel-Anwendung alle Ereignis-Prozeduren bis auf Widerruf oder Excel-Neustart deaktiviert bleiben.

```
'=====
' Auf CD      Buchdaten\Beispiele\02_Allgemein\
' Dateiname  02_01_SumMyself.xls
' Tabelle    Zirkelbezug
' Ereignis   Tabelle1 (Zirkelbezug)
'=====
```

```
Private Sub Worksheet_Change(ByVal Target As Range)
    If Target.Address <> "$E$2" Then Exit Sub

    If Not IsNumeric(Target.Value) Then
        MsgBox "Eingabe muss eine Zahl sein", _
            vbCritical, _
            "Eingabefehler"

        Target.Select
        Exit Sub
    End If

    Application.EnableEvents = False
    Range("F2").Value = Range("F2").Value + Target.Value
    Application.EnableEvents = True
End Sub
```

41 Zellenwert kumulieren

Im Beispiel 39 haben wir Ihnen gezeigt, dass es mit der Funktion `ITERATION` möglich ist, Formeln zu erstellen, welche zusätzlich zu Werten aus Bezügen ihren eigenen Zellenwert mitberechnen können. Das geht mit einer einzigen Zelle allerdings nicht. Sie können in *eine* Zelle entweder einen Wert bzw. Text *oder* eine Formel eingeben. Angenommen, Sie haben in Zelle A1 den Wert 20 stehen und wollen zu diesem Wert die Zahl 10 addieren. Entweder rechnen Sie das Ergebnis im Kopf oder mit einem Taschenrechner aus und ersetzen in A1 den Wert 20 durch 30, oder Sie verwenden in Zelle A1 die Formel `=20+10`.

Mit Hilfe einiger VBA-Befehle kann man ganz bequem Zahlen nacheinander in *eine* Zelle eingeben, welche dann mit dem vorherigen Wert in dieser Zelle verrechnet werden.

Im Tabellenblatt `KUMULIEREN` der Beispieldatei werden alle Werte, die in die Zelle F13 eingegeben werden, addiert. Enthält die Zelle F13 den Wert 20 und Sie überschreiben diesen Zellenwert durch den Wert 10, wird in Zelle F13 der berechnete Wert 30 angezeigt. Ebenso werden negative Eingaben vom Vorwert abgezogen. Das Wichtigste ist hierbei, dass Excel sich den vorherigen Wert *mer-*

ken muss, um ihn mit dem neuen Wert verrechnen zu können. Dazu wird der Wert der Zelle F13 beim Öffnen der Datei an die öffentliche Variable `gdblRemember` übergeben.

```
'=====
' Auf CD      Buchdaten\Beispiele\02_Allgemein\
' Dateiname  02_01_SumMyself.xls
' Ereignis    DieseArbeitsmappe
'=====

Private Sub Workbook_Open()
    gdblRemember = Worksheets("Kumulieren").Range("F13").Value
End Sub
```

Die Deklaration dieser öffentlichen Variablen wird in einem allgemeinen Modul hinterlegt, damit alle Prozeduren dieser Arbeitsmappe auf die Variable `gdblRemember` und deren Wert Zugriff haben. Der jeweilige Wert bleibt auch nach Beendigung der Prozeduren im Speicher erhalten.

```
'=====
' Auf CD      Buchdaten\Beispiele\02_Allgemein\
' Dateiname  02_01_SumMyself.xls
' Modul      mdl_02_Deklare
'=====

Public gdblRemember As Double
```

Bei jeder Zelländerung im Tabellenblatt KUMULIEREN wird das `Worksheet_Change`-Ereignis ausgelöst. Hier wird mittels `Target.Address` geprüft, ob die Zelle F13 geändert wurde, da nur die Werte dieser Zelle kumuliert werden sollen. Die Prozedur wird beendet, wenn eine andere Zelle der Auslöser für das Ereignis ist.

Bevor wir zum eigentlichen Rechenvorgang kommen, müssen noch mögliche Fehlerquellen, wie zum Beispiel Texteingaben oder das Löschen des Zellenwertes, abgefangen werden. `Target.Value` weist den Wert der Zelle F13 auf und wird im Code dazu verwendet, um die gemachte Eingabe auszuwerten. Wurde der Zellinhalt gelöscht, enthält `Target.Value` einen Leerstring (»«). Die Variable `gdblRemember` erhält infolgedessen den Wert 0 und die Prozedur wird verlassen.

Ist die Eingabe keine Zahl, wird mit der `Undo`-Methode die letzte Benutzeraktion vor der Ausführung des Makros, also die Eingabe in Zelle F13, rückgängig gemacht und anschließend die Prozedur verlassen. Der Wert der Variablen `gdblRemember` bleibt dabei unverändert.

Die `Undo`-Methode muss im Makro aufgerufen werden, *bevor* durch das Makro Änderungen im Tabellenblatt vorgenommen werden. Visual-Basic-Befehle können mit dieser Methode nicht rückgängig gemacht werden.

Hinweis

Um Visual-Basic-Befehle rückgängig zu machen, verwendet man die `OnUndo`-Methode. Weitere Informationen und Beispiele zu dieser Methode finden Sie in der Kategorie »Ereignisse«.

Ist die Eingabe eine Zahl gewesen, dann wird zu dem Eingabewert der Wert der Variablen `gdblRemember` addiert und in die Zelle F13 geschrieben. Dieser neue Wert wird für die nächste Berech-

Grundlagen
Allgemein
Datum und Zeit
Formeln
Diagramme
Steuerungselemente
Befehlsliste
Objekte
Diagramm
Ereignisse
User-Form
Web/Mail
Extern
Gemischtes
Spezial

nung an der Variable `gdb1Remember` übergeben. Um fortlaufend Eingaben in die Zelle F13 machen zu können, wird als letzter Befehl in der Prozedur diese Zelle wieder selektiert.

```
'=====
' Auf CD      Buchdaten\Beispiele\02_Allgemein\
' Dateiname  02_01_SumMyself.xls
' Ereignis   Tabelle2 (Kumulieren)
'=====

Private Sub Worksheet_Change(ByVal Target As Range)
    If Target.Address <> "$F$13" Then Exit Sub

    If Target.Value = "" Then
        gdb1Remember = 0
        Exit Sub
    End If

    If Not IsNumeric(Target.Value) Then
        With Application
            .EnableEvents = False
            .Undo
            .EnableEvents = True
        End With

        Exit Sub
    End If

    Application.EnableEvents = False
    Target.Value = Target.Value + gdb1Remember
    Application.EnableEvents = True

    gdb1Remember = Target.Value
    Target.Select
End Sub
```

42 Welche Zellen sind gesperrt?

Standardmäßig sind alle Zellen eines Tabellenblattes gesperrt und die Option **AUSGEBLENDET** ist deaktiviert. Diese Einstellungen können Sie im Dialog **ZELLEN FORMATIEREN** einsehen und einstellen. Sie erreichen diesen Dialog über das Menü **FORMAT | ZELLEN** Registerkarte **SCHUTZ** oder im **KONTEXTMENÜ** der Zelle über **ZELLE FORMATIEREN** oder über die Tastenkombination **Strg+1**.

Die Option **GESPERRT** verhindert, dass markierte Zellen geändert, verschoben, in ihrer Größe verändert oder gelöscht werden. Die Option **AUSGEBLENDET** blendet die Formel in einer Zelle aus, so dass diese beim Markieren der Zelle nicht in der Bearbeitungszeile angezeigt wird.

Das Sperren und Ausblenden von Zellen ist nur bei geschützten Blättern wirksam. Um ein Tabellenblatt zu schützen, aktivieren Sie im Menü **EXTRAS | SCHUTZ | BLATT** das Kontrollkästchen **INHALTE**.

Standardmäßig ist es in Excel nicht möglich, den Sperrstatus einer Zelle per Formel ausgeben zu lassen. Aber mit Hilfe eines **Excel4-Makros** oder einer benutzerdefinierten VBA-Funktion können Sie dies erreichen.



Abbildung 32: Standardeinstellung des Zellschutzes

Für die Excel4-Makro-Lösung definieren Sie unter EINFÜGEN | NAMEN | DEFINIEREN den Namen Zelle_gesperrt und schreiben in die Zeile BEZIEHT SICH AUF folgende Formel:

```
=ZELLE.ZUORDNEN(14;INDIREKT("ZS(-1)";FALSCH))
```

Tragen Sie die Formel =Zelle_gesperrt in eine Zelle ein, um den Sperr-Status der links neben der Formelzelle liegenden Zelle als Ergebnis der Formel ausgeben zu lassen. Die Formel liefert den Wert WAHR, wenn die Zelle gesperrt ist, FALSCH, wenn sie nicht gesperrt ist.

Um das gleiche Ergebnis über eine benutzerdefinierte Funktion zu erhalten, schreiben Sie die Funktion IsLocked in ein Modul. Den Sperr-Status der Zelle A10 ermitteln Sie, indem Sie in eine Zelle die Formel =IsLocked(A10) eingeben.

Durch die Volatile-Methode wird die Funktion als flüchtig gekennzeichnet. Solche Funktionen werden immer neu berechnet, wenn in einer beliebigen Zelle des Tabellenblattes eine Berechnung durchgeführt wird. Nicht flüchtige Funktionen werden nur dann neu berechnet, wenn sich die Eingabevariablen ändern. Diese Methode wird in der Regel in einer benutzerdefinierten Funktion zur Berechnung einer Tabellenzelle verwendet, andernfalls hat sie keine Auswirkung.

Alleine durch die Änderung der Zellsperre wird kein Ereignis ausgelöst, das die Neuberechnung der Formel auslösen würde. Aktualisieren Sie die Formelergebnisse dieser Funktion, indem Sie die Tastenkombination [F9] drücken. Dadurch erfolgt eine Neuberechnung des Tabellenblattes.

Der in der Formel =IsLocked(A10) angegebene Bezug wird über die Variable rngCell an die Funktion übergeben. Der Ausdruck rngCell.Locked liefert den Sperr-Zustand WAHR oder FALSCH der Zelle A10. Das Ergebnis vom Datentyp Boolean wird an die Funktion IsLocked übergeben, welche dementsprechend als Ergebnis WAHR oder FALSCH zurückgibt.

```
'=====
' Auf CD      Buchdaten\Beispiele\02_Allgemein\
' Dateiname  02_02_ CellProtection.xls
' Modul      mdl_01_IsLocked
'=====
```

```
Function IsLocked(rngCell As Range)
    Application.Volatile
    IsLocked = rngCell.Locked
End Function
```

43 Gesperrte Zellen einfärben

Über die `IsLocked`-Funktion aus dem vorherigen Beispiel und der bedingten Formatierung können Sie geschützte Zellen farblich hervorheben. Markieren Sie einen Bereich, welchen Sie auf den Zellschutz überprüfen wollen, und rufen Sie den Dialog **BEDINGTE FORMATIERUNG** über das Menü **FORMAT | BEDINGTE FORMATIERUNG** auf. Wählen Sie im ersten DropDown-Feld den Eintrag **FORMEL IST** und geben Sie die Formel mit Bezug auf die erste Zelle des markierten Bereichs ein. Wählen Sie anschließend ein Format und bestätigen Sie Ihre Eingaben mit **OK**. Wollen Sie alle *nicht gesperrten* Zellen markieren, lautet die Formel:

```
=IsLocked(A10)=FALSCH
```

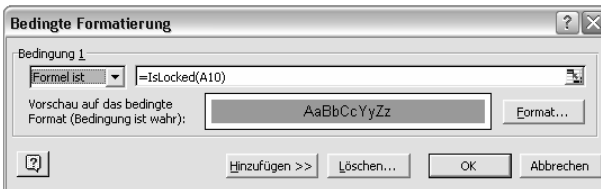


Abbildung 33: Gesperrte Zellen farblich markieren

Im Tabellenblatt `ISLOCKED` der Beispieldatei haben wir zu der bedingten Formatierung eine Tabelle erstellt.

Neben der bedingten Formatierung ist es mit einer VBA-Prozedur ebenfalls möglich, alle gesperrten Zellen farblich hervorzuheben. Um den nachfolgenden Code zu testen, markieren Sie im Tabellenblatt `ShowLockedCells` der Beispieldatei einen Bereich, bevor Sie das Makro `ShowLockedCells` über die Schaltfläche auf dem Tabellenblatt starten.

In der Prozedur wird über eine `For Each...Next`-Anweisung jede Zelle eines markierten Bereichs angesprochen. In einer `If...Then...Else`-Anweisung wird der Sperrstatus der jeweiligen Zelle über die `Locked`-Eigenschaft abgefragt. Ist die Zelle gesperrt, beträgt der Wert `True` und die Zelle erhält die im Code angegebene Hintergrundfarbe 45. Die `Else`-Anweisung wird nur durchlaufen, wenn die Zelle nicht gesperrt ist, also den Wert `False` zurückgibt. In dem Fall wird der Zelle die Hintergrundfarbe 0 zugewiesen. Eine eventuell vorhandene Farbe wird dadurch entfernt.

```
'=====
' Auf CD      Buchdaten\Beispiele\02_Allgemein\
' Dateiname   02_02_CellProtection.xls
' Tabelle     Show Locked Cells
' Modul       mdl_02_ShowLockedCells
'=====
```

```
Sub ShowLockedCells()  
    Dim rngCell As Range  
  
    For Each rngCell In Selection  
        If rngCell.Locked Then  
            rngCell.Interior.ColorIndex = 45  
        Else  
            rngCell.Interior.ColorIndex = 0  
        End If  
    Next rngCell  
  
End Sub
```

Da bekanntlich viele Wege nach Rom führen, kann man obigen Code auch anders aufbauen. Anstelle der If...Then...Else-Anweisung verwenden wir im nachfolgenden Code die IIf-Funktion für die Farbzuzuweisung der Zelle. Beide Prozeduren führen zu dem gleichen Ergebnis. Hier kann jeder Programmierer seinen persönlichen Stil einbringen.

```
'=====
```

```
' Auf CD      Buchdaten\Beispiele\02_Allgemein\  
' Dateiname  02_02_ CellProtection.xls  
' Tabelle    Show Locked Cells  
' Modul      mdl_02_ShowLockedCells
```

```
'=====
```

```
Sub ShowLockedCells2()  
    Dim rngCell As Range  
  
    For Each rngCell In Selection  
        rngCell.Interior.ColorIndex = IIf(rngCell.Locked, 45, 0)  
    Next rngCell  
End Sub
```

Um die Hintergrundfarben aller Zellen zu löschen, markieren Sie durch Drücken der Tastenkombination **[Strg]+[A]** das gesamte Tabellenblatt und klicken in der Symbolleiste **FORMAT** auf den Pfeil des **FÜLLEIMER**-Symbols und wählen die Option **KEINE FÜLLFARBE**.

Im Tabellenblatt **ShowLockedCells** befindet sich eine zweite Schaltfläche, um die Hintergrundfarben der Tabelle zu entfernen. Dieser haben wir das Makro **ClearColors** zugewiesen. Es ist nicht notwendig, das gesamte Tabellenblatt vor dem Aufruf der Prozedur zu markieren.

```
'=====
```

```
' Auf CD      Buchdaten\Beispiele\02_Allgemein\  
' Dateiname  02_02_ CellProtection.xls  
' Tabelle    Show Locked Cells  
' Modul      mdl_02_ShowLockedCells
```

```
'=====
```

```
Sub ClearColors()  
    ActiveSheet.Cells.Interior.ColorIndex = 0  
End Sub
```

Grundlagen

Allgemein

Datum
Zeit

tungen

Steuerelemente

Befehle
leisten

Objekt

Diagramm

Ereignisse

User-Form

Web/
Mail

Extern

Gemischtes

Spezial

Zellenformate können auch über den Menüpunkt BEARBEITEN | LÖSCHEN | FORMATE gelöscht werden. Der entsprechende VBA-Befehl lautet:

```
Selection.ClearFormats
```

Dabei werden *alle* Formate der markierten Zellen auf die Standard-Einstellungen zurückgesetzt. Ebenfalls zu den Standard-Einstellungen gehört, dass Zellen als *gesperrt* formatiert sind. Bedenken Sie diese Auswirkung, bevor Sie alle Formate einer Zelle löschen.

44 Formelzellen schützen

Standardmäßig sind alle Zellen eines neuen Tabellenblattes gesperrt und die Option AUSGEBLENDET ist deaktiviert. Diese Einstellungen wirken sich erst bei eingeschaltetem Blattschutz aus. Oftmals sollen in einem Tabellenblatt lediglich die Zellen geschützt werden, welche Formeln enthalten. Dazu müssen Sie in allen Zellen des Tabellenblattes die Sperroption der Zellen ohne Formeln aufheben, um diese zur Bearbeitung freizugeben. Auf dem Tabellenblatt sehen Sie normalerweise nur Zahlen oder Text in den einzelnen Zellen. Ob diese Werte über eine Formel berechnet wurden, erkennen Sie an dem Inhalt der Bearbeitungszeile. Zur Überprüfung müsste jede Zelle einzeln markiert werden.

In der folgenden Prozedur `LockedFormulas1` wird zunächst der Blattschutz aufgehoben. Anschließend werden alle Zellen im benutzten Bereich des Tabellenblattes entsperrt und die Formel in der Bearbeitungszeile eingeblendet. Über das `Type`-Argument `xlCellTypeFormulas` der `SpecialCells`-Methode werden dann alle Formelzellen des benutzten Bereichs gesperrt und die Formeln in der Bearbeitungszeile ausgeblendet.

In dem Tabellenblatt wurden Formeln der Funktion `ISLOCKED()` aus dem Rezept 43 hinterlegt. Diese Formeln werden erst durch eine Neuberechnung des Tabellenblattes aktualisiert. Über die `Calculate`-Methode wird nur der in der `With`-Anweisung angesprochene Formel-Bereich `.UsedRange.SpecialCells(xlCellTypeFormulas)` neu berechnet. Damit die gemachten Schutz-Einstellungen wirksam werden, wird das Tabellenblatt über die `Protect`-Methode geschützt. Hier vergeben wir das Passwort `Codebook`.

```
'=====
' Auf CD      Buchdaten\Beispiele\02_Allgemein\
' Dateiname  02_02_ CellProtection.xls
' Tabelle     Locked Formula Cells
' Modul      mdl_03_LockedFormulas
'=====
```

```
Sub LockedFormulas1()
```

```
    With ActiveSheet
```

```
        .Unprotect "Codebook"
```

```
        ' Alle Zellen entsperren und Formeln einblenden
```

```
        With .UsedRange.Cells
```

```
            .Locked = False
```

```
            .FormulaHidden = False
```

```
        End With
```

```
        ' Alle Formelzellen sperren und Formeln ausblenden
```

```
With .UsedRange.SpecialCells(xlCellTypeFormulas)
    .Locked = True
    .FormulaHidden = True
    ' Formelbereich neu berechnen
    .Calculate
End With
.Protect "Codebook"
End With
End Sub
```

In der vorherigen Prozedur `LockedFormulas1` werden zunächst alle Zellen entsperrt und anschließend nur die Formelzellen in einem Durchgang gesperrt.

Es ist ebenso möglich, über die `HasFormula`-Eigenschaft der Zellen die Schutzeinstellungen vorzunehmen. Der `HasFormula`-Rückgabewert `True` oder `False` wird auf die `Locked`-Eigenschaft einer Zelle übertragen. Diese Methode empfiehlt sich nur bei kleinen Tabellenbereichen, da in einer `For Each...Next`-Anweisung jede Zelle einzeln abgearbeitet werden muss.

```
'=====
' Auf CD      Buchdaten\Beispiele\02_Allgemein\
' Dateiname  02_02_ CellProtection.xls
' Tabelle    Locked Formula Cells
' Modul      mdl_03_LockedFormulas
'=====

Sub LockedFormulas2()
    Dim rngCell As Range

    With ActiveSheet
        .Unprotect "Codebook"
        For Each rngCell In .UsedRange
            With rngCell
                .Locked = .HasFormula
                .FormulaHidden = .HasFormula
            End With
        Next
        .Calculate
        .Protect "Codebook"
    End With
End Sub
```

45 Autofilter trotz Blattschutz

Ab Excel 2002 wird die Nutzung des Autofilters über die `AllowFiltering`-Eigenschaft des `Protection`-Objektes aktiviert (siehe Rezepte 48 und 49).

Achtung

Die benutzerdefinierten Schutz-Einstellungen, die Sie in Excel 2002 vorgenommen haben, werden in der Arbeitsmappe nicht unterstützt, wenn Sie diese in älteren Excel-Versionen öffnen.

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

Bei eingestelltem Blattschutz ist es bis einschließlich Excel 2000 standardmäßig nicht möglich, den Autofilter anzuwenden. Wenn der AUTOFILTER im Tabellenblatt eingestellt ist, können Sie die AutoFilter-Pfeile mit der EnableAutoFilter-Eigenschaft aktivieren. Setzen Sie dazu deren Eigenschaft auf True. Anschließend schützen Sie das Tabellenblatt und schalten zusätzlich den Schutz *Nur-Benutzerschnittstelle* über die UserInterfaceOnly-Eigenschaft ein. Dadurch wird nur die Benutzeroberfläche, jedoch keine Makros geschützt

Diese Eigenschaft muss für jedes Arbeitsblatt eingestellt werden und wird nicht mit dem Arbeitsblatt oder der Sitzung gespeichert. Im folgenden Beispiel werden die AUTOFILTER-Pfeile im geschützten Arbeitsblatt aktiviert.

```
' =====
' Auf CD      Buchdaten\Beispiele\02_Allgemein\
' Dateiname   02_03_WorksheetProtectTools.xls
' Tabelle     AutoFilter
' Modul       mdl_01_AutoFilter
' =====

Sub AutoFilter()
    With ActiveSheet
        .EnableAutoFilter = True
        .Protect UserInterfaceOnly:=True
    End With
End Sub
```

46 Gliederung trotz Blattschutz

Für die Gliederungs-Funktion gelten die gleichen Voraussetzungen wie für die Autofilter-Funktion. Bei eingeschaltetem Blattschutz können die Gliederungssymbole bis zur Excel-Version 2000 nur aktiviert werden, wenn die Outlining-Eigenschaft auf True gesetzt wird.

```
' =====
' Auf CD      Buchdaten\Beispiele\02_Allgemein\
' Dateiname   02_03_WorksheetProtectTools.xls
' Tabelle     Gliederung_Outlining
' Modul       mdl_02_Outlining
' =====

Sub Outlining()
    With ActiveSheet
        .EnableOutlining = True
        .Protect UserInterfaceOnly:=True
    End With
End Sub
```

47 Sortierung trotz Blattschutz

Im Gegensatz zu den vorherigen Funktionen AUTOFILTER und GLIEDERUNG gibt es bis Excel 2000 für die Sortierung von Tabellen keine Eigenschaft, diese bei eingeschaltetem Blattschutz zuzulassen.

Ab Excel 2002 steht zur Sortierung von Daten in einem Tabellenblatt die AllowSorting-Eigenschaft des Protection-Objektes zur Verfügung. Bei der AllowSorting-Eigenschaft ergibt sich allerdings ein großer Nachteil: In dem zu sortierenden Bereich eines geschützten Tabellenblattes (incl. Überschrift) dürfen keine Zellen gesperrt sein. Sobald auch nur eine Zelle im zu sortierenden Bereich gesperrt ist, kann keine Sortierung von Daten vorgenommen werden. Excel gibt die Fehlermeldung aus, dass vor der Sortierung der Blattschutz aufgehoben werden muss.

Um eine individuelle Sortierung in allen Excel-Versionen durch den Anwender zuzulassen, bedienen wir uns eines kleinen Makros. Nachdem der Blattschutz aufgehoben wurde, wird der SORTIEREN-Dialog angezeigt. Der Anwender kann nun nach Belieben sortieren, solange das Dialogfenster angezeigt wird. Es ist aber weiterhin nicht möglich, Änderungen im Tabellenblatt vornehmen.

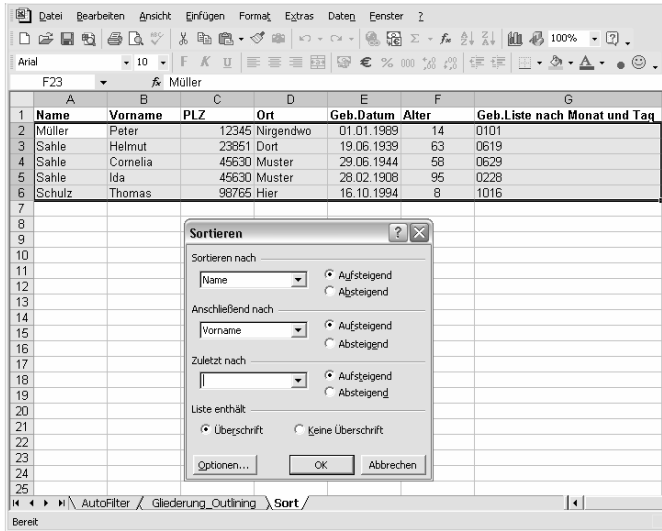


Abbildung 34: Sortieren bei eingeschaltetem Blattschutz

Hinweis

Eine Übersicht der geläufigsten in Excel integrierter Dialogfelder finden Sie im Anhang.

Um eine Tabelle zu sortieren, können Sie den Bereich Ihrer Tabelle markieren, welcher sortiert werden soll. Markieren Sie eine einzelne Zelle innerhalb eines zusammenhängenden Bereichs, wird dieser zusammengehörende Bereich ohne ausdrückliche Markierung sortiert. Damit die Prozedur SortBySheetProtection den Sortierbereich erkennt, müssen Sie entweder eine Zelle innerhalb des Datenbereichs oder den gesamten Sortierbereich markieren. Wir markieren den benutzten Bereich UsedRange des Tabellenblattes. Die vor Makroaufruf aktive Zelle wird in der Variablen rngCell gespeichert, damit diese Zelle nach der Sortierung wieder markiert werden kann.

```
' =====
' Auf CD      Buchdaten\Beispiele\02_Allgemein\
' Dateiname   02_03_WorksheetProtectTools.xls
' Tabelle     Sort
```

Grundlagen

Allgemein

Datum
Zeit

Formeln

Steuerelemente

Befehle
leisten

Objekte

Diagramme

Ereignisse

User-Formen

Web/
Mail

Extern

Gemischtes

Spezial

```

' Modul      mdl_03_Sort
' =====

Sub SortBySheetProtection()
    Dim rngCell As Range

    ' aktuelle Zellmarkierung merken
    Set rngCell = ActiveCell

    With ActiveSheet
        ' Blattschutz aufheben
        .Unprotect "Codebook"
        ' verwendeten Bereich markieren
        .UsedRange.Activate
        ' Dialog >>Sortieren<< anzeigen
        Application.Dialogs(xlDialogSort).Show
        ' Blattschutz setzen
        .Protect "Codebook"
    End With

    ' Startzelle markieren
    rngCell.Select
End Sub

```

48 Schutzoptionen auslesen

Neu in Excel 2002 ist die individuelle Einstellung des Blattschutzes. Es ist endlich möglich, gezielte Angaben zu machen, welche Aktionen bei aktiviertem Blattschutz durch den Anwender möglich sind.

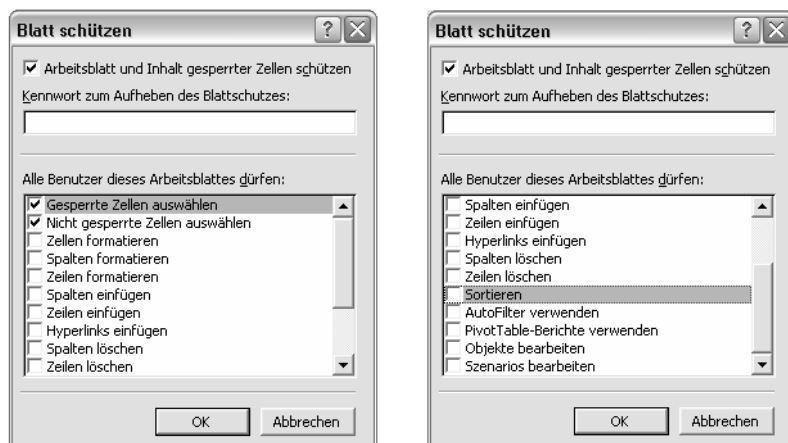


Abbildung 35: Standardeinstellungen Blattschutz

Die ersten zwei Optionen GESPERRTE ZELLEN AUSWÄHLEN und NICHT GESPERRTE ZELLEN AUSWÄHLEN (siehe Abbildung 35) geben an, welche Elemente auf dem Blatt markiert werden können. Es ist nicht möglich, diese beiden Schutzoptionen einzeln auszulesen. Verwenden Sie die EnableSelection-Eigenschaft, um die Kombination dieser beiden Optionen zu ermitteln.

Schutzeinstellungen	Bedeutung	Konstante	Wert
[x] GESPERRTE ZELLEN AUSWÄHLEN [x] NICHT GESPERRTE ZELLEN AUSWÄHLEN	keine Einschränkung der Markierung	xlNoRestrictions	0
[] GESPERRTE ZELLEN AUSWÄHLEN [x] NICHT GESPERRTE ZELLEN AUSWÄHLEN	Markierung nicht gesperrter Zellen	xlUnlockedCells	1
[] GESPERRTE ZELLEN AUSWÄHLEN [] NICHT GESPERRTE ZELLEN AUSWÄHLEN	keine Markierung möglich	xlNoSelection	4142

Tabelle 8: Konstanten der EnableSelection-Eigenschaft

Bis zur Excel-Version 2000 ist es über den Menüpunkt EXTRAS | SCHUTZ | BLATT SCHÜTZEN | INHALTE nur möglich, Änderungen gesperrter Zellen zu unterbinden. Den Sperrstatus einer Zelle geben Sie über das Zellenformat unter dem Menüpunkt FORMAT | ZELLE Registerkarte SCHUTZ Option GESPERRT an. Einstellungen, welche die Markierung von Zellen betreffen, müssen über VBA vorgenommen werden. Bis Excel 2000 muss bei jedem Öffnen einer Arbeitsmappe diese Einstellung über die EnableSelection-Eigenschaften neu gesetzt werden.

Die beiden letzten Optionen, OBJEKTE BEARBEITEN und SZENARIOS BEARBEITEN, welche Sie dem Dialogfenster entnehmen können, wurden aus den vorherigen Excel-Versionen übernommen.

In Excel 2002 ist das Protection-Objekt mit zwölf Elementen neu hinzugekommen. Es stellt die verschiedenen Arten von Schutzoptionen dar, die für ein Arbeitsblatt zur Verfügung stehen. Elf dieser Elemente sind im Dialog BLATT SCHÜTZEN (Abbildung 35) aufgeführt. Das zwölfte Element AllowEditRange gibt eine Auflistung der benutzerberechtigten Bereiche zurück.

Schutzoption	Protection-Eigenschaft
Zellen formatieren	AllowFormattingCells
Spalten formatieren	AllowFormattingColumns
Zeilen formatieren	AllowFormattingRows
Spalten einfügen	AllowInsertingColumns
Zeilen einfügen	AllowInsertingRows
Hyperlinks einfügen	AllowInsertingHyperlinks
Spalten löschen	AllowDeletingColumns
Zeilen löschen	AllowDeletingRows
Sortieren	AllowSorting
Autofilter	AllowFiltering
Pivot-Table-Berichte	AllowUsingPivotTables
Benutzerberechtigte Bereiche	AllowEditRanges

Tabelle 9: Übersicht der Protection-Elemente

Grundlagen

Allgemein

Datum
Zeit

Formeln

Steuerelemente

Befehle

Objekt

Diagramm

Ereignisse

User-Form

Web/Mail

Extern

Gemischtes

Spezial

Achtung Mit VBA können Sie die gegenwärtigen Schutzoptionen für die Anwendung auslesen und einstellen. Es ist nicht möglich, die Werte aller Optionen in einer Schleife abzufragen. Diese müssen einzeln aufgeführt und ausgelesen werden.

Der Objektvariablen `wks` wird das aktive Tabellenblatt zugewiesen. Das hat den Vorteil, dass die IntelliSense bei der Auswahl der `Protection`-Elemente angesprochen wird. Des Weiteren kann in einer `With`-Anweisung eine Reihe von Anweisungen für ein einzelnes Objekt zusammengefasst werden.

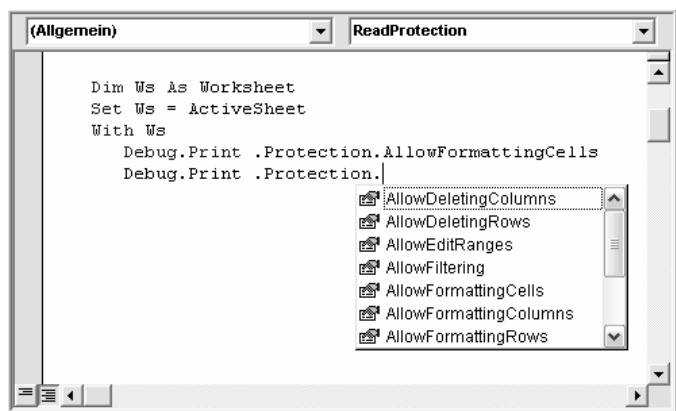


Abbildung 36: IntelliSense-Auflistung des `Protection`-Objektes

Tip

Nähere Informationen zur IntelliSense finden Sie in der Kategorie »Einführung«.

In nachfolgender Prozedur werden alle Schutzoptionen mit ihrem Wert im Direktfenster ausgegeben. In einer `For...Next`-Anweisung wird über die `Count`-Eigenschaft des `AllowEditRanges`-Objektes die Anzahl an Benutzerberechtigungen zurückgegeben. Sind im aktiven Blatt Benutzerberechtigungen zum Bearbeiten von Bereichen vorhanden, dann werden zu jedem Bereich die Bereichs-Nummer, der -Name sowie -Bezug ausgegeben. Wenn das Direktfenster im VBA-Editor nicht sichtbar ist, zeigen Sie es über den Menüpunkt `ANSICHT | DIREKTFENSTER` oder der Tastenkombination `[Strg]+[G]` an.

Achtung

Das `Protection`-Objekt wird in früheren Excel-Versionen nicht unterstützt. Verwenden Sie Makros mit diesem Objekt, kommt es in den Versionen bis Excel 2000 zu Fehlermeldungen.

```
' =====
' Auf CD    Buchdaten\Beispiele\02_Allgemein\
' Dateiname 02_04_Protection.xls
' Tabelle   ReadProtection
' Modul     mdl_01_ReadProtection
' =====
```

```
Sub ReadProtection()  
    Dim wks As Worksheet  
    Dim rngArea As Range  
    Dim int1 As Integer  
  
    Set wks = ActiveSheet  
    With wks  
        Debug.Print "Blattschutz          --> "; _  
            .ProtectContents  
  
        Debug.Print "Zellmarkierung        --> "; _  
            .EnableSelection  
  
        Debug.Print "Zellen formatieren      --> "; _  
            .Protection.AllowFormattingCells  
  
        Debug.Print "Spalten formatieren    --> "; _  
            .Protection.AllowFormattingColumns  
  
        Debug.Print "Zeilen formatieren     --> "; _  
            .Protection.AllowFormattingRows  
  
        Debug.Print "Spalten einfügen       --> "; _  
            .Protection.AllowInsertingColumns  
  
        Debug.Print "Zeilen einfügen        --> "; _  
            .Protection.AllowInsertingRows  
  
        Debug.Print "Hyperlinks einfügen    --> "; _  
            .Protection.AllowInsertingHyperlinks  
  
        Debug.Print "Spalten löschen       --> "; _  
            .Protection.AllowDeletingColumns  
  
        Debug.Print "Zeilen löschen        --> "; _  
            .Protection.AllowDeletingRows  
  
        Debug.Print "Sortieren             --> "; _  
            .Protection.AllowSorting  
  
        Debug.Print "Autofilter            --> "; _  
            .Protection.AllowFiltering  
  
        Debug.Print "Pivot Table-Berichte  --> "; _  
            .Protection.AllowUsingPivotTables  
  
        Debug.Print "Objekte               --> "; _  
            .ProtectDrawingObjects  
  
        Debug.Print "Szenarios bearbeiten  --> "; _  
            .ProtectScenarios  
  
        Debug.Print Chr(10); "Anzahl Benutzerberechtigte"; Chr(10); _  
            "Bereiche          --> "; _
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

```

        .Protection.AllowEditRanges.Count

' Auflistung Benutzerberechtigte Bereiche
For intI = 1 To .Protection.AllowEditRanges.Count
    With .Protection.AllowEditRanges.Item(intI)
        Debug.Print " "
        Debug.Print "Bereichs Nummer      --> "; _
            intI
        Debug.Print "Bereichs Titel      --> "; _
            .Title
        Debug.Print "Bereichs Adresse    --> "; _
            .Range.Address
    End With
Next
End With
End Sub

```

49 Schutzoptionen einstellen

Über das `Protection`-Objekt können Sie die Schutzeinstellungen eines Tabellenblattes per VBA vorgeben. Am Beispiel der Schutzoption `ZELLEN FORMATIEREN` zeigen wir Ihnen die Befehle zur Aktivierung bzw. Deaktivierung der Optionen. Bis auf die `AllowEditRanges`-Eigenschaft können Sie alle anderen Eigenschaften des `Protection`-Objektes nach dem gleichen Muster einstellen. Eine Übersicht aller Eigenschaften finden Sie im Rezept 48.

Das Tabellenblatt `SETPROTECTION` der Beispieldatei ist geschützt. Änderungen an den Zelleninhalten sind nur in den nicht gesperrten Zellen zulässig. Die Makros `ProtectTrue` und `ProtectFalse` haben wir jeweils einer Schaltfläche zugewiesen. Über diese Schaltflächen wird die Schutzoption `ZELLEN FORMATIEREN` aktiviert bzw. deaktiviert. Nur wenn die Schutzoption `ZELLEN FORMATIEREN` aktiviert ist, können sowohl die gesperrten als auch die nicht gesperrten Zellen formatiert werden. Bei einer Deaktivierung der Option sind keine Zellen des Tabellenblattes formatierbar. Die Deaktivierung erkennen Sie am leichtesten an den Symbolleisten `STANDARD` und `FORMAT`. Alle Formatierungs-Symbole und die Farbpaletten der `FÜLLFARBE` und `SCHRIFTFARBE` sind ausgegraut, also nicht anwählbar.

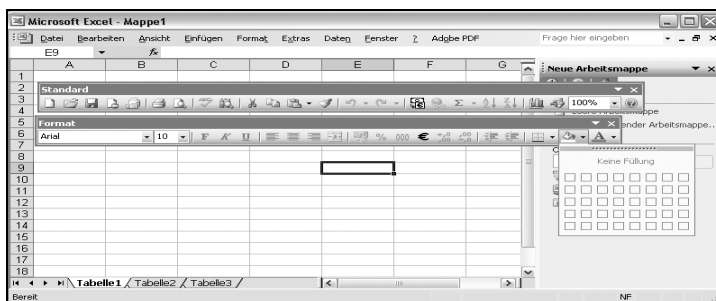


Abbildung 37: Ausgegraute Formatierungs-Schaltflächen

```
' =====  
' Auf CD      Buchdaten\Beispiele\02_Allgemein\  
' Dateiname  02_04_Protection.xls  
' Tabelle    SetProtection  
' Modul      mdl_02_SetProtection  
' =====  
  
Sub ProtectTrue()  
    ActiveSheet.Protect AllowFormattingCells:=True  
End Sub  
  
Sub ProtectFalse()  
    ActiveSheet.Protect AllowFormattingCells:=False  
End Sub
```

50 Nicht gesperrte Zellen markieren und formatieren

In unserem vorherigen Beispiel ist es bei eingestelltem Blattschutz und Aktivierung der Option ZELLEN FORMATIEREN möglich, auch die gesperrten Zellen zu formatieren. Damit nur die *nicht gesperrten* Zellen formatiert werden können, untersagen Sie einfach die Markierung der gesperrten Zellen. Aktivieren Sie dazu unbedingt die Option NICHT GESPERRTE ZELLEN AUSWÄHLEN, da sonst keine Zellen des Tabellenblattes markiert werden können. In Abbildung 38 sehen Sie die entsprechenden Einstellungen.

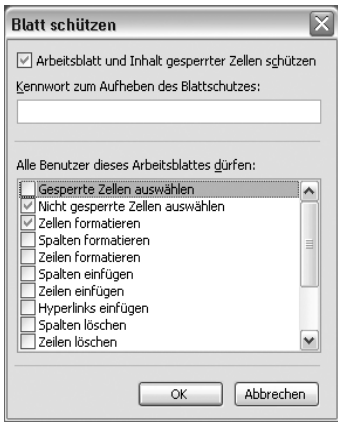


Abbildung 38: Alle nicht gesperrten Zellen sind formatierbar

In VBA nehmen Sie die Einstellungen der Zellenmarkierung über die `SelectionEnable`-Eigenschaft vor. Um nur die Markierung nicht gesperrter Zellen zu erlauben, wird die `EnableSelection`-Konstante `xlUnlockedCells` verwendet.

Tipp

Ersetzen Sie die `AllowFormattingCell`-Eigenschaft durch eine andere `Protection`-Eigenschaft aus der Tabelle 9. Bis auf die `AllowEditRanges`-Eigenschaft können Sie alle anderen Elemente des `Protection`-Objektes nach dem gleichen Muster wie in den folgenden Prozeduren einstellen. Zum Thema »Benutzerberechtigte Bereiche einstellen« zeigen wir Ihnen im Rezept 54 ein Beispiel.

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

```
'=====
' Auf CD      Buchdaten\Beispiele\02_Allgemein\
' Dateiname  02_04_Protection.xls
' Tabelle     FormatUnlockedCells
' Modul       mdl_03_ProtectLockedCells
'=====

Sub ProtectLockedCellsTrue()
    ActiveSheet.EnableSelection = xlUnlockedCells
    ActiveSheet.Protect AllowFormattingCells:=True
End Sub

Sub ProtectLockedCellsFalse()
    ActiveSheet.EnableSelection = xlNoRestrictions
    ActiveSheet.Protect AllowFormattingCells:=True
End Sub
```

51 Nicht gesperrte Zellen formatieren (ab Excel 2002)

Angenommen, Sie wollen in einem geschützten Tabellenblatt lediglich die Formatierung von gesperrten Zellen verhindern, dabei soll aber eine Markierung der nicht gesperrten Zellen nach wie vor möglich sein.

Diese Kombination lässt sich nicht alleine über den Dialog BLATT SCHÜTZEN einstellen. Nehmen Sie dazu das Worksheet_SelectionChange-Ereignis des Tabellenblattes zur Hilfe. In der Ereignis-Prozedur verwenden Sie eine For Each...Next-Anweisung. Diese ermöglicht es, sowohl eine einzelne markierte Zelle als auch einen Bereich von mehreren markierten Zellen auf ihre jeweilige Locked-Eigenschaften zu überprüfen.

Sobald auch nur eine Zelle dieses markierten Bereichs gesperrt ist, wird die Schutzoption ZELLEN FORMATIEREN deaktiviert und die Prozedur verlassen. Mit dem Schlüsselwort Me wird das Tabellenblatt angesprochen, in welchem die Ereignisprozedur hinterlegt ist. Nur wenn keine Zelle des markierten Bereichs Target gesperrt ist, wird eine Formatierung der Zellen ermöglicht.

Achtung

Das Protection-Objekt wird in früheren Excel-Versionen nicht unterstützt. Verwenden Sie Makros mit diesem Objekt, kommt es in den Versionen bis Excel 2000 zu Fehlermeldungen.

```
'=====
' Auf CD      Buchdaten\Beispiele\02_Allgemein\
' Dateiname  02_04_Protection.xls
' Tabelle     FormatUnlockedCells_Change
' Ereignis    Tabelle4 (FormatUnlockedCells_Change)
'=====

Private Sub Worksheet_SelectionChange(ByVal Target As Range)
    Dim rngCell As Range

    For Each rngCell In Target
        If rngCell.Locked = True Then
            Me.Protect AllowFormattingCells:=False
        End If
    Next rngCell
End Sub
```

```
Exit Sub
End If
Next rngCell

Me.Protect AllowFormattingCells:=True
End Sub
```

Grundlagen

Allgemein

Datum und Zeit

Formeln

Steuerelemente

Befehle

Objekte

Diagramme

Ereignisse

User-Formen

Web/Mail

Extern

Gemischtes

Spezial

52 Nicht gesperrte Zellen formatieren (alle Excel-Versionen)

In Excel 2000 ist es ebenfalls mit Hilfe des Worksheet_SelectionChange-Ereignisses möglich, Formatierungen in nicht gesperrten Zellen zu erlauben. Das Tabellenblatt der Beispieldatei ist dabei mit dem Passwort »Codebook« geschützt. Wird eine nicht gesperrte Zelle markiert, wird der Blattschutz aufgehoben, bei gesperrten Zellen wird er wieder gesetzt.

Wird ein Bereich aus mehreren Zellen markiert, welche einen unterschiedlichen GESPERRT-Status haben, kann Excel sich nicht entscheiden, ob die Syntax Target.Locked den Wert True oder False hat, und gibt den Wert NULL zurück. Dementsprechend wurde die Prozedur so programmiert, dass der Blattschutz nur dann aufgehoben wird, wenn Target.Locked den Wert False zurückgibt. Dies ist der Fall, wenn alle Zellen im markierten Bereich nicht gesperrt sind.

```
'=====
' Auf CD      Buchdaten\Beispiele\02_Allgemein\
' Dateiname  02_04a_FormatUnLockedCells_Excel2000.xls
' Ereignis   Tabelle1 (FormatCells)
'=====

Private Sub Worksheet_SelectionChange(ByVal Target As Range)
    If Target.Locked = False Then
        Me.Unprotect "Codebook"
    Else
        Me.Protect "Codebook"
    End If
End Sub
```

53 Schutzoptionen ändern

In unserem nächsten Beispiel verwenden wir die Schutzoption SPALTEN EINFÜGEN. Anstelle der entsprechenden Protection-Eigenschaft AllowInsertingColumns können Sie auch ein anderes Element aus der Tabelle 9 einsetzen. Lediglich die AllowEditRanges-Eigenschaft benötigt zusätzliche Argumente. Dazu zeigen wir in Ihnen im Rezept 54 ein Beispiel.

Hinweis

Eine eingefügte Spalte übernimmt standardmäßig die Formatierung der Spalte, die links von ihr steht. Das bedeutet, dass diese möglicherweise über gesperrte Zellen verfügt und die Benutzer vielleicht nicht in der Lage sind, die von ihnen eingefügten Spalten wieder zu löschen.

In der Prozedur ChangeProtection wird zunächst die Einstellung der Schutzoption SPALTEN EINFÜGEN ausgelesen und an die Variable blnOption übergeben. Im weiteren Verlauf der Prozedur

kann über diese Variable jederzeit auf die anfängliche Schutz-Einstellung zurückgegriffen werden. Die Variable `blnOption` hat den Wert `True`, wenn die Schutzoption aktiviert ist, `False` wenn deaktiviert. Diese Information wird in einer `If...Then...Else`-Anweisung in einen sprechenden Text umgewandelt. Die Variable `strResult` erhält für den Wahr-Wert den Text »zugelassen«, für den Falsch-Wert »untersagt«. Anschließend wird in einem Meldungsfenster die anfänglich ermittelte Einstellung bekannt gegeben. Das Meldungsfenster weist mittels der Konstante `vbYesNo` eine JA- und eine NEIN-Schaltfläche auf. Klicken Sie auf JA, um die aktuelle Einstellung der Schutzoption zu ändern. Bei NEIN werden Sie in einem weiteren Meldungsfenster darüber informiert, dass die Einstellung nicht geändert wurde.

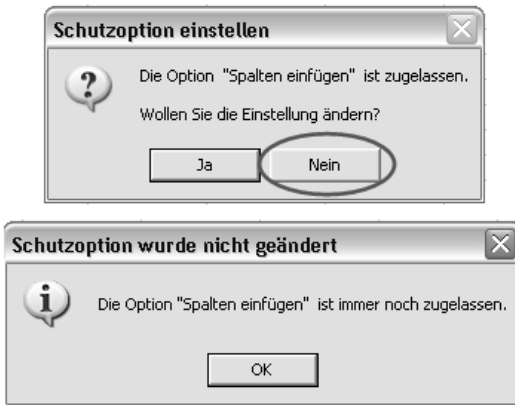


Abbildung 39: Es wurden keine Änderungen vorgenommen

Wollen Sie die Einstellung der Schutzoption SPALTEN EINFÜGEN ändern, klicken Sie im Meldungsfenster SCHUTZOPTION EINSTELLEN auf JA. In einer weiteren `If...Then...Else`-Anweisung wird anhand des Wertes der Variablen `blnOption` die Schutzeinstellung umgedreht. Aus `False` wird `True`, aus `True`, `False`. Der Blattschutz wurde mit dem Passwort `Codebook` versehen. Bei Änderung der Schutz-Einstellungen muss dieses im Code mit angegeben werden. Geben Sie das Passwort nicht in der Prozedur an, wird das vergebene Passwort zurückgesetzt. Der Dialog BLATT SCHÜTZEN ist damit für den Anwender ohne Eingabe eines Passwortes zugänglich. Für die Schlussmeldung erhält die Variable `strResult` den umgekehrten Wert.

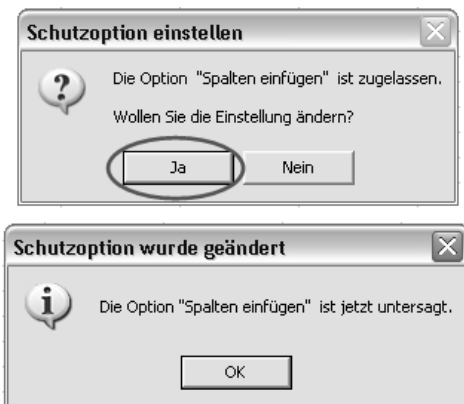


Abbildung 40: Änderungen wurden vorgenommen

```

'=====
' Auf CD      Buchdaten\Beispiele\02_Allgemein\
' Dateiname  02_04_Protection.xls
' Tabelle    ChangeProtection
' Modul      mdl_04_ChangeProtection
'=====

Sub ChangeProtection()
    Dim blnOption As Boolean
    Dim strResult As String
    Dim bytQuestion As Byte

    ' Schutzzustand auslesen
    blnOption = ActiveSheet.Protection.AllowInsertingColumns

    If blnOption = True Then
        strResult = "zugelassen"
    Else
        strResult = "untersagt"
    End If

    bytQuestion = _
        MsgBox("Die Option ""Spalten einfügen"" ist " & _
            strResult & "." & _
            vbNewLine & vbNewLine & _
            "Wollen Sie die Einstellung ändern?" _
            , vbYesNo + vbQuestion, _
            "Schutzoption einstellen")

    If bytQuestion = vbYes Then ' Einstellung soll geändert werden
        If blnOption = True Then ' Schutz ist WAHR gewesen
            ActiveSheet.Protect _
                Password:="Codebook", _
                AllowInsertingColumns:=False
            strResult = "untersagt"
        Else ' Schutz ist FALSCH gewesen
            ActiveSheet.Protect _
                Password:="Codebook", _
                AllowInsertingColumns:=True
            strResult = "zugelassen"
        End If

        MsgBox "Die Option ""Spalten einfügen"" ist jetzt " & _
            strResult & ".", _
            vbOKOnly + vbInformation, _
            "Schutzoption wurde geändert"
    Else ' Einstellung soll nicht
        ' geändert werden
        MsgBox "Die Option ""Spalten einfügen"" ist immer noch " & _
            strResult & ".", _
            vbOKOnly + vbInformation, _
            "Schutzoption wurde nicht geändert"
    End If
End Sub

```

Grund-
lagenAllge-
meinDatum
Zeit

tungen

Steuer
elementeBefehl
leisten

Objekt

Dia-
grammEreig-
nisseUser-
FormWeb/
Mail

Extern

Ge-
misch-
tes

Specia

54 Benutzerberechtigte Bereiche hinzufügen

Ab der Version Excel 2002 ist es möglich, Benutzerberechtigungen für ausgewählte Bereiche zu erteilen. Diese Einstellung können Sie unter EXTRAS | SCHUTZ | BENUTZER DÜRFEN BEREICHE BEARBEITEN vornehmen. Diese Option ermöglicht es, definierte Zellen oder Bereiche in einem geschützten Arbeitsblatt zur Bearbeitung freizugeben. Benutzerberechtigte Bereiche gelten je Tabellenblatt. Der gleiche Bereichsname kann mit den gleichen oder verschiedenen Bezugsangaben in anderen Tabellenblättern erstellt werden. Zur Identifizierung muss in VBA jeweils das Tabellenblatt mit angegeben werden.

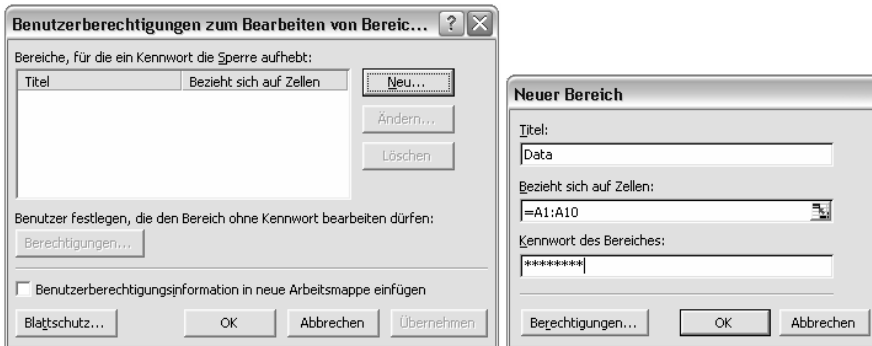


Abbildung 41: Benutzerberechtigungen in Excel 2002 definieren

Die Einstellungen der Bereiche werden in VBA über das `AllowEditRange`-Objekt vorgenommen. Dieses Objekt ist eine der Eigenschaften des `Protection`-Objektes (siehe Tabelle 9).

In der Prozedur `AddAllowEditRange` soll im aktiven Tabellenblatt der benutzerberechtigte Bereich `Data` neu erstellt werden. Existiert der Titel des neu einzufügenden Bereichs bereits, erhalten Sie eine Fehlermeldung. Dieser Fehler würde durch die vorangestellte `On Error GoTo`-Anweisung zur Sprungmarke `Errorhandler` verzweigen. Hier wird ein Meldungsfenster mit dem Fehlerhinweis ausgegeben. Anstelle der `MsgBox` können auch andere oder gar keine Befehle angegeben werden.

Die Anweisung `With ActiveSheet` ermöglicht es, eine Reihe von Anweisungen für das aktive Tabellenblatt zusammenzufassen. Dadurch muss der Name `ActiveSheet` nicht vor jeder Anweisung angegeben werden. Vor einem Befehl für `ActiveSheet` muss lediglich ein Punkt geschrieben werden. Dieser ersetzt den Namen des Objektes `ActiveSheet`.

In unserem Beispiel wurde das Tabellenblatt `ADD_DELETE_AER` mit dem Passwort `VBA` geschützt. Um einen neuen benutzerberechtigten Bereich zu erstellen, muss der Blattschutz aufgehoben werden. Anschließend erstellen Sie einen neuen benutzerberechtigten Bereich über die `Add`-Methode des `AllowEditRange`-Objektes. Es ist zwingend erforderlich, den Bereichsnamen `Title` sowie den Bereich `Range`, der bearbeitet werden darf, anzugeben. Die Angabe eines Kennwortes ist optional. Für den Bereich `Data` vergeben wir das Passwort `Codebook`. Der Bereichsschutz für den hinzugefügten Bereich wird erst wirksam, wenn das Tabellenblatt geschützt wird. Hier verwenden wir, wie bereits erwähnt, das Passwort `VBA`.

Zur besseren Orientierung schreiben wir die Einstellungen des neu erstellten Bereichs `Data` in die Zellen `C17:C19` des Tabellenblattes `ADD_DELETE_AER`. Der `TITEL` sowie `BEREICH` wird über das `Protection`-Objekt ausgelesen. Die Passwörter des Tabellen- und Bereich-Schutzes können nicht ermittelt werden. Diese Angaben geben wir im Code vor. Im Anschluss daran wird ein Meldungsfenster angezeigt mit der Meldung, dass ein benutzerberechtigter Bereich erstellt wurde.

Achtung

Das AllowEditRange-Objekt ist in der Excel-Version 2002 neu hinzugekommen. Es wird in früheren Excel-Versionen nicht unterstützt. Verwenden Sie Makros mit diesem Objekt, kommt es in den Versionen bis einschließlich Excel 2000 zu Fehlermeldungen.

```
'=====
' Auf CD      Buchdaten\Beispiele\02_Allgemein\
' Dateiname  02_05_AllowEditRange.xls
' Tabelle     Add_Delete_AER
' Modul      mdl_01_Add_AER
'=====

Sub AddAllowEditRange()
  On Error GoTo ErrorHandler
  With ActiveSheet
    .Unprotect "VBA"                ' Blattschutz aufheben

    ' Neuer Benutzerberechtigter Bereich
    .Protection.AllowEditRanges.Add _
      Title:="Data", _
      Range:=Range("A1:A10"), _
      Password:="Codebook"

    ' Auflistung im Tabellenblatt
    .Range("C17").Value = .Protection.AllowEditRanges("Data") _
      .Title
    .Range("C18").Value = .Protection.AllowEditRanges("Data") _
      .Range.Address
    .Range("C19").Value = "Codebook"    ' Bereichs Paßwort

    .Protect "VBA"                ' Blattschutz setzen

    MsgBox _
      "Der benutzerberechtigte Bereich " & Range("C17").Value & _
      " wurde erstellt.", vbInformation
  Exit Sub
ErrorHandler:
  MsgBox _
    "Der benutzerberechtigte Bereich " & Range("C17").Value & _
    " existiert bereits.", vbExclamation
  .Protect "VBA"
End With
End Sub
```

55 Benutzerberechtigte Bereiche löschen

Bevor ein benutzerberechtigter Bereich gelöscht werden kann, muss ein eventuell vorhandener Blattschutz aufgehoben werden. Bei bestehendem Blattschutz würde der Debugger sonst eine Fehlermeldung ausgeben. Wenn Sie versuchen, einen nicht vorhandenen Bereich zu löschen, würde ebenfalls eine Fehlermeldung ausgegeben. Diese wird wie in dem vorherigen Beispiel durch eine On Error GoTo-Anweisung verhindert. Nachdem der Bereich Data gelöscht wurde, wird das Tabellenblatt wieder geschützt. Der Bereich C17:C19, welche die Einstellungen des Bereichs Data ent-

Grundlagen
Allgemein
Datum und Zeit
Formeln
Steuerungselemente
Befehlsliste
Objekte
Diagramme
Ereignisse
User-Forms
Web/Mail
Extern
Gemischtes
Spezial

hält, wird nach dem Löschvorgang des Bereichs Data ebenfalls gelöscht. Am Ende der Prozedur werden Sie über ein Meldungsfenster darüber informiert, ob der Löschvorgang des benutzerberechtigten Bereichs Data erfolgreich war.

```
'=====
' Auf CD      Buchdaten\Beispiele\02_Allgemein\
' Dateiname  02_05_AllowEditRange.xls
' Tabelle    Add_Delete_AER
' Modul      mdl_02_Delete_AER
'=====

Sub DeleteAllowEditRange()
    On Error GoTo ErrorHandler
    With ActiveSheet
        .Unprotect "VBA"
        .Protection.AllowEditRanges("Data").Delete
        .Range("C17:C19").ClearContents
        .Protect "VBA"

        MsgBox _
            "Der benutzerberechtigte Bereich " & "Data" & " " & _
            "wurde gelöscht.", vbInformation
    End With
Exit Sub

ErrorHandler:
    MsgBox _
        "Der benutzerberechtigte Bereich " & "Data" & " " & _
        "existiert nicht!", vbExclamation
    .Protect "VBA"
End With
End Sub
```

56 Schutz der benutzerberechtigten Bereiche aufheben

Sie können mit VBA den Schutz benutzerberechtigter Bereiche aufheben. Dazu steht Ihnen in der Objektbibliothek die `Unprotect`-Methode des `AllowEditRange`-Objektes zur Verfügung. In unserem Beispiel soll der Schutz des Bereichs DATA aufgehoben werden. Excel würde eine Fehlermeldung ausgeben, wenn der Bereich nicht existieren würde. Durch die vorangestellte `On Error GoTo`-Anweisung würde diese Fehlermeldung übergangen werden. Nachdem ein Fehler aufgetreten ist, wird die Prozedur zur Sprungmarke `ErrorHandler` verzweigt. Hier erfolgt in unserer Prozedur ein Meldungsfenster. Wurde der Bereichsschutz aufgehoben, kann anschließend der Bereich oder einzelne Zellen bearbeitet werden. In unserem Beispiel erhalten die erste und letzte Zelle des Bereichs einen Textwert.

```
'=====
' Auf CD      Buchdaten\Beispiele\02_Allgemein\
' Dateiname  02_05_AllowEditRange.xls
' Tabelle    Add_Delete_AER
' Modul      mdl_03_Unprotect_AER
'=====
```

<pre>Sub UnprotectAllowEditRange() Dim rngData As Range On Error GoTo ErrorHandler ActiveSheet.Protection.AllowEditRanges("Data"). _ Unprotect "Codebook" Set rngData = _ ActiveSheet.Protection.AllowEditRanges("Data").Range ' erste Zelle im Bereich Data beschreiben rngData(1, 1).Value = "erste Zeile" ' letzte Zelle im Bereich Data beschreiben rngData.Cells(rngData.Rows.Count, rngData.Columns.Count) _ .Value = "letzte Zeile" MsgBox _ "Der benutzerberechtigte Bereich ""Data"" ist nicht " & _ " geschützt.", vbInformation Exit Sub ErrorHandler: MsgBox _ "Der benutzerberechtigte Bereich" & " ""Data"" " & _ "existiert nicht", vbCritical End Sub</pre>	Grundlagen Allgemein Datum Zeit tungen Steuerelemente Befehle leisten Objekt Diagramm Ereignisse

57 Benutzerberechtigte Bereiche schützen

Wurde der Schutz eines benutzerberechtigten Bereichs einmal aufgehoben, lässt er sich nicht ohne weiteres wieder einschalten. In der Objektbibliothek steht zu der `Unprotect`-Methode des `AllowEditRange`-Objektes kein Gegenstück `Protect` zur Verfügung. Um einen Bereich dennoch erneut zu schützen, muss zunächst der Blattschutz aufgehoben und anschließend gleich wieder eingeschaltet werden. Geben Sie bei diesen Befehlen ein vorhandenes Passwort zum Blattschutz mit im Code an. Ein Kennwort des Bereichsschutzes muss nicht angegeben werden. Wird die Arbeitsmappe mit aufgehobenem Bereichsschutz geschlossen, dann wird der Schutz nach erneutem Öffnen automatisch wieder hergestellt.

<pre>'===== ' Auf CD Buchdaten\Beispiele\02_Allgemein\ ' Dateiname 02_05_AllowEditRange.xls ' Tabelle Add_Delete_AER ' Modul mdl_04_ReProtect_AER '===== Sub ReProtectAllowEditRange() With ActiveSheet .Unprotect "VBA" ' Blattschutz aufheben .Protect "VBA" ' Blattschutz setzen End With MsgBox _</pre>	User-Form Web/Mail Extern Gemischtes Spezial

```

        "Der benutzerberechtigte Bereich ""Data"" ist geschützt.", _
        vbInformation
    End Sub

```

58 Benutzerberechtigte Bereiche – Passwort ändern

Um das Kennwort für den Zugriff auf einen benutzerberechtigten Bereich zu ändern, rufen Sie den Dialog **BEREICH BEARBEITEN** über den Menüpunkt **EXTRAS | SCHUTZ | BENUTZER DÜRFEN BEREICHE BEARBEITEN Schaltfläche ÄNDERN** auf (siehe Abbildung 42).

Mit VBA greifen Sie für die Änderung des Passwortes eines benutzerdefinierten Bereichs auf die **ChangePassword-Methode** des **AllowEditRange-Objektes** zu.

In der Prozedur **ChangePasswordAER** verzweigt die anfängliche **On Error GoTo**-Anweisung auf die Sprungmarke **Errorhandler**, falls der Bereich **Data** nicht existiert. Vergessen Sie nicht, vor der Passwort-Änderung den Blattschutz des Tabellenblattes aufzuheben. Das ursprüngliche Passwort **Codebook** des benutzerberechtigten Bereichs **Data** wird in **neues Passwort** geändert. Anschließend wird der Blattschutz wieder eingestellt. Hier vergeben wir ebenfalls ein neues Kennwort, indem wir das neue Passwort **Excel** hinter der **Protect**-Eigenschaft aufführen. In unserem Beispiel werden die neuen Kennwörter zur Orientierung in die Zellen **C16** und **C19** der Tabelle geschrieben.

```

' =====
' Auf CD      Buchdaten\Beispiele\02_Allgemein\
' Dateiname   02_05_AllowEditRange.xls
' Tabelle     ChangePassword_AER
' Modul       mdl_05_ChangePasswordAER
' =====

Sub ChangePasswordAER()
    On Error GoTo ErrorHandler
    With ActiveSheet
        .Unprotect "VBA"
        .Protection.AllowEditRanges("Data"). _
            ChangePassword "neues Passwort"
        .Protect "Excel"

        .Range("C16") = "Excel"           ' Blattschutz
        .Range("C19") = "neues Passwort" ' Bereichsschutz
    End With
    Exit Sub

ErrorHandler:
    MsgBox _
        "Der benutzerberechtigte Bereich ""Data"" & _
        " existiert nicht!", vbCritical
End Sub

```

59 Benutzerberechtigte Bereiche – Titel ändern

Ab Excel 2002 kann man in der Excel-Anwendung über den Menüpunkt **EXTRAS | SCHUTZ | BENUTZER DÜRFEN BEREICHE BEARBEITEN | ÄNDERN** den **TITEL**, **BEZUG** und das **KENNWORT** eines benutzerberechtigten Bereichs ändern.



Abbildung 42: Ändern eines benutzerberechtigten Bereichs

Mit VBA können Sie standardmäßig lediglich das Passwort eines benutzerdefinierten Bereichs ändern (siehe Rezept 58). Zur Änderung des Titels und Bezugs steht im Objektkatalog keine Methode zur Verfügung. Auch in der Liste der integrierter Dialogfenster fehlt der Dialog **BEREICH BEARBEITEN** (siehe Abbildung 42). Um dennoch den Titel eines benutzerberechtigten Bereichs mühelos zu ändern, haben wir die benutzerdefinierte Funktion **AERChangeTitle** programmiert. Die Funktion ist vom Datentyp **Boolean** und liefert als Rückgabewert einen Wahrheitswert **True** oder **False**.

Beim Aufruf der Funktion müssen als Argumente der alte **strOldTitle** und der neue Bereichstitel **strNewTitle** übergeben werden. Die Angabe eines Passwortes **strSheetPassword** ist optional. Das bedeutet, dass eine Angabe nur erforderlich ist, wenn das Tabellenblatt mit einem Kennwort geschützt ist. Bei Angabe eines Passwortes wird das Tabellenblatt durch die Funktion mit diesem Kennwort geschützt. Dies geschieht unabhängig davon, ob das Tabellenblatt vorher geschützt war.

Der Wahrheitswert der Funktion ergibt sich daraus, ob der Titel eines übergebenen Bereichs erfolgreich geändert wurde. In einer **For Each...Next**-Anweisung-Schleife wird jeder benutzerberechtigte Bereich des aktiven Tabellenblattes durchlaufen. In einer **If...Then...Else**-Anweisung wird der gerade durchlaufene Bereichstitel mit der Variablen **strOldTitle** verglichen. Diese enthält den Namen des Bereichs, dessen Titel geändert werden soll. Wird der alte Bereichstitel in der Auflistung aller benutzerberechtigten Bereiche gefunden, erfolgt die Umbenennung. Dazu muss vorab der Blattschutz aufgehoben und anschließend wieder gesetzt werden. War die Umbenennung erfolgreich, wird der Rückgabewert der Funktion auf **True** gesetzt und die Funktion mit **Exit Function** verlassen. Wird der übergebene Bereich in der **For Each...Next**-Anweisung nicht gefunden, oder hat die Angabe eines falschen Passwortes zu einem Fehler geführt, wird die Schleife verlassen und zu der Sprungmarke **ERRORHANDLER** verwiesen. Der Rückgabewert der Funktion **AERChangeTitle** bleibt in diesen Fällen unverändert auf **False**.

```
' =====  
' Auf CD      Buchdaten\Beispiele\02_Allgemein\  
' Dateiname   02_05_AllowEditRange.xls  
' Tabelle     ChangeTitle_AER  
' Modul       mdl_06_ChangeTitleAER  
' =====  
  
Function AERChangeTitle(strOldTitle As String, _  
                        strNewTitle As String, _  
                        Optional strSheetPassword As String) _
```

- Grundlagen
- Allgemein
- Datum und Zeit
- Formeln
- Diagramme
- Erreignisse
- User-Formen
- Web/Mail
- Extern
- Gemischtes
- Spezial

As Boolean

```

Dim AER As AllowEditRange

On Error GoTo ErrorHandler

With ActiveSheet
    For Each AER In .Protection.AllowEditRanges
        If AER.Title = strOldTitle Then
            .Unprotect strSheetPassword
            AER.Title = strNewTitle
            .Protect strSheetPassword
            ' Funktionswert auf True setzen
            AERChangeTitle = True
        Exit Function
    End If
Next
End With
ErrorHandler:
End Function

```

Um einen Bereichstitel über die AERChangeTitle-Funktion zu ändern, rufen Sie die Funktion in einem Makro auf und übergeben die benötigten Argumente. Beim Schreiben der Argumente hilft Ihnen die Quick- bzw. Parameterinfo bei der Zuweisung der Argumente. Das Makro in Abbildung 43 ändert den Bereichstitel Data in MyData. Wenn der Bereich Data nicht im aktiven Tabellenblatt existiert, oder das Passwort falsch eingegeben wurde, erfolgt keine Fehlermeldung.



Abbildung 43: Parameterinfo der Funktions-Argumente

In der folgenden Prozedur ChangeDataTitel wird die AERChangeTitle-Funktion in einer If...Then...Else-Anweisung aufgerufen. Es wird überprüft, ob der Rückgabewert der Funktion mit den angegebenen Parametern False ist, also die Ausführung der Funktion fehlerhaft war.

Ebenso können Sie den Funktions-Wert False über den Not-Operator überprüfen:

```
If Not AERChangeTitle("Data", "Daten", "VBA") Then ' erwartet False
```

Den Rückgabewert True ermitteln Sie mit:

```
If AERChangeTitle("Data", "Daten", "VBA") Then ' erwartet True
```

Im Anschluss an den Funktionsaufruf erfolgt eine Meldung, ob die Umbenennung erfolgreich war. Zur Orientierung wird der neue Titel des benutzerberechtigten Bereichs in die Zelle C17 geschrieben.

```
' =====  
' Auf CD      Buchdaten\Beispiele\02_Allgemein\  
' Dateiname   02_05_AllowEditRange.xls  
' Tabelle     ChangeTitle_AER  
' Modul       mdl_06_ChangeTitleAER  
' =====  
  
Sub ChangeDataTitel()  
    If AERChangeTitle("Data", "MyData", "VBA") = False Then  
  
        MsgBox _  
            "Der benutzerberechtigte Bereich ""Data"" " & _  
            "existiert nicht," & _  
            vbNewLine & _  
            "oder das Passwort des Blattschutzes ist falsch!", _  
            vbCritical, "Fehlermeldung"  
    Else  
        With ActiveSheet  
            .Range("C17").Value = .Protection.AllowEditRanges(1).Title  
        End With  
    End If  
End Sub
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

60 Benutzerberechtigte Bereiche – Bereich ändern

Für die Änderung des Zellenbereichs eines benutzerberechtigten Bereichs sieht das AllowEditRange-Objekt keine Methode vor. Wie schon für den Bereichstitel haben wir deshalb für die Änderung des Zellenbereichs die benutzerdefinierte Funktion AERChangeRange programmiert. Diese ermöglicht es, die Bereichsangaben eines benutzerberechtigten Bereichs zu ändern.

Der Aufbau der Funktion AERChangeRange ähnelt der Funktion AERchangeTitle zur Änderung des Bereichstitels aus dem Rezept 59. Zur Identifizierung des benutzerberechtigten Bereichs muss strTitle als erstes Argument beim Aufruf an die Funktion übergeben werden. Das zweite Argument rngNewRange enthält den neuen Zellenbereich. Hier kann entweder ein zusammenhängender Zellbereich oder mehrere Bereiche durch Semikolons voneinander getrennt angegeben werden. Die Angabe eines Kennwortes ist optional und richtet sich danach, ob das Tabellenblatt mit einem Passwort geschützt ist.

Sowohl benutzerdefinierte Funktionen als auch die Prozeduren zum Aufruf der Funktion werden in Standardmodulen hinterlegt. Es ist nicht erforderlich, dass das Aufrufmakro und die Funktion in dem gleichen Modul sind.

Abbildung 44 zeigt den Aufruf der Funktion über ein Makro. Die Parameterinfo leistet wiederum Hilfe bei der Angabe der Argumente.

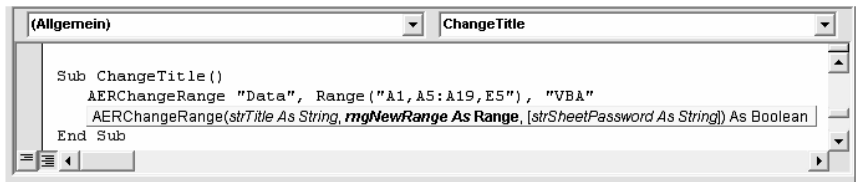


Abbildung 44: Parameterinfo der Funktions-Argumente

```

' =====
' Auf CD      Buchdaten\Beispiele\02_Allgemein\
' Dateiname   02_05_AllowEditRange.xls
' Tabelle     ChangeRange_AER
' Modul       mdl_07_ChangeRangeAER
' =====

Function AERChangeRange(strTitle As String, _
                        rngNewRange As Range, _
                        Optional strSheetPassword As String) _
                        As Boolean

    Dim rngAER As AllowEditRange

    On Error GoTo ErrorHandler

    With ActiveSheet
        For Each rngAER In .Protection.AllowEditRanges
            If rngAER.Title = strTitle Then
                .Unprotect strSheetPassword
                Set rngAER.Range = rngNewRange
                .Protect strSheetPassword
                ' Funktionswert auf True setzen
                AERChangeRange = True
                Exit Function
            End If
        Next
    End With
ErrorHandler:
End Function

```

In der folgenden Prozedur soll der benutzerdefinierte Bereich `Data` geändert werden. Zunächst wird der aktuelle Bereich im Tabellenblatt markiert. Anschließend wird ein Eingabedialog `InputBox` aufgerufen. Als Default-Wert werden die Bereiche `"A1:A5;I25;K25"` angegeben. Diese Zellbereiche werden beim Aufruf der `InputBox` durch Strichumrandungen der Zellen und Bereiche angezeigt. Das Argument `Type` wird auf 8 gesetzt, damit die `InputBox` ein `Range`-Objekt zurückgibt. Mit Hilfe der `Set`-Anweisung weisen wir das Ergebnis dem `Range`-Objekt `rngInputBox` zu. Wenn Sie keine `Set`-Anweisung verwenden, wird die Variable auf den Wert im abgegebenen Bereich statt auf das `Range`-Objekt gesetzt.

Vor dem Aufruf der `InputBox` haben wir die Anweisung `On Error Resume Next` geschrieben. Diese Anweisung ist notwendig, falls der Anwender in der `InputBox` die Schaltfläche `ABBRECHEN` klickt. Dadurch kann der Variablen `rngInputBox` kein Wert zugewiesen werden. Das wiederum würde zu einer Fehlermeldung führen. Im Anschluss an die `InputBox` wird über eine `If...Then...Else`-Anweisung überprüft, ob `rngInputBox` einen Wert oder `Nothing` enthält.

Über eine weitere `If...Then...Else`-Anweisung wird die Funktion `AERChangeRange` aufgerufen. Dabei übergeben wir folgende Argumente:

- ▶ den Bereichstitel `Data`
- ▶ den neuen Bereich mittels der Variablen `rngInputBox`
- ▶ das Passwort VBA des Tabellenblattes `CHANGERANGE_AER`

Die Funktion `AERChangeRange` gibt den Wert `False` zurück, wenn bei deren Abarbeitung ein Fehler aufgetreten ist. Demzufolge wird ein Meldungsfenster angezeigt. War die Änderung des Bereichs erfolgreich, wird die neue Bereichsadresse zur Orientierung in die Zelle C18 der Tabelle geschrieben. Zum Abschluss wird der neue Bereich im Tabellenblatt markiert.

```
' =====  
' Auf CD      Buchdaten\Beispiele\02_Allgemein\  
' Dateiname  02_05_AllowEditRange.xls  
' Tabelle    ChangeTitle_AER  
' Modul      mdl_07_ChangeRangeAER  
' =====
```

```
Sub ChangeDataRange()  
    Dim rngInputBox As Range  
  
    ' Aktuellen »Data«-Bereich markieren  
    ActiveSheet.Protectio.AllowEditRanges("Data").Range.Select  
  
    On Error Resume Next  
    ' Neuen Bereich über InputBox wählen  
    Set rngInputBox = Application.InputBox _  
        ("Markieren Sie den neuen Zellbereich mit der Maus." & _  
        vbNewLine & _  
        "Trennen Sie nicht zusammenhängende Bereiche mit einem " & _  
        "Semikolon.", _  
        "Bereich für " &"Data" ändern", _  
        Default:="A1:A5;I25;K25", _  
        Type:=8)  
  
    If rngInputBox Is Nothing Then Exit Sub  
  
    If AERChangeRange("Data", rngInputBox, "VBA") = False Then  
  
        MsgBox _  
            "Der benutzerberechtigte Bereich " &"Data" " & _  
            "existiert nicht," & _  
            vbNewLine & _  
            "oder das Passwort des Blattschutzes ist falsch!", _  
            vbCritical, "Fehlermeldung"  
    Else  
        With ActiveSheet  
            .Range("C18").Value = _  
                .Protection.AllowEditRanges(1).Range.Address  
            ' Neuen »Data«-Bereich markieren  
            .Protection.AllowEditRanges("Data").Range.Select  
        End With  
    End If  
End Sub
```

61 Ausgewählte Bereiche freigeben

Die Funktion `BENUTZERBERECHTIGUNGEN ZUM BEARBEITEN VON BEREICHEN` ist erst ab Excel 2002 als feste Funktion in Excel integriert. Das bedeutet aber nicht, dass Sie in den vorherigen Excel-

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

Versionen auf diese Möglichkeit verzichten müssen. Die Freigabe definierter Bereiche können Sie ebenso über eine VBA-Prozedur steuern. Das nachfolgende Beispiel ist für alle Excel-Versionen ab Excel 97 geeignet.

In einem Tabellenblatt (die Mappe liegt auf dem Server und wird von mehreren Usern genutzt) sollen bestimmte Zellen bzw. Zeilen/Spalten geschützt werden, so dass diese nur gelesen und nicht geändert werden können. In Abhängigkeit des Benutzernamens (Netzwerkanmeldung unter Windows) sollen bestimmte Zellen bzw. Spalten automatisch beim Öffnen freigegeben werden.

Hinweis

In Excel 2002 müssen Sie mit Windows 2000 arbeiten, um bestimmten Benutzern über die Funktion `BENUTZERBERECHTIGUNGEN ZUM BEARBEITEN VON BEREICHEN` den Zugriff auf Bereiche zu erteilen. Verwenden Sie ein anderes Betriebssystem, bietet Ihnen dieses Beispiel eine Alternative, um über die `Environ`-Funktion Bereiche für bestimmte Benutzer freizugeben.

Zunächst müssen die freizugebenden Bereiche je Benutzer festgelegt werden. Hierzu verwenden Sie `BENANNTRE BEREICHE`. Markieren Sie, während Sie die `STRG`-Taste gedrückt halten, nacheinander alle Zellen bzw. Zeilen/Spalten, welche durch einen Benutzer geändert werden dürfen. Schreiben Sie anschließend in das Namensfeld (links neben der Bearbeitungszeile) den Usernamen (Netzwerkanmeldung unter Windows) und bestätigen Sie die Eingabe mit der `RETURN`-Taste. Bereichsnamen dürfen keine Leerzeichen enthalten. Wenn in einem Anmeldenamen Leerschritte enthalten sind, müssen diese bei der Vergabe eines Bereichsnamens durch Unterstriche ersetzt werden (»Melanie Breden« als »Melanie_Breden«). Wiederholen Sie diese Schritte für weitere Benutzer.

Soll es möglich sein, dass mehrere User den gleichen Bereich bearbeiten können, also die Zuweisung von mehreren Namen zu einem Bereich, dann wählen Sie im Namensfeld einen bereits vorhandenen Bereichsnamen aus. Der entsprechende Bereich wird daraufhin markiert. Schreiben Sie jetzt einen anderen Namen ins Namensfeld und bestätigen Sie Ihre Eingabe wieder mit der `RETURN`-Taste. Beide Namen verweisen jetzt auf den identischen Bereich.

Die Vergabe eines Bereichsnamens können Sie ebenso über VBA vornehmen.

```
Dim rngMyRange As Range
Set rngMyRange = Range("A1:B20,D6,D9,D13")
ActiveWorkbook.Names.Add Name:="Monika_Weber", RefersTo:=rngMyRange
```

Noch einfacher ist die Namensvergabe über das Direktfenster. Ausführliche Informationen zum Direktfenster finden Sie in der Kategorie »Allgemein«.

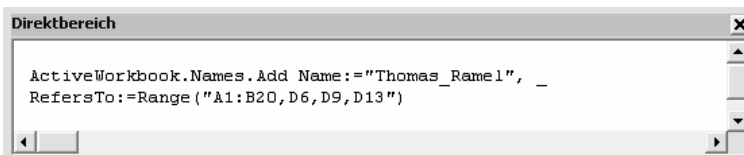


Abbildung 45: Schnelle Namensvergabe

Beim Öffnen der Excel-Arbeitsmappe wird das `Workbook_Open`-Ereignis ausgelöst. In dieser Prozedur wird zunächst das betreffende Tabellenblatt an die Variable `wks` übergeben. Anschließend

wird der unter Windows angemeldete Benutzername über die `Environ`-Funktion ausgelesen und in der Variablen `strName` gespeichert.

Tipp

Weitere Informationen zu der `Environ`-Funktion erhalten Sie im Rezept 64 und in der Kategorie »Einstieg«.

Um bei der Suche nach dem dazugehörigen Bereichsnamen Fehlermeldungen zu vermeiden, werden über die Funktion `Wechseln` mögliche Leerzeichen im Benutzernamen durch Unterstriche ersetzt. Tabellenblattfunktionen müssen im VBA-Code mit ihren englischen Bezeichnungen aufgerufen werden.

```
strName = Application.WorksheetFunction.Substitute _  
    (Environ("UserName"), " ", "_")
```

Hinweis

Eine Übersicht mit Übersetzung der meisten Funktionen und Schlüsselwörter ist in der Datei `VBALISTE.XLS` aufgeführt. Bei einer vollständig installierten Excel-Anwendung befindet sich diese Datei auf Ihrem Rechner in dem Installationsordner. Der Standardpfad lautet:

`C:\Programme\Microsoft Office\Office\1031VBALISTE.XLS`

In Tabellenblatt `SHEETNAMEDRANGE` wird der Blattschutz aufgehoben und anschließend alle Zellen gesperrt. Danach wird der definierte Bereich des Benutzernamens, welcher in der Variablen `strName` enthalten ist, entsperrt. Über die `EnableSelection`-Eigenschaft wird festgelegt, welche Zellen im Tabellenblatt markiert werden können.

Diese Eigenschaft ist nur wirksam, wenn das Arbeitsblatt geschützt ist. Infolgedessen wird das Tabellenblatt geschützt und das Passwort `Codebook` vergeben. Um anzuzeigen, welche Zellen im Tabellenblatt freigegeben wurden, werden diese über die `GoTo`-Methode markiert.

```
Application.GoTo Reference:=(strName)
```

Wenn der freigegebene Bereich nicht markiert werden soll, können Sie diesen Befehl auch auskommentieren oder löschen. Er beeinflusst nicht die Funktionalität des Beispiels.

Wird die Datei durch einen Benutzer geöffnet, für den kein Bereich definiert wurde, würde das Makro bei dem Versuch, den nach ihm benannten Bereich freizugeben, eine Fehlermeldung ausgeben. Mit der `On Error Resume Next`-Anweisung wird sichergestellt, dass der Code dennoch weiter ausgeführt wird, damit das Tabellenblatt nach einem eventuell auftretenden Fehler wieder geschützt wird. Vor Beendigung der Prozedur wird über die `Number`-Eigenschaft des `Err`-Objektes geprüft, ob innerhalb der Prozedur ein Fehler aufgetreten ist. In dem Fall wird eine Meldung ausgegeben, dass für den Benutzer der Datei kein freigegebener Bereich festgelegt ist.

```
If Err.Number > 0 Then MsgBox "Es ist kein Bereich freigegeben"
```

Natürlich möchten wir Ihnen dieses Beispiel in einer Excel-Datei demonstrieren. Da wir aber Ihren Anmeldenamen nicht kennen, würde es zu einer Fehlermeldung kommen, wenn der freizugebende Bereich nach dem Autor des Beispiels benannt wäre. In der Beispieldatei wurde deshalb ein Bereich mit dem Namen `NamedRange` definiert. Zu Beginn der Prozedur wird der Beispieldatei

Grundlagen
Allgemein
Datum und Zeit
Formeln
Diagramme
Objekt
Diagramm
Ereignisse
User-Form
Web/Mail
Extern
Gemischtes
Spezial

ein benannter Bereich hinzugefügt. Mittels der `Environ`-Funktion wird Ihr Benutzername ausgelesen und über die Variable `strName` als Bereichsname vergeben. Als Bezug, für den nach Ihnen benannten Bereich, wird der bereits vorhandene Bereichsname `NamedRange` angegeben.

```
ActiveWorkbook.Names.Add Name:=strName, _
                          RefersTo:=Range("NamedRange")
```

Wenn Sie dieses Beispiel in reellen Excel-Dateien verwenden möchten, benötigen Sie keinen definierten Bereich namens `NamedRange` und löschen obige Codezeilen aus der Prozedur.

```
' =====
' Auf CD      Buchdaten\Beispiele\02_Allgemein\
' Dateiname   02_06_UserEditRange.xls
' Ereignis    DieseArbeitsmappe
' =====
```

```
Private Sub Workbook_Open()
    Dim wks      As Worksheet
    Dim strName As String

    ' Tabellenblattname anpassen
    Set wks = Worksheets("SheetNamedRange")

    ' Windows-Anmeldenamen in Variable speichern
    ' mögliche Leerschritte werden durch Unterstriche ersetzt
    strName = Application.WorksheetFunction.Substitute _
              (Environ("UserName"), " ", "_")

    On Error Resume Next

    ' #####
    ' Um dieses Beispiel zu testen, wird dem unter Windows
    ' angemeldeten Benutzer ein vorhandener benannter Bereich
    ' zugewiesen.
    ' Wenn Sie dieses Beispiel in reellen Excel-Dateien verwenden
    ' möchten, löschen Sie folgende Codezeilen aus der Prozedur.

    ActiveWorkbook.Names.Add Name:=strName, _
                              RefersTo:=Range("NamedRange")

    ' #####

    With wks
        ' Blattschutz aufheben
        .Unprotect "Codebook"
        ' alle Zellen sperren
        .Cells.Locked = True
        ' Bereich nach Username entsperren
        .Range(strName).Locked = False
        ' alle gesperrten Zellen nicht selektierbar
        .EnableSelection = xlUnlockedCells
        ' Blattschutz setzen
```

```
.Protect "Codebook"
End With
```

```
' Freigegebenen Bereich des Benutzers selektieren
Application.GoTo Reference:=(strName)
```

```
' Meldung wenn kein Bereich zu Username definiert wurde
If Err.Number > 0 Then MsgBox "Es ist kein Bereich freigegeben"
End Sub
```

62 Löschen von Blättern verhindern

In Excel ist es möglich, eine gesamte Arbeitsmappe vor Veränderungen an der Struktur zu schützen. Aktivieren Sie dazu im Menü EXTRAS | SCHUTZ | ARBEITSMAPPE SCHÜTZEN die Option STRUKTUR. Im Kontextmenü der Registerreiter wie auch in den Menüoptionen werden alle Funktionen, welche die Struktur verändern können, deaktiviert (siehe Abbildung 46).

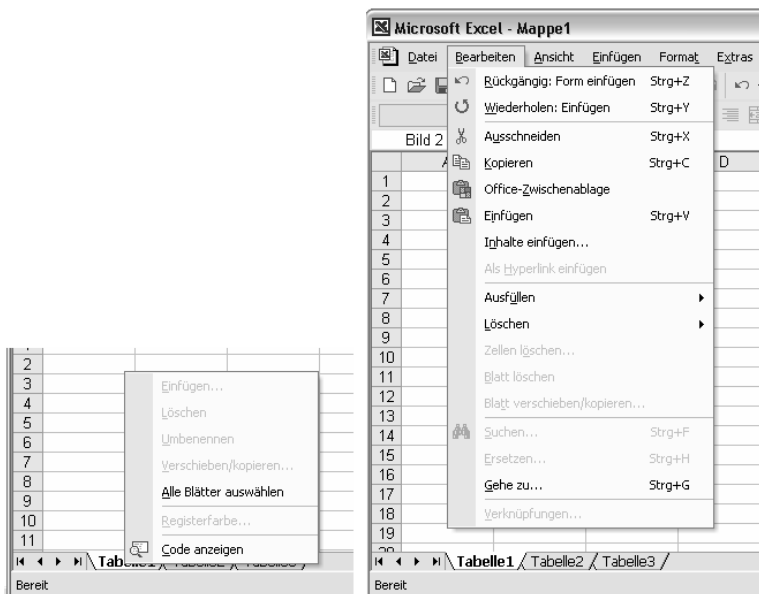


Abbildung 46: Deaktivierte Optionen beim Arbeitsmappenschutz

Wenn der Arbeitsmappenschutz aktiv ist, sind folgende Funktionen weder in Excel noch mit VBA ausführbar:

- ▶ Blätter einfügen/löschen
- ▶ Blätter umbenennen
- ▶ Blätter verschieben/kopieren
- ▶ Blätter ausblenden/einblenden
- ▶ Registerfarben ändern
- ▶ Veränderungen in den Eigenschaften der Arbeitsmappe

Grundlagen

Allgemein

Datenzeit

Funktionen

Steuerelemente

Befehle ausführen

Objekt

Diagramm

Ereignisse

User-Form

Web/Mail

Extern

Gemischtes

Spezial

Der Arbeitsmappenschutz gilt mappenweit, also für alle vorhandenen Arbeitsblätter. Er ist unabhängig vom Blattschutz einsetzbar.

Einzelne Blätter lassen sich nur aus VBA heraus vor dem Umbenennen, Löschen etc. schützen. Dazu wird bei einem Blattwechsel im `Workbook_SheetActivate`-Ereignis anhand des Blattnamens überprüft, welches Blatt aktiviert wurde. Die Namen der zu schützenden Blätter sind in einer `If ... Then ... Else`-Anweisung hinterlegt. Wenn das aktivierte Blatt aufgeführt ist, wird für die gesamte Arbeitsmappe der Strukturschutz aktiviert. Damit der Anwender nicht die Möglichkeit hat, zu experimentieren, wie er diesen Schutz umgehen kann, blenden wir zusätzlich das Kontextmenü der Registerreiter `Ply` aus.

Wenn Sie ein Arbeitsblatt aktivieren, welches nicht im Code aufgeführt ist, wird der Blattschutz und damit gleichzeitig der Strukturschutz aufgehoben.

Bei unseren Tests hat sich herausgestellt, dass es mit etwas Geschick und Schnelligkeit möglich ist, gesicherte Blätter zu verschieben. Um dies zu erreichen halten Sie beim Wechsel von einem ungeschütztem zu einem geschütztem Blatt die linke Maustaste gedrückt. Ziehen Sie dann den Registerreiter schnell an eine andere Position. Ansonsten haben wir keine weiteren Sicherheitslücken feststellen können.

```
'=====
' Auf CD      Buchdaten\Beispiele\02_Allgemein\
' Dateiname   02_07_WorkbookProtection.xls
' Ereignis    DieseArbeitsmappe
'=====
```

```
Private Sub Workbook_SheetActivate(ByVal Sh As Object)

    With ThisWorkbook
        If Sh.Name = "Tabelle1" Or _
           Sh.Name = "Tabelle3" Then
            .Protect Password:="Codebook", Structure:=True
            Application.CommandBars("Ply").Enabled = False
        Else
            .Unprotect Password:="Codebook"
            Application.CommandBars("Ply").Enabled = True
        End If
    End With

End Sub
```

Beim Wechsel in eine andere Arbeitsmappe muss sichergestellt werden, dass das Kontextmenü der Registerreiter wieder eingeblendet wird. Einen Mappenwechsel überwachen wir mit dem `Workbook_Deactivate`-Ereignis. Sobald dieses Ereignis eintritt, wird die Anweisung zur Aktivierung des Registerreiter-Kontextmenüs ausgeführt.

```
'=====
' Auf CD      Buchdaten\Beispiele\02_Allgemein\
' Dateiname   02_07_WorkbookProtection.xls
' Ereignis    DieseArbeitsmappe
'=====
```

```
Private Sub Workbook_Deactivate()  
    Application.CommandBars("Ply").Enabled = True  
End Sub
```

63 Schreibgeschützte Datei ohne Speichern-Rückfrage schließen

Den Schreibschutz-Status einer Excel-Arbeitsmappe können Sie im Windows-Explorer über das Eigenschaftsfenster der Datei vornehmen (siehe Abbildung 47).

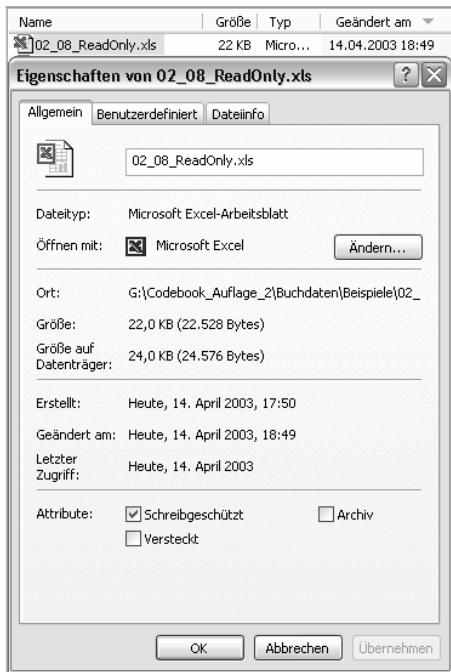


Abbildung 47: Schreibschutz einstellen

Ob eine Excel-Arbeitsmappe schreibgeschützt ist, können Sie mit VBA ermitteln und entsprechend auf deren Einstellung reagieren.

Folgende Prozedur gibt beim Öffnen der Arbeitsmappe die Meldung aus, ob das Attribut SCHREIBGESCHÜTZT aktiviert ist. Der Wahrheitswert dieser Option wird in der IIF-Funktion ausgelesen und der entsprechende Rückgabewert an die Funktion zurückgegeben.

```
'=====
' Auf CD      Buchdaten\Beispiele\02_Allgemein\
' Dateiname   02_08_ReadOnly.xls
' Ereignis    DieseArbeitsmappe
'=====

Private Sub Workbook_Open()  
    MsgBox _
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

```

        "Die Datei  "" & ThisWorkbook.Name & "" ist " & _
        vbNewLine & vbNewLine & _
        IIf(ThisWorkbook.ReadOnly, _
            "schreibgeschützt", _
            "nicht schreibgeschützt")
    End Sub

```

In einer schreibgeschützten Datei können Sie Veränderungen vornehmen, oder die Datei kann durch Verknüpfungen aktualisiert werden. Nach jeglichen Änderungen fragt Excel standardmäßig beim Schließen der Datei, ob diese gespeichert werden soll. Solange die Datei als *schreibgeschützt* geöffnet wurde, kann auf diese Meldung eigentlich verzichtet werden. Für die Umsetzung verwenden wir das `Workbook_BeforeClose`-Ereignis, welches in das Codeblatt von *DIESEARBEITSMAPPE* geschrieben wird. Die Speicher-Abfrage richtet sich danach, welchen Wahrheitswert die `Saved`-Eigenschaft des `Workbook`-Objektes enthält. Wurden Änderungen in der Datei vorgenommen, ist dessen Wert `False` und Excel fragt, ob Sie die Datei speichern möchten. Demzufolge übergeben wir den Wahrheitswert der `ReadOnly`-Eigenschaft an die `Saved`-Eigenschaft. Ist der `ReadOnly`-Wert `True`, dann ist die `Saved`-Eigenschaft ebenfalls `True`. Die Speichern-Abfrage vor dem Schließen der Datei fällt dann aus. Umgekehrt wird bei einer *nicht* schreibgeschützten Datei der `Saved`-Wert auf `False` gesetzt und Sie werden nach einer Speicherung gefragt.

```

'=====
' Auf CD      Buchdaten\Beispiele\02_Allgemein\
' Dateiname   02_08_ReadOnly.xls
' Ereignis     DieseArbeitsmappe
'=====

Private Sub Workbook_BeforeClose(Cancel As Boolean)
    ThisWorkbook.Saved = ThisWorkbook.ReadOnly
End Sub

```

64 Entfernen des Schreibschutzes ignorieren

Im Windows-Explorer kann über das Eigenschaften-Fenster einer Excel-Arbeitsmappe (siehe Abbildung 47) der Schreibschutz einer Datei aufgehoben werden. Dadurch kann die Datei geöffnet, Änderungen vorgenommen und die Datei gespeichert werden. Unabhängig von der Schreibschutz-Einstellung im Eigenschaftsfenster einer Datei ist es mit VBA auf einfache Weise möglich, eine Excel-Datei nur im schreibgeschützten Modus zu öffnen.

In folgender Prozedur wird beim Öffnen der Beispieldatei die Prozedur `MyReadOnly` aufgerufen.

```

'=====
' Auf CD      Buchdaten\Beispiele\02_Allgemein\
' Dateiname   02_09_ProtectReadOnly.xls
' Ereignis     DieseArbeitsmappe
'=====

Private Sub Workbook_Open()
    MyReadOnly
End Sub

```

Wird der Schreibschutz für eine Datei aktiviert, welche bereits schreibgeschützt ist, gibt Excel eine Fehlermeldung aus. Bevor in der Prozedur `MyReadOnly` der Schreibschutz aktiviert werden kann, muss deshalb über die `ReadOnly`-Eigenschaft der eingestellte Schreibschutz-Status ausgelesen werden. Wenn die Datei nicht schreibgeschützt ist, wird über die `ChangeFileAccess`-Methode die Zugriffsberechtigung zu der Arbeitsmappe auf *Schreibgeschützt xlReadOnly* eingestellt.

```
' =====
' Auf CD      Buchdaten\Beispiele\02_Allgemein\
' Dateiname  02_09_ProtectReadOnly.xls
' Modul      mdl_01_ReadOnly
' =====

Sub MyReadOnly()
    If ThisWorkbook.ReadOnly = False Then
        ThisWorkbook.ChangeFileAccess _
            Mode:=xlReadOnly
    End If
End Sub
```

Die Prozedur `MyReadWrite` hebt den Schreibschutz wieder auf. Auf dem Tabellenblatt der Beispieldatei haben wir zwei Schaltflächen zum Aktivieren und Deaktivieren des Schreibschutzes erstellt. Diesen sind die beiden Prozeduren zugewiesen.

```
' =====
' Auf CD      Buchdaten\Beispiele\02_Allgemein\
' Dateiname  02_09_ProtectReadOnly.xls
' Modul      mdl_01_ReadOnly
' =====

Sub MyReadWrite()
    If ThisWorkbook.ReadOnly = True Then
        ThisWorkbook.ChangeFileAccess _
            Mode:=xlReadWrite
    End If
End Sub
```

65 Benutzerdefinierte Ansichten schützen

In Excel können Sie in einer Arbeitsmappe verschiedene Ansichten in einer Excel-Datei speichern. Um eine benutzerdefinierte Ansicht zu erstellen, blenden Sie die gewünschten Zeilen, Spalten und Blätter ein oder aus. Auch die Druck- und Filtereinstellungen sowie das aktive Tabellenblatt und die aktuelle Zellposition können in der Ansicht hinterlegt werden. Wählen Sie im Menü **ANSICHT** den Dialog **BENUTZERDEFINIERT ANSICHTEN**, klicken auf die Schaltfläche **HINZUFÜGEN** und geben einen Namen für die Ansicht ein.

Mit VBA vergeben Sie mit folgender Syntax für die aktuelle Ansicht den Namen »UserA«. Die Optionen **DRUCKEINSTELLUNGEN** und **AUSGEBLENDETE ZEILEN** werden aktiviert.

Grundlagen

Allgemein

Datum
Zeit

Formeln

Steuerelemente

Befehle
Leistungen

Objekte

Diagramme

Ereignisse

User-Formen

Web/
Mail

Extern

Gemischtes

Spezial



Abbildung 48: Aktuelle Ansicht speichern

```
ActiveWorkbook.CustomViews.Add _
    ViewName:="UserA", _
    PrintSettings:=True, _
    RowColSettings:=True
```

Um eine benutzerdefinierte Ansicht anzuzeigen, wählen Sie im Dialog laut Abbildung 49 eine Ansicht aus und klicken auf Anzeigen. Das Tabellenblatt darf bei dieser Funktion nicht geschützt sein.

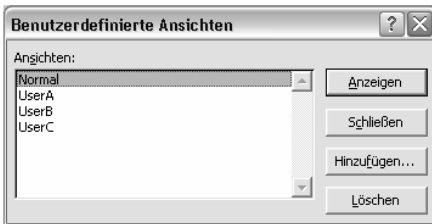


Abbildung 49: Gespeicherte Ansichten anzeigen

Mit VBA zeigen Sie eine benutzerdefinierte Ansicht mit folgender Syntax an:

```
ActiveWorkbook.CustomViews("UserA").Show
```

In unserem Beispiel wurden in der Beispieldatei vier Ansichten gespeichert. Die NORMAL-Ansicht zeigt alle Spalten des Tabellenblattes an. In den drei weiteren Ansichten wurden einzelne Spalten aus- oder eingeblendet. Die drei Ansichten sind für verschiedene Benutzer (UserA, UserB und UserC) vorgesehen und tragen die entsprechenden Namen der Anwender. Jeder dieser Benutzer soll seine Ansicht nur mit der Eingabe eines Passwortes aufrufen können. Damit der Anwender den Dialog BENUTZERDEFINIERT ANSICHTEN nicht aufrufen kann, wird der Menüpunkt beim Öffnen der Datei im Workbook_Activate-Ereignis deaktiviert. Im Workbook_Deaktivieren-Ereignis wird diese Option wieder aktiviert, damit der Dialog in anderen Arbeitsmappen wieder zur Verfügung steht.

```
' =====
' Auf CD      Buchdaten\Beispiele\02_Allgemein\
' Dateiname   02_10_CustomViewsWithPassword.xls
' Ereignis    DieseArbeitsmappe
' =====
```

```
Private Sub Workbook_Activate()  
    Application.CommandBars("View") _  
        .Controls(GetControlCaption).Enabled = False  
End Sub  
  
Private Sub Workbook_Deactivate()  
    Application.CommandBars("View") _  
        .Controls(GetControlCaption).Enabled = True  
End Sub
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

Die Beschriftung des Menüpunktes zum Aufruf des Dialoges Benutzerdefinierte Ansichten (siehe Abbildung 49) ist in Excel 97, 2000 und 2002/2003 unterschiedlich. Damit es nicht zu Fehlermeldungen kommt, prüfen wir in der benutzerdefinierten GetControlCaption-Funktion, welche Excel-Version im Einsatz ist. An die Funktion wird entsprechend der ermittelten Excel-Version der jeweilige Beschriftungstext des Menüpunktes zurückgegeben.

```
' =====  
' Auf CD      Buchdaten\Beispiele\02_Allgemein\  
' Dateiname   02_10_CustomViewsWithPassword.xls  
' Modul       mdl_01_GetControlCaption  
' =====  
  
Public Function GetControlCaption() As String  
    Select Case Val(Application.Version)  
        Case 8      ' EXCEL 97  
            GetControlCaption = "Ansicht anpassen..."  
        Case 9      ' EXCEL 2000  
            GetControlCaption = "Benutzerdefinierte Ansicht..."  
        Case 10, 11 ' EXCEL 2002/2003  
            GetControlCaption = "Benutzerdefinierte Ansichten..."  
    End Select  
End Function
```

Die Prozedur zum Wechsel einer benutzerdefinierten Ansicht haben wir in der Beispieldatei einer Schaltfläche zugewiesen. Oberhalb der Prozedur sind für jede Ansicht zwei Konstanten deklariert worden. Diese enthalten den Namen der Ansicht sowie das Passwort. Die Werte können in den Deklarierungszeilen beliebig geändert werden und erfordern keine weiteren Korrekturen in der Prozedur. Achten Sie jedoch darauf, dass die angegebenen Ansichtsnamen in der Arbeitsmappe vorhanden sind.

Um zur Ansicht *Normal* zu wechseln, geben Sie in der ersten InputBox das Wort »Normal« ein. Der Eingabewert wird in der Variablen strUser gespeichert. Wurde in der ersten InputBox auf ABBRECHEN geklickt, oder enthält diese keine Eingabe, wird die Prozedur in der ersten If...Then...Else-Anweisung beendet. In der folgenden If...Then...Else-Anweisung wird überprüft, ob der eingegebene Wert einer definierten Ansicht entspricht. Geben Sie anschließend in der zweiten InputBox das hinterlegte Passwort zu der Ansicht ein; für die *Normal*-Ansicht den Buchstaben »N«. Diese Eingabe wird zur späteren Überprüfung in der Variablen strPw hinterlegt.

Zum Schluss der Prozedur wird in einer Select Case-Anweisung der Name der Ansicht über die Variable strUser ausgewertet. In einzelnen Case-Abschnitten werden die vier gespeicherten Ansichten abgefragt. Trifft ein Ansichtsnamen auf den Textausdruck strUser zu, wird die Prozedur

zur Überprüfung des Passwortes aufgerufen. Dabei wird über die Variable `strPw` das eingegebene Passwort an die aufgerufene Prozedur übergeben.

```
'=====
' Auf CD      Buchdaten\Beispiele\02_Allgemein\
' Dateiname  02_10_CustomViewsWithPassword.xls
' Modul      mdl_02_MyCustomViews
'=====

Private strUser As String
Private strPw   As String

Const strNormal As String = "Normal"
Const strPwN    As String = "N"

Const strUserA  As String = "UserA"
Const strPwA    As String = "A"

Const strUserB  As String = "UserB"
Const strPwB    As String = "B"

Const strUserC  As String = "UserC"
Const strPwC    As String = "C"

Sub MyCustomViews()
    strUser = Application.InputBox( _
        "Geben Sie einen Ansichtsnamen ein:" & _
        vbNewLine & _
        strNormal & vbNewLine & _
        strUserA & vbNewLine & _
        strUserB & vbNewLine & _
        strUserC, _
        "Ansichten", "User", Type:=2)
    If strUser = "Falsch" Or strUser = "" Then Exit Sub

    If strUser <> strNormal And _
        strUser <> strUserA And _
        strUser <> strUserB And _
        strUser <> strUserC Then
        MsgBox "Für " & strUser & " liegt keine Ansicht vor.", _
            vbExclamation
        Exit Sub
    End If

    strPw = Application.InputBox( _
        strUser & ", geben Sie Ihr Passwort ein:", _
        "Passwortabfrage", "Passwort", Type:=2)
    If strPw = "Falsch" Or strUser = "" Then Exit Sub

    Select Case strUser
        Case strNormal
            CheckPassword strPwN
        Case strUserA
```

```
        CheckPassword strPwA
    Case strUserB
        CheckPassword strPwB
    Case strUserC
        CheckPassword strPwC
    End Select
End Sub
```

Folgende Prozedur `CheckPassword` überprüft den Wert der Variablen `strPw`, also die Passworteingabe aus der zweiten `InputBox`. Man könnte diese Prozedur auch in jedem der `Case`-Abschnitte schreiben. Dadurch würde die erste Prozedur aber zu unübersichtlich und vier mal den gleichen Code enthalten.

Die beim Aufruf der Prozedur übergebene Zeichenkette wird in der Variablen `strP` gespeichert. Ist das eingegebene Passwort gleich dem hinterlegten Passwort aus der Konstantendeklaration, wird in einer `With`-Anweisung zunächst der Blattschutz aufgehoben. Anschließend wird die benutzerdefinierte Ansicht aus der Eingabe der ersten `InputBox` angezeigt und das Tabellenblatt mit dem Passwort »Codebook« wieder geschützt.

```
'=====
' Auf CD      Buchdaten\Beispiele\02_Allgemein\
' Dateiname  02_10_CustomViewsWithPassword.xls
' Modul      mdl_01_MyCustomViews
'=====

Sub CheckPassword(strP As String)
    If strP <> strP Then
        MsgBox strUser & ", das Passwort ist falsch!", _
            vbExclamation
    Exit Sub
    Else
        With ActiveSheet
            .Unprotect "Codebook"
            ' benutzerdefinierte Ansichten nach Benutzernamen benannt
            ActiveWorkbook.CustomViews(strUser).Show
            .Protect "Codebook"
        End With
    End If
End Sub
```

66 Informationen des Betriebssystems auslesen

Die `Environ`-Funktion gibt über einen Zeichenfolgen- oder numerischen Ausdruck Informationen des Betriebssystems zurück. Abhängig vom Betriebssystem wird eine unterschiedliche Anzahl von Umgebungs-Variablen zur Verfügung gestellt. Die folgende Prozedur listet alle verfügbaren Umgebungs-Variablen Ihres Rechners in einem neuen Tabellenblatt auf. Der Blattname wird nach dem ausgelesenen Computernamen benannt.

```
'=====
' Auf CD      Buchdaten\Beispiele\02_Allgemein\
' Dateiname  02_11_OSInfo.xls
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

```
' Tabelle      Environ-Funktion
' Modul        mdl_01_OSInfo
' =====
```

```
Sub OSInfo()
    Dim intI As Integer

    ' Tabellenblatt einfügen
    Worksheets.Add after:=ActiveSheet

    ' Blatt löschen, wenn das Makro schon gelaufen ist
    For intI = 1 To Sheets.Count
        If Sheets(intI).Name = Environ("ComputerName") Then

            ' Warnhinweise abschalten
            Application.DisplayAlerts = False
            Sheets(intI).Delete
            ' Warnhinweise einschalten
            Application.DisplayAlerts = True

            ' Schleife verlassen
            Exit For
        End If
    Next intI

    ' Blattname nach Computernamen benennen
    ActiveSheet.Name = Environ("ComputerName")

    ' Tabellenüberschriften
    Range("A1") = "Nr."
    Range("B1") = "Zeichenfolge"
    Range("C1") = "Ausgabe"

    ' Überschriften formatieren
    With Range("A1:C1").Font
        .Bold = True
        .Size = 12
    End With

    intI = 1
    Do
        ' numerischer Ausdruck
        Cells(intI + 1, 1).Value = intI
        ' Zeichenfolge zu numerischem Ausdruck
        ' und Informationsausgabe
        Cells(intI + 1, 2).Value = Environ(intI)
        intI = intI + 1
    Loop Until Environ(intI) = ""

    ' Zeichenfolge von Ausgabeinformation trennen
    Range("B2:B100").TextToColumns _
        Destination:=Range("B2"), _
        DataType:=xlDelimited, _
```

```
Other:=True, OtherChar:=""

With Columns("A:C")
    ' Spaltenbreite anpassen
    .AutoFit
    ' Zellenausrichtung links
    .HorizontalAlignment = xlLeft
End With

End Sub
```

Die Information zu einer Umgebungsvariablen erhalten Sie im VBA-Code mit der Syntax:

```
Environ("Zeichenfolge")
```

Zum Beispiel liefert `Environ("OS")` den Namen des Betriebssystems und `Environ("Username")` den unter Windows angemeldeten Benutzernamen.

67 Zwei Benutzernamen auslesen

Neben dem unter Windows angemeldeten Benutzernamen verwaltet Excel einen eigenständigen Benutzernamen. Diesen können Sie in Excel unter EXTRAS | OPTIONEN | ALLGEMEIN | BENUTZERNAME festlegen, ändern oder löschen.

Im VBA-Code wird dieser mit der Syntax `Application.UserName` ermittelt. Ebenso können Sie den Namen des aktuellen Excel-Benutzers mit VBA ändern:

```
Application.UserName="Codebook-Leser"
```

Mit folgender Anweisung wird der unter Windows angemeldete Benutzername als Excel-Benutzername eingesetzt:

```
Application.UserName=Environ("UserName")
```

Achtung

Umgekehrt ist das Ändern des Windows-Benutzernamens unter VBA nicht möglich. Dieser ist Teil der Betriebssystem-Ebene und kann nur über die Verwaltung der Systemsteuerung geändert werden.

Folgendes Makro zeigt beide Benutzernamen in einem Meldungsfenster an.



Abbildung 50: Windows- und Excel-Benutzernamen anzeigen

Grundlagen

Allgemein

Datum
Zeit

tungen

Steuer
elemente

Befehl
leisten

Objekt

Diagramm

Ereignisse

User-
Form

Web/
Mail

Extern

Gemischtes

Spezial

```
'=====
' Auf CD      Buchdaten\Beispiele\02_Allgemein\
' Dateiname   02_11_OSInfo.xls
' Tabelle     ShowUserNames
' Modul       mdl_02_ShowUserNames
'=====
```

```
Sub ShowUserNames()
    MsgBox _
        "Windows-Benutzername:  " & Environ("UserName") & _
        vbNewLine & _
        "Excel-Benutzername:      " & Application.UserName, , _
        "Benutzernamen auslesen"
End Sub
```

68 Datum und Zeitformat anpassen

Microsoft Excel speichert Datumsangaben intern als fortlaufende Zahlen, um sie in Berechnungen verwenden zu können. Die Zahl 1 entspricht standardmäßig dem 1. Januar 1900, die Zahl 37622 ist gleich dem 01. Januar 2003, da er genau 37622 Tage nach dem 1. Januar 1900 liegt.

Zeitangaben werden intern als Bruchteile eines ganzen Tages gespeichert, da die Uhrzeit als Unter-
menge des Tages betrachtet wird.

1 Tag = 1

1 Stunde = 1/24

1 Minute = 1/1440 (24*60)

1 Sekunde = 1/86400 (24*60*60)

Über das Zellenformat kann eine Zahl in einem beliebigen Datum- oder Zeit-Format dargestellt werden. Öffnen Sie den Dialog ZELLEN FORMATIEREN über das Menü FORMAT | ZELLEN oder drücken Sie die Tastenkombination **[Strg]+[1]**, um die selektierten Zellen zu formatieren. In der Kategorie BENUTZERDEFINIERT können Sie eigene Zahlenformate definieren (siehe dazu auch Abbildung 51).

Die Formatierung der Tage, Monate und Jahre wird durch Angabe von Buchstaben dargestellt. Für die Anzeige der Tage verwenden Sie den Buchstaben *T*. Die Monatsanzeige geben Sie über den Buchstaben *M*, die Jahreszahl über den Buchstaben *J* an. Je nachdem wie oft Sie den Buchstaben eingeben, ändert sich die Anzeige in der Zelle. Am besten lässt sich dies an einem Beispiel veranschaulichen. Der Tag des Datums 01.01.2003 lässt sich auf vier verschiedene Arten anzeigen.

Tag	Anzeige	Monat	Anzeige	Jahr	Anzeige
T	1	M	1	J	3
TT	01	MM	01	JJ	03
TTT	Mi	MMM	Jan	JJJJ	2003
TTTT	Mittwoch	MMMM	Januar		

Tabelle 10: Datumsformatierungen

Je nach Kombination der Buchstaben erzeugen Sie individuelle Datumsformate, welche durch zusätzliche Textangaben ergänzt werden können. Der definierte Text muss dabei zwischen Anführungszeichen gesetzt werden.

Darstellung in Zelle	Kategorie	Zellenformat
37622	Standard	
01.01.2003	Datum	TT.MM.JJJJ

Tabelle 11: Datumsformate

Darstellung in Zelle	Kategorie	Zellenformat
So, 01.01.2003	Benutzerdefiniert	TTT, TT.MM.JJJJ
Mittwoch, der 01.01.2003	Benutzerdefiniert	TTTT, »der« TT.MM.JJJJ
München, den 1. Jan 2003	Benutzerdefiniert	»München, den« T. MMM JJJJ

Tabelle 11: Datumsformate (Forts.)

Die Uhrzeit können Sie standardmäßig in Stunden, Minuten und Sekunden anzeigen. Für die Formatierung werden wieder Buchstaben verwendet. Der Buchstabe *h* steht für Stunden, *m* für Minuten und *s* für Sekunden. Da der Buchstaben *M* sowohl für den Monat im Datumsformat als auch für die Minutenanzeige im Uhrzeitformat verwendet wird, werden die Schriftzeichen des Uhrzeitformats in Kleinbuchstaben angegeben. Am Beispiel der Uhrzeit 08:05:09 kann man die Auswirkung der einzelnen Buchstaben gut erkennen:

Format	Darstellung
h	8
hh	08
m	5
mm	05
s	9
ss	09

Tabelle 12: Uhrzeitformate

Auch hier gibt es wieder benutzerdefinierte Formatierungsmöglichkeiten.

Darstellung in Zelle	Kategorie	Zellenformat
0,5200833	Standard	
12:30:00	Uhrzeit	hh:mm:ss
12	Benutzerdefiniert	[hh]
750	Benutzerdefiniert	[mm]
45000	Benutzerdefiniert	[ss]
12 Uhr 30 Minuten	Benutzerdefiniert	hh »Uhr« mm »Minuten«

Tabelle 13: Zeitformate

Tipp

Die Anzeige und Berechnung von Hundertstelsekunden erreichen Sie in Excel über das benutzerdefinierte Zahlenformat hh:mm:ss,000.

Im Dialogfenster ZELLEN FORMATIEREN in den Kategorien DATUM und UHRZEIT, Listenfeld TYP können Sie ein vordefiniertes Format auswählen. Die Standardeinstellung der Systemsteuerung erkennen Sie an dem jeweils vorangestellten Sternchen (*).

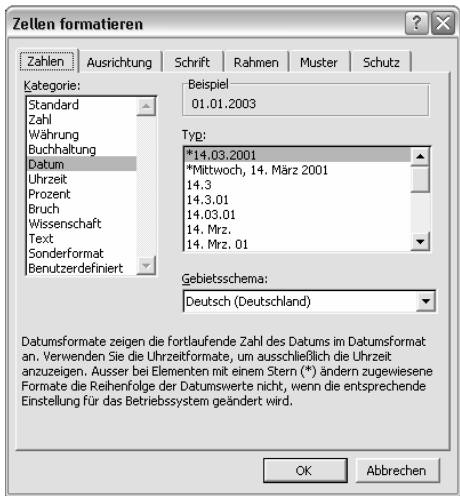


Abbildung 51: Standardeinstellung der Systemsteuerung

Das Zellenformat lässt sich sowohl über den Dialog ZELLEN FORMATIEREN laut Abbildung 51 als auch per VBA-Code bestimmen.

Im Tabellenblatt *DATEFORMAT* der Beispieldatei wird das Zellenformat eines Datums über ein Kombinationsfeld aus der STEUERELEMENT-TOOLBOX-Symbolleiste geändert. Hierzu verwenden wir das Change-Ereignis der ComboBox, welches bei jeder Änderung des Kombinationsfeldes ausgelöst wird. Der zu der Auswahl gehörende Formattyp steht im Tabellenblatt in dem benannten Bereich *CellFormat* und wird in der Prozedur ausgelesen.

Hinweis

Ausführliche Information zu Steuerelementen und deren Ereignisse finden Sie in der Kategorie »Steuerelemente«.

```
'=====
' Auf CD      Buchdaten\Beispiele\03_Datum_Zeit\
' Dateiname   03_01_DateAndTimeFormat.xls
' Tabelle     DateFormat
' Ereignis    Tabelle2 (DateFormat)
'=====

Private Sub ComboBox1_Change()
    Dim strFormat As String
    strFormat = Cells(ComboBox1.ListIndex + 2, [CellFormat].Column)
    Range("B6").NumberFormat = strFormat
End Sub
```

Analog dazu wird im Tabellenblatt *TIMEFORMAT* ein Zeitwert in unterschiedlichen Formatierungen in einer Zelle angezeigt.

Der Code ist fast identisch zum vorhergehenden, geändert hat sich lediglich der Name des Steuerelementes und der Bereichsname.

Grundlagen

Allgemein

Datum und Zeit

tungen

Steuerelemente

Befehle

Objekt

Diagramm

Ereignisse

User-Form

Web/Mail

Extern

Gemischtes

Spezial

```
strFormat = Cells(ComboBox2.ListIndex + 2, [CellFormatTime].Column)
```

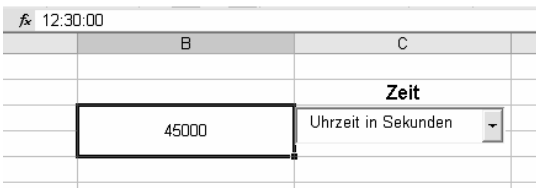


Abbildung 52: Zeiten formatieren

Wie Sie einen bereits vorhandenen Zellinhalt über das Zellformat im Dialog ZELLEN FORMATIEREN formatieren können, wissen Sie jetzt. Mit VBA ist es möglich, in eine Zelle einen formatierten Wert hineinzuschreiben, dies geschieht unabhängig vom vorhandenen Zellenformat.

```
Range("A1")=Format(Date,"MMMM") ' liefert den Monat der Systemzeit  
                                ' im Textformat
```

Hinweis

Wenn Sie einer im Format Text formatierten Zelle mit VBA ein Datum beliebigen Formates zuweisen, wird Excel dieses Datum korrekt anzeigen. Im Hintergrund bleibt diese Zelle allerdings weiterhin im Textformat bestehen, was zu Problemen bei weiteren Berechnungen führen kann.

Werden Datums- oder Zeitangaben in der Entwicklungsumgebung gemacht, dann wird ohne spezifische Formatanweisung das kurze Datums- oder Zeitformat aus der aktuellen Systemeinstellung zur Darstellung verwendet.

```
'=====
' Auf CD   Buchdaten\Beispiele\03_Datum_Zeit\
' Dateiname 03_01_DateAndTimeFormat.xls
' Tabelle   VBA_Format
' Modul     mdl_01_DateFormat
'=====
```

```
Sub DateToday ()
    MsgBox Date ' liefert das aktuelle Datum im Format »dd.mm.yyyy«
End Sub
```

Über die `Format`-Funktion weisen Sie einer Zelle oder Variablen neben einem Wert zugleich ein Format zu. Der zugewiesene Wert wird in dem angegebenen Formatausdruck dargestellt. Beachten Sie, dass die `Format`-Funktion einen Wert im Textformat zurückgibt. Das erkennen Sie daran, dass ein über die `Format`-Funktion gesetztes Datum in einer Zelle links ausgerichtet ist. Die Syntax dazu lautet:

```
Format(Ausdruck, Format)
```

Die Angabe des Formates wird in einem String dargestellt, welcher jeweils mit Anführungszeichen beginnt und endet.


```
Format(Date, "DD. MMMM YYYY") ' liefert 17. Januar 2002
```

Geben Sie zusätzlich zu dem Datumsformat einen Text in dem Formatausdruck mit an, dann muss dieser in Anführungszeichen gesetzt werden. Daraus resultieren die vielen Anführungszeichen in dem nachfolgenden Beispiel-Code.

```
'=====
' Auf CD      Buchdaten\Beispiele\03_Datum_Zeit\
' Dateiname  03_01_DateAndTimeFormat.xls
' Tabelle    VBA_Format
' Modul      mdl_01_DateFormat
'=====

Sub DateAndPlace()
    MsgBox Format(Date, ""München, den "DD. MMMM YYYY")
End Sub
```

69 Wochentag des Geburtstages ermitteln

Um herauszufinden, an welchem Wochentag Sie geboren sind, brauchen Sie nicht einmal VBA oder eine Formel für die Berechnung einzusetzen. Wie eingangs erwähnt, steht in einer Zelle lediglich eine Zahl, das Erscheinungsbild dieser Zahl bestimmen Sie durch die Formatierung der Zelle. Geben Sie in eine Zelle Ihr Geburtsdatum, zum Beispiel 03.12.1966, ein und formatieren Sie diese Zelle im benutzerdefinierten Zahlenformat TTTT. An dem Inhalt der Bearbeitungszeile sehen Sie, dass in der Zelle immer noch das eingegebene Datum 03.12.1966 steht. Die Zelle selbst stellt aber den Wochentag Samstag dar.

	B4		f _x =A4
	A	B	C
1			
2			
3			
4	03.12.1966	Samstag	
5			
6			

Abbildung 53: Wochentag eines Datums

70 Anzeigen des Systemdatums

In der VBA-Umgebung ermitteln Sie das aktuelle Datum mit der Date-Funktion. Hierbei wird, ebenso wie über die Tastenkombination **[Strg]+[.]** oder die Tabellenfunktionen =HEUTE(), das eingestellte Datum der Systemsteuerung abgerufen und im kurzen Datumsformat in die aktive Zelle geschrieben.

Die Time-Funktion gibt die aktuelle Uhrzeit zurück. Die Funktion Now liefert das aktuelle Datum und die Uhrzeit.

```
ActiveCell = Date ' ergibt: 17.01.2003

ActiveCell = Time ' ergibt: 22:15:54

ActiveCell = Now ' ergibt: 17.01.2003 22:15:55
```

Grundlagen

Allgemein

Datumszeit

Formeln

Steuerelemente

Befehle

Objekte

Diagramme

Ereignisse

UserForm

Web/Mail

Extern

Gemischtes

Spezial

Selbstverständlich sind diese Funktionen nicht nur darauf beschränkt, Werte in Zellen zu schreiben. Sie können auch einer Variablen das aktuelle Systemdatum oder die aktuelle Zeit zuweisen, um sie im weiteren Code zu verwenden:

```
Dim Datel As Date
Datel = Date
MsgBox Datel
```

Achtung

Durch Zuweisung eines Werts an die `Date`-Funktion können Sie das Systemdatum einstellen, was zu unerwünschten Nebeneffekten führen kann. Die gesamte Windows-Umgebung ist von dieser Einstellung betroffen. Das zugewiesene Datum wird als aktuelles Datum eingestellt und täglich um einen Tag erhöht.

```
Date = "24.12.2030"
```

Name	Gr...	Typ	Geändert am
Test_Systemdatum.xls	14 KB	Microsoft Excel Worksheet	24.12.2030 22:42

Abbildung 54: Datumsumstellung der Systemzeit

71 Schaltjahre bestimmen

Nach dem gregorianischen Kalender, der heute weltweit verbreitet ist, haben Gemeinjahre 365 Tage, Schaltjahre 366 Tage. Nach der genauen wahren Dauer des Umlaufs der Erde um die Sonne hat ein Jahr allerdings 365,25 Tage. Um diese Ungenauigkeit auszugleichen, wurde als Schalttag der 29. Februar eingeführt. Letztendlich wurden im Oktober 1582 von Papst Gregor XIII folgende Regeln aufgestellt. Diese definieren, welches Jahr ein Schaltjahr ist.

- ▶ Glatt durch 4 teilbare Jahre sind Schaltjahre
- ▶ Glatt durch 100 teilbare Jahre sind jedoch keine Schaltjahre
- ▶ Glatt durch 400 teilbare Jahre sind aber wieder Schaltjahre

So sind z.B. die Jahre 1800, 1900 und 2100 keine Schaltjahre. Die Jahre 2000, 2400 dagegen sind Schaltjahre. In den Schaltjahren hat ein Jahr also 366 Tage, da der 29. Februar enthalten ist. Damit dauert das gregorianische Jahr 365,2425 Tage.

Für Jahreszahlen ab 1901 liefert folgende Tabellenformel das Ergebnis WAHR für ein Schaltjahr und FALSCH für kein Schaltjahr. In der Zelle A1 wird eine Jahreszahl erwartet.

```
=TAG(DATUM(A1;2;29))=29
```

Bei der Schaltjahrberechnung macht Excel im Jahr 1900 allerdings einen Fehler. Hier zeigt Excel nach dem 28.02. den 29.02. an und nicht den 01.03.1900. Um für alle Jahre ein Schaltjahr korrekt zu definieren, verwenden Sie folgende Formel, die diesen Makel ausgleicht.

```
=N(TAG(("3/"&A1+(2000*(A1<5000)))-1)=29)=1
```

In der VBA-Umgebung werden alle Schaltjahre einwandfrei erkannt. Die benutzerdefinierte Funktion `IsLeapYear` gibt zu einer Jahreszahl den Wahrheitswert WAHR oder FALSCH für ein Schaltjahr zurück.

Hinweis

Ausführliche Informationen zu benutzerdefinierten Funktionen finden Sie in der Kategorie »Einstieg«.

```
'=====
' Auf CD      Buchdaten\Beispiele\03_Datum_Zeit\
' Dateiname   03_01a_LeapYear.xls
' Modul       mdl_01_LeapYear
'=====

Function IsLeapYear(intYear As Integer) As Boolean
    IsLeapYear = Day(DateSerial(intYear, 2, 29)) = 29
End Function
```

72 Dateizugriffe dokumentieren

Interessiert es Sie zum Beispiel, wann und von wem eine bestimmte Datei geöffnet wurde? Dann halten Sie beim Öffnen dieser Datei einfach das Datum, die Zeit und den Namen des Anwenders in einem separaten Tabellenblatt dieser Datei fest. Den Code schreiben Sie als Ereignisprozedur in das Codemodul DieseArbeitsmappe. Dadurch wird beim Öffnen der Datei die Workbook_Open-Prozedur automatisch durchlaufen.

Hinweis

Weitere Informationen zu Ereignisprozeduren lesen Sie in der Kategorie »Ereignisse«.

```
'=====
' Auf CD      Buchdaten\Beispiele\03_Datum_Zeit\
' Dateiname   03_01_DateAndTimeFormat.xls
' Tabelle     Logbook
' Ereignis    DieseArbeitsmappe
'=====

Private Sub Workbook_Open()
    Dim ws      As Worksheet
    Dim LastRow As Long

    Set ws = Worksheets("Logbook")
    LastRow = ws.Cells(Rows.Count, 1).End(xlUp).Row + 1

    With ws
        .Cells(LastRow, 1) = Format(Now, "DDD, DD.MM.YY hh:mm")
        .Cells(LastRow, 2) = Application.UserName
    End With

    On Error Resume Next
    ThisWorkbook.Save
End Sub
```

Grundlagen

Allgemein

Datum und Zeit

Funktionen

Steuerungselemente

Befehle

Objekte

Diagramme

Ereignisse

UserForm

Web/Mail

Extern

Gemischtes

Spezial

Um dieses Blatt vor anderen Benutzern unsichtbar zu machen, können Sie das Tabellenblatt über den Menüpunkt **FORMAT | BLATT** aus- und einblenden. Im VBA-Code verwenden Sie dazu folgende Befehle:

```
Worksheets("Logbook ").Visible = xlSheetHidden      ' ausblenden
Worksheets("Logbook ").Visible = xlSheetVisible     ' einblenden
```

Im Eigenschaftsfenster des Tabellenblattes bzw. mit VBA ist es möglich, Tabellenblätter so auszu- blenden, dass diese nur im VBA-Editor oder via VBA-Code eingeblendet werden können. Klicken Sie dazu im Projekt-Explorer des VBA-Editors auf das Tabellenblatt und setzen Sie im dazugehörigen Eigenschaftsfenster die *Visible*-Eigenschaft des Tabellenblattes auf *xlSheetVeryHidden*.

```
Worksheets("Logbook ").Visible = xlSheetVeryHidden  ' verstecken
```

Die VBA-Befehle zum Ein- und Ausblenden des Tabellenblattes LOGBOOK haben wir im Tabellenblatt DATEFORMAT der Beispieldatei einer Umschaltfläche aus der STEUERELEMENT-TOOLBOX-Symbolleiste zugewiesen. Je nach Wert der Umschaltfläche wird das Tabellenblatt ein- oder ausgeblendet. Den Code dazu können Sie zum Beispiel in einer *If...Then...Else*-Anweisung einarbeiten:

```
If ToggleButton1.Value = True Then
    Worksheets("Logbook").Visible = xlSheetVeryHidden
Else
    Worksheets("Logbook").Visible = xlSheetVisible
End If
```

Anstelle der *If...Then...Else*-Anweisung können Sie bei diesem Beispiel ebenso die *IIF*-Funktion verwenden. Die *IIF*-Funktion wertet einen vorgegebenen Ausdruck als WAHR oder FALSCH aus und gibt den definierten Rückgabewert aus. Die Syntax der *IIF*-Funktion lautet: *Iif*(expr, truepart, falsepart).

Hinweis

Iif wertet immer sowohl den Teil *truepart* als auch den Teil *falsepart* aus, auch dann, wenn nur einer von beiden Teilen zurückgegeben wird. Aus diesem Grund kann es zu unerwünschten Nebeneffekten kommen. Wenn z.B. die Auswertung von *falsepart* zu einem Fehler aufgrund einer Division durch Null führt, tritt ein Fehler auch dann auf, wenn *expr* den Wert *True* hat.

Wir wollen also der *Visible*-Eigenschaft des Tabellenblattes einen Wert zuweisen:

```
Worksheets("Logbook").Visible = ..._ ' Wahrheitswert
```

Diesen Wert ermitteln wir über die *IIF*-Funktion:

```
IIf(ToggleButton1.Value = True, xlSheetVeryHidden, xlSheetVisible)
```

Ist der Ausdruck *ToggleButton1.Value = True* WAHR, dann gibt die *IIF*-Funktion den hinterlegten Wahrheitswert *xlSheetVeryHidden* zurück und das Tabellenblatt LOGBOOK wird versteckt. Das entspricht der Schreibweise:

```
Worksheets("Logbook").Visible = xlSheetVeryHidden
```

Ist der Ausdruck `ToggleButton1.Value = True` FALSCH, dann gibt die IIF-Funktion den Parameter `falsepart` zurück. In unserem Beispiel haben wir diesem Parameter den Wert `xlSheetVisible` übergeben, wodurch das Tabellenblatt LOGBOOK eingeblendet wird. Das würde folgender Schreibweise entsprechen:

```
Worksheets("Logbook ").Visible = xlSheetVisible
```

Die Beschriftung der Umschaltfläche über die `Caption`-Eigenschaft und die Zuweisung der `Visible`-Eigenschaft werden in je einer IIF-Funktion in einer Prozedur abgearbeitet.

```
'=====
' Auf CD      Buchdaten\Beispiele\03_Datum_Zeit\
' Dateiname  03_01_DateAndTimeFormat.xls
' Tabelle     DateFormat
' Ereignis    Tabelle2 (DateFormat)
'=====
```

```
Private Sub ToggleButton1_Click()
    Application.ScreenUpdating = False

    Me.ToggleButton1.Caption = _
        IIf(ToggleButton1.Value = True, _
            "Logbook einblenden", "Logbook ausblenden")

    Worksheets("Logbook").Visible = _
        IIf(ToggleButton1.Value = True, _
            xlSheetVeryHidden, xlSheetVisible)

    Application.ScreenUpdating = True
End Sub
```

73 Bestandsveränderungen dokumentieren

Nachfolgend zeigen wir Ihnen ein kleines Beispiel, welches nicht nur das Thema Datum/Zeit und die dazugehörige Formatierung betrifft, sondern zugleich das Ereignis `Worksheet_Change` ausnutzt und die sinnvolle Verwendung eines benannten Bereichs aufzeigt.

In der verwendeten Warenbestandstabelle wird zu verschiedenen Artikeln die entsprechende Stückzahl in einer Zelle eingetragen. Die Eingabe der Stückzahl erfolgt im Bereich D2:D18. Diesem Bereich wurde über `EINFÜGEN | NAME | DEFINIEREN` der Name `Number` zugewiesen.

Der Vorteil dieser Namensvergabe liegt klar auf der Hand. Angenommen, Sie wollen für die Zeitdokumentation nicht zwei Spalten »Datum und Uhrzeit« wie in unserem Beispiel verwenden, sondern halten lediglich das »Datum« oder »Datum und Zeit« in einer Spalte fest. Dann können Sie die überflüssige Spalte aus der Tabelle löschen. Der vorherige Bezug zwischen dem Namen `Number` und dem Bereich D2:D18 wird auf den neuen Bereich C2:C18 angeglichen.

Würden Sie sich im Code auf einen absoluten Bereich `Range("D2:D18")` festlegen, müssten Sie das Makro an die neue Tabellenstruktur anpassen und den Wirkungsbereich des Makros in `Range("C2:C18")` ändern. Mit der Verwendung eines benutzerdefinierten Namens halten Sie Ihre Tabellenstruktur variabel und vermeiden Fehlerquellen.

Grundlagen

Allgemein

Datum
Zeit

ungen

Steuer
elemente

Befehl
leisten

Objekt

Diagramm

Ereignisse

User-
Form

Web/
Mail

Extern

Gemischtes

Spezial



Abbildung 55: Benannter Bereich

Nachdem in dem Tabellenblatt `SHORTDATE` ein Zellwert geändert wurde, überprüft die `Intersect`-Methode im `Worksheet_Change`-Ereignis des Tabellenblattes, ob die Eingabezelle im Bereich `Number` erfolgt ist. Liegt keine Übereinstimmung der Schnittmenge zwischen Eingabezelle und dem definierten Bereich vor, wird die Prozedur mit der Anweisung `Exit Sub` beendet. Befindet sich die Eingabezelle `Target` im angegebenen Range, geht das Makro zur `Select Case`-Anweisung weiter. Hier wird überprüft, in welcher Zeile die Änderung gemacht wurde, damit das jeweilige Format für die Eingabezelle aus der Zelle ausgelesen wird.

In unserer Beispieltabelle werden drei verschiedene Formate zugewiesen. Sie sehen, dass das Format `Short Date` das gleiche Ergebnis zeigt wie die benutzerdefinierte Datumsformatierung `DD.MM.YYYY`, da `Short Date` das kurze Datumsformat aus den Einstellungen der Systemsteuerung verwendet.

```
' =====
' Auf CD      Buchdaten\Beispiele\03_Datum_Zeit\
' Dateiname  03_01_DateAndTimeFormat.xls
' Tabelle     shortDate
' Ereignis   Tabelle4(shortDate)
' =====
```

```
Private Sub Worksheet_Change(ByVal Target As Range)
    If Application.Intersect(Target, [Number]) Is Nothing _
        Then Exit Sub
```

```
Application.EnableEvents = False
On Error Resume Next
```

```
Select Case (Target.Row)
    Case Is > 13
        Cells(Target.Row, Range("Date1").Column) _
            = Format(Date, "DDDD DD.MM.YYYY")
        Cells(Target.Row, Range("Time1").Column) _
            = Format(Time, "hh:mm:ss")
    Case Is > 7
        Cells(Target.Row, Range("Date1").Column) _
            = Format(Date, "DD.MM.YYYY")
        Cells(Target.Row, Range("Time1").Column) _
```

```
        = Format(Time, "hh:mm")
    Case Is > 1
        Cells(Target.Row, Range("Date1").Column) _
            = Format(Date, "Short Date")
        Cells(Target.Row, Range("Time1").Column) _
            = Format(Time, "Short Time")
    End Select

    Application.EnableEvents = True
End Sub
```

Nachfolgend sehen Sie eine Auflistung der in Excel zur Verfügung stehenden benannten Datums- und Zeitformate und deren Bedeutung.

Format	Beschreibung
General Date	Zeigt ein Datum und/oder eine Zeit an. Bei reellen Zahlen werden Datum und Uhrzeit angezeigt (zum Beispiel 4.3.93 05:34). Werden keine Nachkommastellen angegeben, so wird nur ein Datum (zum Beispiel 4.3.93) angezeigt. Enthält der Wert ausschließlich Nachkommastellen, so wird nur eine Uhrzeit ausgegeben (zum Beispiel 05:34). Die Anzeige von Datum und Zeit wird durch die Systemeinstellungen festgelegt.
Long Date	Zeigt ein Datum im langen Datumsformat entsprechend den Systemeinstellungen an.
Medium Date	Zeigt ein Datum im mittleren Datumsformat an, das von der Sprachversion der Host-Anwendung bestimmt wird.
Short Date	Zeigt ein Datum im kurzen Datumsformat entsprechend den Systemeinstellungen an.
Long Time	Zeigt eine Zeit entsprechend der Einstellung für das lange Zeitformat an, einschließlich Stunden, Minuten und Sekunden.
Medium Time	Zeigt eine Zeit im 12-Stunden-Format mit Stunden, Minuten und einer AM/PM-Kennung an.
Short Time	Zeigt eine Zeit im 24-Stunden-Format an (zum Beispiel 17:45).

Tabelle 14: Benannte Datums- und Zeitformate

74 Vereinfachte Datumseingabe

Die schnellste Art, das aktuelle Datum in der selektierten Zelle auszugeben, ist über die Tastenkombination **[Strg] + [.]**. Um die aktuelle Uhrzeit einzufügen, drücken Sie **[Strg] + [:]**.

Für das Datum gilt in Deutschland der Punkt als Trennzeichen zwischen Tag, Monat und Jahr. Die Datumseingabe können Sie sich vereinfachen, indem Sie statt des Punktes das Minuszeichen auf der Nummerntastatur eingeben. Dadurch ersparen Sie sich den Wechsel zwischen der Nummer- und Schreibtaatur.

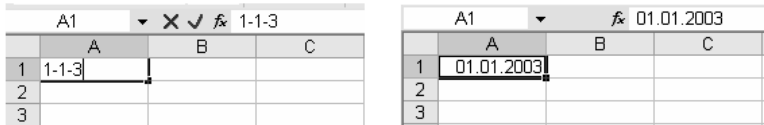


Abbildung 56: Vereinfachte Datumseingabe

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

Ein Datum des aktuellen Jahres können Sie noch schneller eingeben. Tippen Sie auf der Nummern-tastatur zuerst die Tageszahl gefolgt von dem Divisionszeichen und dann die Monatszahl. Für den 1. Januar schreiben Sie **1 / 1**. Nachdem Sie die Eingabezelle verlassen haben, generiert Excel daraus den ersten Januar des derzeitigen Jahres.

A1			A1		
X ✓ ✕ ✎ 1/1			✎ 01.01.2003		
A	B	C	A	B	C
1	1/1		1	01.01.2003	
2			2		
3			3		

Abbildung 57: Schnelles Datum des aktuellen Jahres

75 Uhrzeiten mit Autokorrektur

Oft muss man in einer Tabelle sehr viele Uhrzeiten eintippen, zum Beispiel in einer Arbeitszeit-tabelle. Damit Excel die Eingabe als Uhrzeit erkennt, ist es erforderlich, die Stunden durch einen Doppelpunkt von den Minuten zu trennen. Für die Uhrzeit 12:49 geben Sie über den Nummern-block der Tastatur die Zahlen 1 und 2 ein, müssen dann auf die Schreibtastatur wechseln, um mit **⇧ + .** einen **:** zu erzeugen, und wieder zurück auf den Nummernblock, um die Minuten einzutippen.

Eine einfache Möglichkeit, sich diese Arbeit zu erleichtern, erhalten Sie durch die Autokorrektur. Wählen Sie im Menü **EXTRAS | AUTOKORREKTUR-OPTIONEN** das Register **AUTOKORREKTUR**. Geben Sie im Eingabefeld **ERSETZEN** zwei Pluszeichen **++** ein, im Eingabefeld **DURCH** einen Doppelpunkt **:**, klicken Sie auf **HINZUFÜGEN** und schließen Sie den **AUTOKORREKTUR**-Dialog wieder.

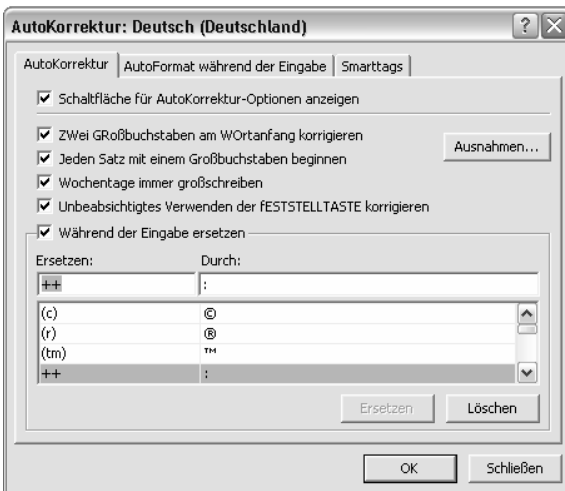


Abbildung 58: Autokorrektur Doppelpunkt

Sobald Sie in einer Zelle zwei Zahlen, gefolgt von zwei Plus-Zeichen und wieder zwei Zahlen hintereinander eingeben, ersetzt Excel diese zwei Plus-Zeichen durch einen Doppelpunkt und es entsteht eine für Excel zulässige Uhrzeit. Selbstverständlich funktioniert das auch mit einer anderen Kombination an Zeichen.

	A1		f _x	12++49
	A	B	C	
1	12++49			
2				
3				

	A1		f _x	12:49:00
	A	B	C	
1	12:49			
2				
3				

Abbildung 59: Schnelle Uhrzeiteingabe

76 Uhrzeiten ohne Doppelpunkt eingeben

Die Eingabe von Uhrzeiten über die Autokorrektur ist schon eine große Erleichterung, aber geht das nicht noch ein bisschen bequemer?

Viel einfacher wäre es doch, Sie tippen lediglich die vier Zahlen ein, welche eine Uhrzeit erfordert, und Excel setzt diese Eingabe in ein von Ihnen gewünschtes Format um. Selbstverständlich muss das Weiterrechnen mit diesem Zellinhalt auch dann noch gewährleistet sein.

Eingabe	Ergebnis
1	00:01
10	00:10
100	01:00
1000	10:00

Tabelle 15: Uhrzeitergebnisse durch Zahleneingaben

Nach Eingabe eines Wertes in eine Zelle wird das `Worksheet_Change`-Ereignis ausgelöst (Näheres zu Ereignis-Prozeduren erfahren Sie in der entsprechenden Kategorie). Dieses Ereignis nutzen Sie, um eine laufende Umwandlung der Eingaben in das Uhrzeitformat durchzuführen. Die Prozedur wird in den Codebereich des Tabellenblattes geschrieben, in welchem die Umwandlung in das Uhrzeitformat ausgeführt werden soll. Es ist erforderlich, den Zellbereich, welcher in das Uhrzeitformat umgewandelt werden soll, im Code zu definieren. Ansonsten würde der Code in jeder Zelle in der Tabelle die eingegebenen Zahlen in Uhrzeiten umwandeln. Das ist meistens nicht erwünscht. In unserem Beispiel werden die Uhrzeiten im Bereich `B2:C32` eingetragen. Mit der `Intersect`-Methode wird der Variablen `rngArea` die Schnittmenge des Eingabe- und Gültigkeitsbereichs übergeben.

```
Set rngArea = Application.Intersect(Target, Range("B2:C32"))
```

Befindet sich keine geänderte Zelle im Bereich `B2:C32`, wird die Prozedur mit der `Exit`-Anweisung vorzeitig beendet.

Um die Geschwindigkeit des Makros zu erhöhen und die einzelnen Schritte des Makros am Bildschirm unsichtbar zu machen, wird die Bildschirmaktualisierung über die `ScreenUpdating`-Eigenschaft deaktiviert. Würde durch den Code der Inhalt der Zelle verändert (dazu gehört auch das Rückgängigmachen der letzten Benutzeraktion), würde das `Worksheet_Change`-Ereignis erneut ausgelöst werden und die Prozedur in einer Endlosschleife enden. Um dies zu verhindern, werden alle Ereignisse mit der `EnableEvents`-Eigenschaft ausgeschaltet. Da sich diese beiden Befehle auf die `Application`-Eigenschaft beziehen, können sie in einer `With`-Anweisung zusammengefasst werden.

Grund-lagen

Allge-mein

Datu-Zeit

tungen

Steuer-elemen

Befehl-leisten

Objekt

Dia-gramm

Ereig-nisse

User-Form

Web/Mail

Extern

Ge-mischtes

Specia

Die Eingabe kann in eine einzelne Zelle oder in einen markierten Bereich erfolgen, indem Sie mehrere Zellen markieren, die Zahl 1249 eintippen und mit `[Strg]+[↵]` die Eingabe beenden. Alle Zellen in dem markierten Bereich erhalten so die Zahl 1249.

Hinweis

Mit `Target` sprechen Sie die Markierung in Ihrem Tabellenblatt an. Ob der markierte Bereich aus nur einer einzelnen Zelle, oder einem Bereich mit mehreren Zellen besteht, spielt hier keine Rolle.

Damit es bei einem Eingabebereich von mehr als einer Zelle nicht zu einer Fehlermeldung kommt, benötigt das Makro eine `For Each...Next`-Anweisung. In dieser Schleife werden mittels der Variablen `rngCell` alle Zellen des markierten Bereichs `Target` angesprochen. Anschließend wird geprüft, ob sich jede Zelle innerhalb der definierten Schnittmenge `rngArea` befindet. Wenn diese Prüfung nicht stattfindet, würde eine Umwandlung der Eingabe auch in dem nicht vorgesehenen Bereich erfolgen.

Liegt eine Zelle außerhalb des definierten Bereichs B2:C32, wird die letzte Benutzeraktion, also die Eingabe eines Wertes, über die `Undo`-Methode rückgängig gemacht. Nachdem die erste Zelle des Bereichs mit `Target(1, 1).Select` markiert wurde, wird das Makro über die `GoTo`-Anweisung zur Zeilenmarke `Errorhandler` verzweigt.

Befindet sich der Eingabebereich im vorgesehenen Bereich, wird in der nächsten `For Each...Next`-Anweisung überprüft, welche Eingabe gemacht wurde. Entspricht die Eingabe nicht einer für Excel umwandlungsfähigen Uhrzeit, wird die Prozedur vorzeitig beendet. Es würde sonst zu einer Fehlermeldung führen. Entfernen Sie zum Beispiel einen vorhandenen Zellinhalt oder verlassen Sie eine Zelle ohne Eingabe eines Wertes, würde die Zelle 00:00 anzeigen. Damit diese leere Zelle (»«) auch tatsächlich leer bleibt, darf das Makro nicht durchlaufen werden. Verwenden Sie für diese Abfrage die Syntax `If rngZelle = ""`.

Geben Sie Text in eine Zelle ein, dann würde Excel versuchen, diesen in ein Uhrzeitformat umzuwandeln, was zu einer Fehlermeldung führen würde. Kontrollieren Sie deshalb mit Hilfe der `IsNumeric`-Funktion, ob es sich bei der Eingabe um eine Zahl handelt. Wenn nicht, wird die Prozedur ebenfalls vorzeitig verlassen.

Die nächste Fehlerquelle liegt in der Anzahl der eingegebenen Zahlen. Der Code kann maximal vierstellige Zahlen in Uhrzeiten umwandeln. Überprüfen Sie deshalb mit der `Len`-Funktion, ob die eingegebene Zahlenlänge über vier liegt (zwei Zahlen für die Stunden, zwei für die Minuten). Wenn dies zutrifft, dann wird das Makro vorzeitig beendet, um eine Fehlermeldung zu verhindern. Eingaben von Zahlenwerten unter vier Zahlen werden fehlerfrei umgesetzt.

Diese drei Abfragen lassen sich in einer Zeile im VBA-Code darstellen:

```
If rngCell = "" Or Not IsNumeric(rngCell) Or Len(rngCell) > 4 _
Then GoTo ErrorHandler
```

Die Zahlen 1249 sind in der Variable `rngCell` enthalten und werden in Stunden und Minuten umgerechnet. Die Variable `intHour` erhält den Wert 12 für die Stunden, `intMinute` 49 für die Minuten. Die gefüllten Variablen werden an die Parameter der `TimeSerial`-Funktion übergeben und das Format der Zelle bestimmt.

Falls ein größerer Bereich mit mehreren Zahlen durch `Copy` und `Paste` eingefügt wird, kann es sich um verschiedene Werte handeln. Aus diesem Grund wird jede Zelle des markierten Bereichs einzeln berechnet und in die jeweilige Zelle geschrieben.

Hinweis

Vor Beendigung der Prozedur muss die Bildschirmaktualisierung wieder eingeschaltet werden.

Vergessen Sie außerdem nicht, alle Ereignis-Prozeduren wieder zu aktivieren, sonst würde bei der nächsten Eingabe in eine Zelle dieses Makro nicht mehr aufgerufen.

Der gesamte Code sieht folgendermaßen aus:

```
'=====
' Auf CD      Buchdaten\Beispiele\03_Datum_Zeit\
' Dateiname  03_02_TimeWithoutDoublepoint.xls
' Ereignis    Tabelle1(Calendar)
'=====

Private Sub Worksheet_Change(ByVal Target As Range)
    Dim rngCell As Range
    Dim rngArea As Range
    Dim intHour As Integer
    Dim intMinute As Integer

    ' Schnittmenge des Eingabe- und Gültigkeitsbereichs
    Set rngArea = _
        Application.Intersect(Target, Range("B2:C32"))

    ' keine Eingabezeile ist im Gültigkeitsbereich
    If rngArea Is Nothing Then Exit Sub

    ' Bildschirmaktualisierung und Ereignisse ausschalten
    With Application
        .ScreenUpdating = False
        .EnableEvents = False
    End With

    ' Eingaben innerhalb >>und<< außerhalb des Gültigkeitsbereichs
    ' werden rückgängig gemacht
    For Each rngCell In Target
        If Intersect(rngCell, rngArea) Is Nothing Then
            Application.Undo
            Target(1, 1).Select
            GoTo ErrorHandler
        End If
    Next

    ' Prüfung der Eingabewerte
    For Each rngCell In Target
        If rngCell = "" Or Not IsNumeric(rngCell) _
            Or Len(rngCell) > 4 Then
            GoTo ErrorHandler
        End If

        ' Berechnung der Uhrzeit
        intHour = rngCell \ 100
```

Grundlagen
Allgemein
Datum und Zeit
Formeln
Diagramme
Ereignisse
User-Formen
Web/Mail
Extern
Gemischtes
Spezial

```

    intMinute = rngCell - intHour * 100

    ' Uhrzeit in Zelle schreiben
    rngCell = TimeSerial(intHour, intMinute, 0)
    rngCell.NumberFormat = "[hh]:mm"
Next rngCell

Errorhandler:
' Bildschirmaktualisierung und Ereignisse einschalten
With Application
    .EnableEvents = True
    .ScreenUpdating = True
End With
End Sub

```

77 Aktuelles Datum finden

Nachfolgend zeigen wir Ihnen, wie Excel mit der Find-Methode das aktuelle Datum in einer Liste von Daten findet. In der Beispieltabelle wird der aktuelle Monat aufgelistet. Wird das heutige Datum nicht gefunden, erhalten Sie eine Fehlermeldung.

```

' =====
' Auf CD      Buchdaten\Beispiele\03_Datum_Zeit\
' Dateiname   03_02_TimeWithoutDoublepoint.xls
' Modul       mdl_01_FindDate
' =====

Sub FindCurrentDate()
    Dim rngArea As Range
    Set rngArea = _
        Worksheets("Calendar").Range("A2:A32").Find(What:=Date, _
            LookIn:=xlValues)

    If Not rngArea Is Nothing Then
        rngArea.Offset(0, 1).Select
    Else
        MsgBox "Das Datum " & Date & " wurde nicht gefunden"
    End If
End Sub

```

78 Beliebiges Datum finden

Sie können aber auch einen beliebigen Tag des aktuellen Monats ausfindig machen. Wie in unserem vorherigen Beispiel soll in der gefundenen Datums-Zeile ebenfalls die Nebenzelle selektiert werden, um eine Eingabe zu machen. In unserem Beispiel wird für die Eingabe der Tageszahl ein Eingabedialog `InputBox` verwendet. Diese wird über eine `Do...Loop`-Anweisung so lange aufgerufen, bis eine gültige Tageszahl des aktuellen Monats eingegeben wurde bzw. Sie die Schaltfläche `ABBRECHEN` der `InputBox` drücken.

Das Makro benötigt zur Zusammensetzung eines Datums zwingend die Eingabe einer Zahl für den Tageswert. Diese Bedingung wird über den letzten Parameter `Type` der `InputBox`-Methode

vorgegeben. Den Wert der `InputBox` übergeben wir zur weiteren Verwendung im Code an die Variable `AnyDay`.

Wie lässt sich nun überprüfen, ob der gewählte Tag einen gültigen Tagwert im aktuellen Monat darstellt? Da die Anzahl der Tage eines Monats zwischen 28 und 31 Tagen variiert, muss ermittelt werden, wie viele Tage der aktuelle Monat hat. Diese Berechnung führen wir über die `DateSerial`-Funktion aus. Von dem nachfolgenden Monat wird ein Tag abgezogen, so dass die `DateSerial`-Funktion das Datum des letzten Tages des derzeitigen Monats ausgibt. Im Januar 2003 liefert die Funktion den 31.01.2003.

Die `DateSerial`-Funktion integrieren wir in die `Day`-Funktion und ermitteln mit dem letzten Tag des Monats zugleich die Anzahl der Tage im aktuellen Monat.

```
Day(DateSerial(Year(Date), Month(Date) + 1, -1)) ' liefert 31
```

Nun lässt sich bequem feststellen, ob die in die `InputBox` eingegebene Tageszahl größer als die Anzahl der Tage des Monats ist. Wenn der Wert von `AnyDay` größer ist, wird dieses Datum nicht in der Tabelle zu finden sein. Wir setzen deshalb den Wert `AnyDay` auf 0, damit die `Do...Loop`-Anweisung nochmals durchlaufen wird. Es erfolgt eine erneute Abfrage des Tages über die `InputBox`.

Wurde ein gültiger Tageswert in die `InputBox` eingegeben, wird im Bereich der Daten A2:A32 mittels der `Find`-Methode der Zellbezug des gesuchten Datums an die Variable `rngDate` übergeben. Der Wert der Variablen `AnyDay` ist nun >0 , so dass die `Do...Loop`-Schleife verlassen und die linke Nebenzelle der gefundenen Datumszelle markiert wird.

```
'=====
' Auf CD      Buchdaten\Beispiele\03_Datum_Zeit\
' Dateiname   03_02_TimeWithoutDoublepoint.xls
' Modul       mdl_01_FindDate
'=====
```

```
Sub FindAnyDate()
    Dim rngDate As Range
    Dim AnyDay As Integer

    Do
        AnyDay = Application.InputBox _
            ("Geben Sie eine Tageszahl des aktuellen Monats ein.", _
            "Searching Date", 1, , , , 1)

        ' Abbrechen wurde geklickt
        If AnyDay = False Then Exit Sub

        If AnyDay > _
            Day(DateSerial(Year(Date), Month(Date) + 1, -1)) Then

            MsgBox "Ihre Eingabe ist kein gültiger Tag des Monats " & _
                Format(Date, "mmm"), _
                vbCritical, "Input Error"

            AnyDay = 0
        Else
            Set rngDate = _
```

Grund-
lagenAllge-
meinDatum
Zeit

tungen

Steuer
elemenBefehl
leisten

Objekt

Dia-
grammEreig-
nisseUser-
FormWeb/
Mail

Extern

Ge-
misch
tes

Specia

```

Range("A2:A32").Find _
(What:=DateSerial(Year(Date), Month(Date), AnyDay), _
LookIn:=xlValues)
End If
Loop Until AnyDay > 0

rngDate.Offset(0, 1).Select
End Sub

```

79 Arbeitsstunden in Meldungsfenster anzeigen

Aus den vorherigen Beispielen wissen Sie nun, wie über die Find-Methode die Adresse einer Datumszelle gefunden wird. Im folgenden Beispiel sollen anhand der Beispieldatei die geleisteten Arbeitsstunden vom 02.03.2003 angezeigt werden. Im Bereich A2:A32 stehen alle Datumswerte des aktuellen Monats. Mit der DateSerial-Funktion wird der zweite Tag des derzeitigen Monats und Jahres ermittelt. Dieses Datum wird über die Find-Methode im Datenbereich gesucht. Mittels der Offset-Eigenschaft Offset(0, 3).Value wird der Wert, welcher sich null Zeilen und drei Spalten gegenüber dem gefundenen Bereich versetzt befindet, zurückgegeben.

In der gesuchten Zelle D3 wird die per Formel errechnete Arbeitszeit über das Uhrzeitformat hh:mm als 09:00 angezeigt. Excel speichert diesen Wert intern jedoch als 0,375. In der Prozedur wird mit der Value-Eigenschaft lediglich der Wert ohne jegliche Formatierungsangaben angefordert. Aus diesem Grund erfolgt in dem Meldungsfenster die Ausgabe der Stunden als Dezimalzahl.

	A2					
	A	B	C	D	E	F
1	Date	Begin	End	Time		
2	01.03.2003					
3	02.03.2003	08:00	17:00	09:00		
4	03.03.2003	07:10	18:30	11:20		
5	04.03.2003	06:00	15:00	09:00		
6	05.03.2003					
7	06.03.2003					
8	07.03.2003					
9	08.03.2003					



Abbildung 60: Uhrzeitausgabe dezimal

```

' =====
' Auf CD      Buchdaten\Beispiele\03_Datum_Zeit\
' Dateiname   03_02_TimeWithoutDoublepoint.xls
' Modul       mdl_03_FindTime
' =====

```

```

Sub FindDecimalTime()
    MsgBox _
        "Am " & Range("A3").Value & " haben Sie " & _
        vbCr & _
        Range("A2:A32").Find _
        (What:=DateSerial(Year(Date), Month(Date), 2), _
        LookIn:=xlValues).Offset(0, 3).Value & _
        " Stunden gearbeitet"
End Sub

```

In der Zelle wird der Wert über das Uhrzeitformat hh:mm als 09:00 angezeigt. Geben Sie in der Format-Funktion das Format hh:mm an. Dadurch wird der gefundene Wert im Meldungsfenster im Uhrzeitformat angezeigt.

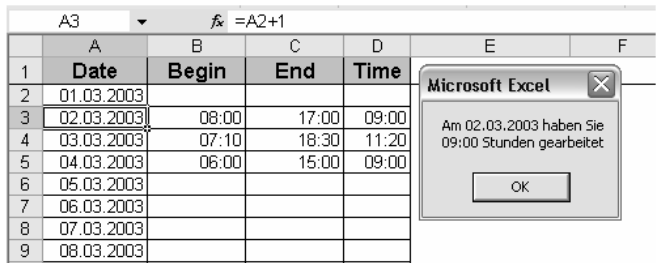


Abbildung 61: Uhrzeitausgabe im Stundenformat

```
'=====
' Auf CD      Buchdaten\Beispiele\03_Datum_Zeit\
' Dateiname  03_02_TimeWithoutDoublepoint.xls
' Modul      mdl_03_FindTime
'=====

Sub FindTime()
    MsgBox _
        "Am " & Range("A3").Value & " haben Sie " & _
        vbCr & _
        Format(Range("A2:A32").Find _
            (What:=DateSerial(Year(Date), Month(Date), 2), _
            LookIn:=xlValues).Offset(0, 3).Value, "hh:mm") & _
        " Stunden gearbeitet"
End Sub
```

80 Arbeitszeiten summieren

Die Arbeitszeit eines Tages lässt sich noch einfach in der Tabelle nachlesen. Für den Zeitraum vom 02.03. bis 04.03.2003 benötigen Sie die Summenformel =SUMME(D3:D5) und formatieren die Ausgabezelle im Uhrzeitformat. Als Ergebnis erhalten Sie 29:20.

Um den zu summierenden Zeitraum variabel zu halten, verwenden Sie nachfolgende Prozedur. Geben Sie den Beginn und das Ende des zu berechnenden Zeitraums jeweils in einer InputBox als Tageszahl ein. Excel gibt die gewünschte Information anschließend in einem Meldungsfenster aus.

Nun, wie kommen wir dahin?

Über die zwei Eingabedialoge erfahren Sie im Code die gesuchten Zelladressen und setzen daraus einen Range zusammen. Die Variablen FirstDay, LastDay und myRange werden als Range deklariert und enthalten die gefundenen Zelladressen der eingegebenen Daten.

```
Set myRange = Worksheets(1).Range(FirstDay, LastDay)
```

Analog zu der Tabellenblattfunktion Summe verwenden Sie in VBA das WorksheetFunction-Objekt. Es wird als Container für Arbeitsblattfunktionen von Microsoft Excel verwendet, die aus Visual

Grundlagen

Allgemein

Datum und Zeit

Formeln

Steuerelemente

Befehlsliste

Objekte

Diagramme

Ereignisse

User-Formen

Web/Mail

Extern

Gemischtes

Spezial

Basic aufgerufen werden können. In VBA müssen die englischen Funktionsnamen verwendet werden. Weisen Sie deshalb diesem Objekt die Eigenschaft `SUM` zu. Die Variable `TotalSum` erhält das Ergebnis aus der Summenberechnung.

```
TotalSum = Application.WorksheetFunction.Sum(myRange)
```

Bei der Berechnung der Uhrzeiten erhalten Sie in unserem Beispiel einen Wert über 24 Stunden. Um Zeiten über 24 Stunden anzuzeigen, wird dafür im Tabellenblatt das Format `[hh]:mm` angewandt. Leider kennt die `Format`-Funktion in VBA die `[]`-Klammern nicht als Formatierung an. Dementsprechend kann die `Format`-Funktion nicht zur Anzeige der errechneten Stunden verwendet werden. Um dennoch eine Ausgabe über die `MsgBox` im Uhrzeitformat zu erzielen, setzen Sie die Uhrzeit in der Prozedur selbst zusammen und integrieren diese im Meldungsfenster. Im Makro wird dafür die Variable `strTime` reserviert.

```
'=====
' Auf CD      Buchdaten\Beispiele\03_Datum_Zeit\
' Dateiname   03_02_TimeWithoutDoublepoint.xls
' Modul       mdl_02_WorkingHours
'=====
```

```
Sub WorkingHours()
    Dim Date1      As Integer
    Dim Date2      As Integer
    Dim FirstDay   As Range
    Dim LastDay    As Range
    Dim myRange    As Range
    Dim TotalSum   As Double
    Dim strTime    As String

    On Error GoTo ErrorHandler

    Date1 = Application.InputBox _
        ("Ab welchem Tag wollen Sie die Stunden summieren?" & _
        vbNewLine & vbNewLine & _
        "Geben Sie eine Zahl für den ersten Tag ein." _
        , "Begin Day", 1, , , , 1)
    If Date1 = False Then Exit Sub

    Date2 = Application.InputBox _
        ("Bis zu welchem Tag wollen Sie die Stunden summieren?" & _
        vbNewLine & vbNewLine & _
        "Geben Sie eine Zahl für den letzten Tag ein." _
        , "End Day", 1, , , , 1)
    If Date2 = False Then Exit Sub

    Set FirstDay = Range("A2:A32").Find _
        (What:=DateSerial(Year(Date), Month(Date), Date1), _
        LookIn:=xlValues).Offset(0, 3)

    Set LastDay = Range("A2:A32").Find _
        (What:=DateSerial(Year(Date), Month(Date), Date2), _
```

```
LookIn:=xlValues).Offset(0, 3)

Set myRange = Range(FirstDay, LastDay)

TotalSum = Application.WorksheetFunction.Sum(myRange)

strTime = Format(Int(TotalSum * 24), "00") & ":" & _
    Format(Int(((TotalSum * 24) - _
        Int(TotalSum * 24)) * 60), "00")

MsgBox "Im Zeitraum vom " & Date1 & " bis " & Date2 & ". " _
    & Format(Date, "mmm") & vbCrLf _
    & "ergeben sich " _
    & strTime & " Stunden" _
    , vbInformation + vbOKOnly _
    , "WorkingHours"

Exit Sub

ErrorHandler:
MsgBox "Im Zeitraum vom " & Date1 & " bis " & Date2 _
    & vbCrLf _
    & "konnte keine Stundensumme gebildet werden" _
    , vbCritical + vbOKOnly _
    , "Fehler"

End Sub
```

Grundlagen

Allgemein

Datum
Zeit

tungen

Steuer
elemente

Befehl
leisten

Objekt

Diagramm

Ereignisse

User-
Form

Web/
Mail

Extern

Gemischtes

Spezial

81 Kalenderwoche nach DIN 1355 berechnen

Die in Excel integrierte Funktion `KALENDERWOCHE` aus dem Add-In *Analyse-Funktionen.xla* liefert eine Zahl, die angibt, in welche Woche des dazugehörigen Jahres das angegebene Datum fällt.

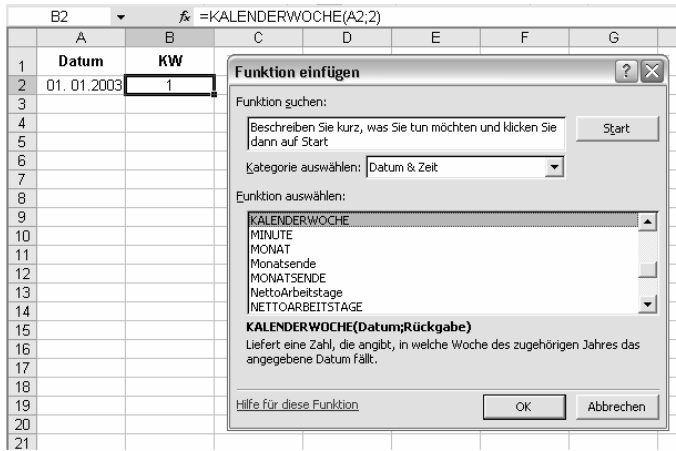


Abbildung 62: Integrierte Funktion Kalenderwoche

Einige Excel-Spezialisten bemühen sich seit Jahren darum, komplizierte Formeln und benutzerdefinierte Funktionen zu Berechnung der Kalenderwoche zu erstellen. »Warum?«, werden Sie fragen, wenn diese Funktion doch bereits mit Excel mitgeliefert wird.

Die Funktion `KALENDERWOCHE()` rechnet falsch, oder sagen wir besser, nach amerikanischem Standard. Der 01.01.2000 z.B. ist laut DIN in der 52. Kalenderwoche des Jahres 1999. In den USA ist dieses Datum in der ersten Kalenderwoche des Jahres 2000. Erst 1976 wurde der Wochenbeginn in Deutschland auf »Montag« festgelegt. Die erste Woche des Jahres ist definiert als die Woche, in die mindestens 4 Tage des neuen Jahres fallen. Beides nach DIN 1355 aus dem Jahr 1974. Diese Definition entspricht der internationalen Norm ISO 8601 (1988); übernommen von der EU als EN 28601 (1992) und in Deutschland als DIN EN 28601 (1993) umgesetzt.

Anders gesagt; die Woche, deren Donnerstag als erster im neuen Jahr liegt, ist KW 1.

Für einzelne Daten innerhalb von Jahrzehnten mag man mit falschen Kalenderwochen-Angaben leben können. Da die in Excel integrierte Funktion `KALENDERWOCHE()` nachweislich für die kompletten Jahre 2005, 2006, 2010 – 2012 und weitere Jahre falsche Werte liefert, ist eine benutzerdefinierte Formel oder Funktion unerlässlich.

Die folgende Tabellen-Funktion berechnet anhand eines Datums in Zelle A1 die korrekte deutsche Kalenderwoche nach der aktuellen DIN 1355 Norm:

```
=KÜRZEN((A1-DATUM(JAHR(A1-REST(A1-2;7)+3);1;REST(A1-2;7)-9)))/7)
```

In der ersten Auflage des »Excel-VBA Codebook« haben wir Ihnen eine benutzerdefinierte Funktion zur Berechnung der Kalenderwoche aufgezeigt. Bedauerlicherweise hat sich nach weiteren Prüfungen herausgestellt, dass diese Funktion innerhalb der nächsten 100 Jahre an 13 Tagen einen falschen Wert liefert.

In der Beispieldatei wurden verschiedene Möglichkeiten zur Berechnung der Kalenderwoche gegenübergestellt. Ausgangsdatum sind die Daten in Spalte A. Hier wurden nur die gefilterten falschen Daten, der Funktion `DINKw` aus der ersten Auflage, aufgenommen. Die interne Excel-Funktion, nach amerikanischem Standard, liefert weitaus mehr Fehlberechnungen.

Zur Kontrolle wurden das Datum und die Kalenderwoche des ersten Donnerstags des angegebenen Jahres mit in die Tabelle aufgenommen.

	A	B Funktion KALENDERWOCHE	C Formel	D DINKw	E WeekNumberDIN	F erster Donnerstag	G KW erster Do	H B=C	I D=E
2	Mo. 29.12.2003	53	1	53	1	Do. 02.01.2003	1	FALSCH	FALSCH
3	Mo. 31.12.2007	53	1	53	1	Do. 04.01.2007	1	FALSCH	FALSCH
4	Mo. 30.12.2019	53	1	53	1	Do. 03.01.2019	1	FALSCH	FALSCH
5	Mo. 29.12.2031	53	1	53	1	Do. 02.01.2031	1	FALSCH	FALSCH
6	Mo. 31.12.2035	53	1	53	1	Do. 04.01.2035	1	FALSCH	FALSCH
7	Mo. 30.12.2047	53	1	53	1	Do. 03.01.2047	1	FALSCH	FALSCH
8	Mo. 29.12.2059	53	1	53	1	Do. 02.01.2059	1	FALSCH	FALSCH
9	Mo. 31.12.2063	53	1	53	1	Do. 04.01.2063	1	FALSCH	FALSCH
10	Mo. 30.12.2075	53	1	53	1	Do. 03.01.2075	1	FALSCH	FALSCH
11	Mo. 29.12.2087	53	1	53	1	Do. 02.01.2087	1	FALSCH	FALSCH
12	Mo. 31.12.2091	53	1	53	1	Do. 04.01.2091	1	FALSCH	FALSCH
13	So. 02.01.2101	2	52	53	52	Do. 06.01.2101	1	FALSCH	FALSCH
14	Mo. 31.12.2103	53	1	53	1	Do. 04.01.2103	1	FALSCH	FALSCH

Abbildung 63: Berechnung Kalenderwoche

Die korrekten Werte in Spalte C wurden mit obiger Formel errechnet.

In Spalte E wird über die benutzerdefinierte Funktion `WeekNumberDIN()` die exakte Kalenderwoche des Datums in Spalte A berechnet. Die Formel in E2 lautet `=WeekNumberDIN(A2)`.

Folgende benutzerdefinierte Funktion muss dem Tabellenblatt für die Berechnung in einem Standardmodul zur Verfügung gestellt werden.

Hinweis

Ausführliche Informationen zu benutzerdefinierten Funktionen finden Sie in der Kategorie »Einstieg«.

```
'=====
' Auf CD      Buchdaten\Beispiele\03_Datum_Zeit\
' Dateiname  03_03_WeeknumberDIN.xls
' Modul      mdl_01_WeekNumberDIN
'=====

Function WeekNumberDIN(dDate As Date) As Integer
    Dim dDate1 As Date

    dDate1 = 4 + dDate - Weekday(dDate, 2)

    WeekNumberDIN = (dDate1 - DateSerial(Year(dDate1), 1, -6)) \ 7
End Function
```

82 Anzahl Kalenderwochen bestimmen

Nach der in Deutschland gültigen DIN-1355-Norm für Wochenzählungen kann ein Jahr 52 oder 53 Wochen enthalten. Ein Jahr hat 53 Wochen, wenn der 01.01. und/oder 31.12. des Jahres ein Donnerstag ist.

In der benutzerdefinierten Funktion CountGermanWeeks wird eine Jahreszahl an die Variable intYear übergeben. Über die DatePart-Funktion werden die Wochentage des 01.01. und 31.12. des Jahres intYear ermittelt. Die Funktion DatePart gibt je nach Wochentag eine Zahl von 1 bis 7 zurück. Die Ergebnisse werden in den Variablen bytDay1 und bytDay2 gespeichert. In der If...Then...Else-Anweisung erfolgt anschließend die Überprüfung des Wochentags und Zuweisung der Anzahl Kalenderwochen an die CountGermanWeeks-Funktion.

```
'=====
' Auf CD      Buchdaten\Beispiele\03_Datum_Zeit\
' Dateiname  03_03_WeeknumberDIN.xls
' Tabelle     CountGermanWeeks
' Modul      mdl_02_CountGermanWeeks
'=====

Function CountGermanWeeks(intYear As Integer) As Byte
    Dim bytDay1 As Byte
    Dim bytDay2 As Byte

    bytDay1 = DatePart("w", DateSerial(intYear, 1, 1), vbMonday)
    bytDay2 = DatePart("w", DateSerial(intYear, 12, 31), vbMonday)

    If bytDay1 = 4 Or bytDay2 = 4 Then
        CountGermanWeeks = 53
    Else
```

Grundlagen

Allgemein

Datum
Zeit

Formeln

Steuerelemente

Befehle
ausführen

Objekte

Diagramme

Ereignisse

User-Formen

Web/Mail

Extern

Gemischtes

Spezial

```

        CountGermanWeeks = 52
    End If
End Function

```

83 Feiertage mit Formeln berechnen

In Abbildung 62 sehen Sie das Dialogfenster FUNKTION EINFÜGEN. Die Kategorie DATUM & ZEIT enthält etliche Funktionen, aber leider fehlt hier die oft nachgefragte Funktion, um Feiertage berechnen zu können. Das mag daran liegen, dass es nicht nur unterschiedliche Feiertage in den einzelnen Ländern gibt, sondern sogar innerhalb Deutschlands die Feiertage in den einzelnen Bundesländern nicht einheitlich sind. Wer zum Beispiel in Nordrhein-Westfalen wohnt, darf in seinem Kalender den 1. November als Feiertag markieren. In Hessen gilt der 1. November »Allerheiligen« dagegen als gewöhnlicher Tag.

Ein Kalender ohne Kennzeichnung der Feiertage wäre kein richtiger Kalender. Aus diesem Grund helfen wir Excel ein bisschen auf die Sprünge.

Gesetzliche Feiertage, wie der 01. Januar, 01. Mai, 03. Oktober etc., lassen sich ohne Probleme formulieren, wenn das entsprechende Jahr bekannt ist. Für nachfolgende Formeln steht das Jahr 2003 als Ganzzahl in der Zelle A1.

```
=DATUM(A1;12;24) ' liefert 24.12.2003
```

Die kirchlichen Feiertage richten sich nach dem Ostersonntag des entsprechenden Jahres aus. Die derzeit kürzeste Osterformel stammt von Norbert Hetterich und erwartet für nachstehende Formel in der Zelle A1 eine Jahreszahl:

```
=DM((TAG(MINUTE(A1/38)/2+55)&" .4."&A1)/7;)*7-6) ' liefert 20.04.2003
```

Beachten Sie bitte, dass diese Formel nur im 1900-Datumssystem das korrekte Datum des Ostersonntags liefert. Die kürzeste Formel, die sowohl im 1900- als auch im 1904-Datumssystem funktioniert, wurde von unserem US-MVP-Kollegen Stephen Bullen entwickelt:

```
=DATUM(A1;3;28)+REST(24-REST(A1;19)*10,63;29)-REST(KÜRZEN(A1*5/4)+REST(24-REST(A1;19)*10,63;29)+1;7)
```

In unserer Beispieldatei finden Sie in dem Tabellenblatt FEIERTAGE_FORMELN eine Übersichtstafel der Jahre 2003 bis 2010 mit den geläufigsten deutschen Feiertagen und anderen wichtigen Ereignissen, wie zum Beispiel »Muttertag« oder »Rosenmontag«. Durch Änderung der Jahreszahlen in den Spaltenüberschriften in Zeile 3 werden die Daten der Feiertage neu berechnet.

84 Formeln in Tabelle anzeigen

Wir könnten die Formeln zu obiger Beispieldatei mühselig aus den einzelnen Zellen kopieren und in einer Tabelle auflisten, um sie hier darzustellen. Aktivieren Sie im Menü EXTRAS | OPTIONEN Registerkarte ANSICHT | FENSTEROPTIONEN den Menüpunkt FORMELN. Dadurch werden in der Tabelle, anstelle der durch Formeln erzeugten Werte, die Formeln angezeigt. Diese Einstellung gilt nicht für die gesamte Mappe, sondern kann für jedes Tabellenblatt einzeln aktiviert/deaktiviert werden. Alternativ verwenden Sie die Tastenkombination **[Strg] + [#]**. Hierbei wird

C8	=DM((TAG(MINUTE(C3/38)/2+55)&"4"&C3)/7;)*7-6									
A	B	C	D	E	F	G	H	I	J	
	Feiertage 2003 bis 2010									
		2003	2004	2005	2006	2007	2008	2009	2010	
	Neujahr	Mi 01.Jan	Do 01.Jan	Sa 01.Jan	So 01.Jan	Mo 01.Jan	Di 01.Jan	Do 01.Jan	Fr 01.Jan	
	Heilige Drei Könige (z.T.)	Mo 06.Jan	Di 06.Jan	Do 06.Jan	Fr 06.Jan	Sa 06.Jan	So 06.Jan	Di 06.Jan	Mi 06.Jan	
	Rosenmontag	Mo 03.Mrz	Mo 23.Feb	Mo 07.Feb	Mo 27.Feb	Mo 19.Feb	Mo 04.Feb	Mo 23.Feb	Mo 15.Feb	
	Karfreitag	Fr 18.Apr	Fr 09.Apr	Fr 25.Mrz	Fr 14.Apr	Fr 06.Apr	Fr 21.Mrz	Fr 10.Apr	Fr 02.Apr	
	Ostersonntag	So 20.Apr	So 11.Apr	So 27.Mrz	So 16.Apr	So 08.Apr	So 23.Mrz	So 12.Apr	So 04.Apr	
	Osternmontag	Mo 21.Apr	Mo 12.Apr	Mo 28.Mrz	Mo 17.Apr	Mo 09.Apr	Mo 24.Mrz	Mo 13.Apr	Mo 05.Apr	
	Maifeiertag	Do 01.Mai	Do 01.Mai	Do 01.Mai	Do 01.Mai	Do 01.Mai	Do 01.Mai	Do 01.Mai	Do 01.Mai	
	Muttertag	So 11.Mai	So 09.Mai	So 15.Mai	So 14.Mai	So 13.Mai	So 11.Mai	So 10.Mai	So 09.Mai	
	Christi Himmelfahrt	Do 29.Mai	Do 20.Mai	Do 05.Mai	Do 25.Mai	Do 17.Mai	Do 01.Mai	Do 21.Mai	Do 13.Mai	
	Pfingstsonntag	So 08.Jun	So 30.Mai	So 15.Mai	So 04.Jun	So 27.Mai	So 11.Mai	So 31.Mai	So 23.Mai	
	Pfingstmontag	Mo 09.Jun	Mo 31.Mai	Mo 16.Mai	Mo 05.Jun	Mo 28.Mai	Mo 12.Mai	Mo 01.Jun	Mo 24.Mai	
	Fronleichnam	Do 19.Jun	Do 10.Jun	Do 26.Mai	Do 15.Jun	Do 07.Jun	Do 22.Mai	Do 11.Jun	Do 03.Jun	
	Maria Himmelfahrt	Fr 15.Aug	So 15.Aug	Mo 15.Aug	Di 15.Aug	Mi 15.Aug	Fr 15.Aug	Sa 15.Aug	So 15.Aug	
	Tag der deutschen Einheit	Fr 03.Okt	So 03.Okt	Mo 03.Okt	Di 03.Okt	Mi 03.Okt	Fr 03.Okt	Sa 03.Okt	So 03.Okt	
	Allerheiligen	Sa 01.Nov	Mo 01.Nov	Di 01.Nov	Mi 01.Nov	Do 01.Nov	Sa 01.Nov	So 01.Nov	Mo 01.Nov	
	Heilig Abend	Mi 24.Dez	Fr 24.Dez	Sa 24.Dez	So 24.Dez	Mi 24.Dez	Mi 24.Dez	Do 24.Dez	Fr 24.Dez	
	1. Weihnachtstag	Do 25.Dez	Sa 25.Dez	So 25.Dez	Mo 25.Dez	Di 25.Dez	Do 25.Dez	Fr 25.Dez	Sa 25.Dez	
	2. Weihnachtstag	Fr 26.Dez	So 26.Dez	Mo 26.Dez	Di 26.Dez	Mi 26.Dez	Fr 26.Dez	Sa 26.Dez	So 26.Dez	
	Silvester	Mi 31.Dez	Fr 31.Dez	Sa 31.Dez	So 31.Dez	Mo 31.Dez	Mi 31.Dez	Do 31.Dez	Fr 31.Dez	

Abbildung 64: Feiertagsberechnung mit Formeln

bei jedem Drücken der Tastenkombination der Anzeigemodus zwischen WERTE und FORMELN gewechselt.

Innerhalb eines Tabellenblattes ist es standardmäßig leider nicht möglich, einen Bereich als Formeln und gleichzeitig einen anderen Bereich als Werte darzustellen.

Aber hier gibt es eine wesentliche Vereinfachung aus den alten Excel4-Makro-Zeiten, die wir Ihnen nicht vorenthalten möchten. Sie finden die dazugehörige Tabelle *im* Tabellenblatt *FEIERTAGE_FORMELN_SICHTBAR* der Beispieldatei.

Hinweis

In der deutschen Excel-Version 11, auch Excel 2003 genannt, werden *Excel4-Makros* ebenfalls unterstützt. Die Sicherheitseinstellung muss auf NIEDRIG oder MITTEL eingestellt werden.

Zunächst fügen Sie neben Spalte C eine Hilfsspalte ein, indem Sie den Spaltenkopf C anklicken, so dass die ganze Spalte markiert ist. Mit Rechtsklick innerhalb der Markierung erhalten Sie ein Kontextmenü, wählen Sie daraus den Eintrag ZELLEN EINFÜGEN.

Die nächste Überlegung ist, welche Formel in welcher Zelle angezeigt werden soll. Wir möchten die Ausgabe einer Formel in der jeweils rechts neben der Formelzelle liegenden Zelle erreichen. Die Formel aus Zelle C4 soll also in D4 angezeigt werden. Dementsprechend markieren Sie nun die Zelle D4, klicken nacheinander auf das Menü EINFÜGEN | NAMEN | DEFINIEREN und schreiben in das Eingabefeld NAMEN IN DER ARBEITSMAPPE das Wort »Formel«. In die Eingabezeile BEZIEHT SICH AUF wird das *Excel4-Makro* eingetragen. Für die Formelanzeige lautet die Syntax:

```
=ZELLE.ZUORDNEN(6;Bezug)
```

Ersetzen Sie entsprechend unserem Beispiel das Wort *Bezug* durch C4 und klicken Sie dann auf HINZUFÜGEN. Excel vervollständigt die genaue Referenzierung automatisch. Schließen Sie den Dialog mit einem Klick auf die Schaltfläche OK.

Grundlagen

Allgemein

Datum
Zeit

tungen

Steuer
elemente

Befehl
leisten

Objekt

Diagramm

Ereignisse

User-
Form

Web/
Mail

Extern

Gemischtes

Spezial



Abbildung 65: Formelanzeige über definierten Namen

Schreiben Sie in die Zelle D4 die Formel `=Formel` und kopieren Sie diese bis an das Tabellenende. Das Resultat sehen Sie in Abbildung 66. Die Formel der jeweils linken Nebenzelle wird als Ergebnis angezeigt. Falls beim Öffnen der Datei diese Formeln den Fehlerwert `#NAME?` anzeigen, müssen die Formeln eventuell neu berechnet werden. Markieren Sie den Zellbereich und drücken Sie für jede Zelle die `[F2]`-Taste gefolgt von der `[Enter]`-Taste.

D8	=Formel		
	B	C	D
1			
2	Feiertage 2003 b		
3		2003	
4	Neujahr	Mi 01.Jan	=DATUM(C\$3;1;1)
5	Heilige Drei Könige (z.T.)	Mo 06.Jan	=DATUM(C\$3;1;6)
6	Rosenmontag	Mo 03.Mrz	=C8-48
7	Karfreitag	Fr 18.Apr	=C8-2
8	Ostersonntag	So 20.Apr	=DM((TAG(MINUTE(C3/38)/2+55)&" 4."&C3/7;)*7-6
9	Ostermontag	Mo 21.Apr	=C8+1
10	Maifeiertag	Do 01.Mai	=DATUM(C\$3;5;1)
11	Muttertag	So 11.Mai	=(DATUM(C\$3;5;1))+15-(WOCHENTAG((DATUM(C\$3;5;1);1))
12	Christi Himmelfahrt	Do 29.Mai	=C8+39
13	Pfingstsonntag	So 08.Jun	=C8+49
14	Pfingstmontag	Mo 09.Jun	=C8+50
15	Fronleichnam	Do 19.Jun	=C8+60
16	Maria Himmelfahrt	Fr 15.Aug	=DATUM(C\$3;8;15)
17	Tag der deutschen Einheit	Fr 03.Okt	=DATUM(C\$3;10;3)
18	Allerheiligen	Sa 01.Nov	=DATUM(C\$3;11;1)
19	Heilig Abend	Mi 24.Dez	=DATUM(C\$3;12;24)
20	1. Weihnachtstag	Do 25.Dez	=DATUM(C\$3;12;25)
21	2. Weihnachtstag	Fr 26.Dez	=DATUM(C\$3;12;26)
22	Silvester	Mi 31.Dez	=DATUM(C\$3;12;31)

Abbildung 66: Formelanzeige über definierten Namen

85 Funktion zur Formelanzeige

Für die Anzeige von Formeln können Sie anstelle eines definierten Namens mit *Excel4-Makro* auch eine benutzerdefinierte Funktion verwenden.

Tipp

Ausführliche Informationen und Beispiele zu Funktionen finden Sie in der Kategorie »Einstieg«.

Die nachfolgende Funktion `ShowFormula` wird in allen Excel-Versionen ab Excel 97 unterstützt. Damit die Formel im Tabellenblatt berechnet werden kann, speichern Sie diese in einem Standardmodul. Die Formel `=ShowFormula(A1)` liefert als Ergebnis die Formel der Zelle A1 im Textformat.

Im Tabellenblatt FUNCTION_SHOWFORMULA der Beispieldatei werden die Formeln der Nebenzelle über die ShowFormula-Funktion angezeigt.

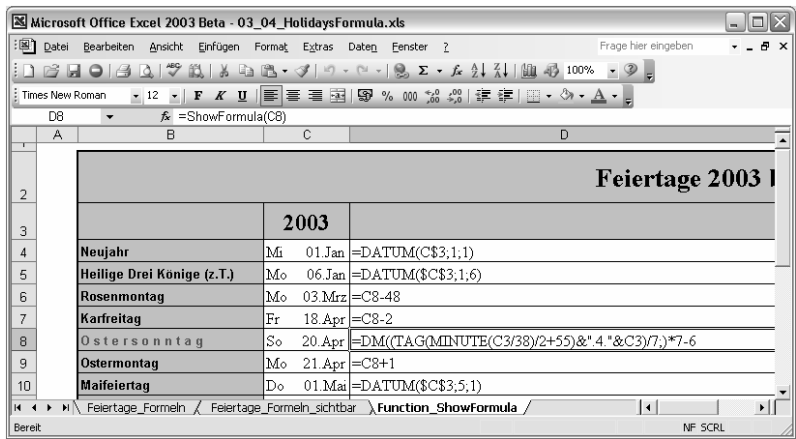


Abbildung 67: Formelanzeige in Excel 2003

```
' =====
' Auf CD      Buchdaten\Beispiele\03_Datum_Zeit\
' Dateiname   03_04_HolidaysFormula.xls
' Tabelle     Function_ShowFormula
' Modul       mdl_01_ShowFormula
' =====

Function ShowFormula(rngCell As Range) As String
    ShowFormula = rngCell.FormulaLocal
End Function
```

86 Feiertage mit VBA berechnen

Zur Berechnung des Ostersonntags und der daraus resultierenden kirchlichen Feiertage gibt es neben der bereits erwähnten Formel auch eine benutzerdefinierte Funktion EASTER. Dadurch wird die lange »Hetterich-Formel« durch nachfolgende benutzerdefinierte Tabellenblatt-Funktion ersetzt. Voraussetzung ist wieder eine Jahreszahl als Ganzzahl in der Zelle A1.

=Easter(A1) *' liefert im Jahr 2003 den 20.04.2003*

```
' =====
' Auf CD      Buchdaten\Beispiele\03_Datum_Zeit\
' Dateiname   03_05_HolidaysVBA.xls
' Tabelle     Feiertage_VBA
' Modul       mdl_01_FunctionEaster
' =====

Function EASTER(intYear As Integer) As Date
    Dim intDummy As Integer

    intDummy = (((255 - 11 * (intYear Mod 19)) - 21) Mod 30) + 21
```

Grundlagen

Allgemein

Datum
Zeit

Formeln

Steuerelemente

Befehle
leisten

Objekt

Diagramm

Ereignisse

User-Form

Web/
Mail

Extern

Gemischtes

Spezial

```

EASTER = DateSerial(intYear, 3, 1) + intDummy + (intDummy > 48) _
        + 6 - ((intYear + intYear \ 4 + intDummy _
        + (intDummy > 48) + 1) Mod 7)

```

End Function

87 Tageskalender erstellen

Für unser nächstes Beispiel haben wir uns eine kleine Spielerei ausgedacht. Sicherlich kennen Sie die kleinen Abreißkalender. Dort zeigt das oberste Blatt den aktuellen Tag an.



Abbildung 68: Tageskalender

In der Beispieldatei, Tabelle DAILY, sehen Sie die Einzelheiten dazu. Abbildung 69 besteht aus vier Zellen, die teilweise aus mehreren Zellen verbunden sind. Diese sind unterschiedlich formatiert und orientieren sich alle an dem Inhalt der Zelle E4, welche das Datum enthält.

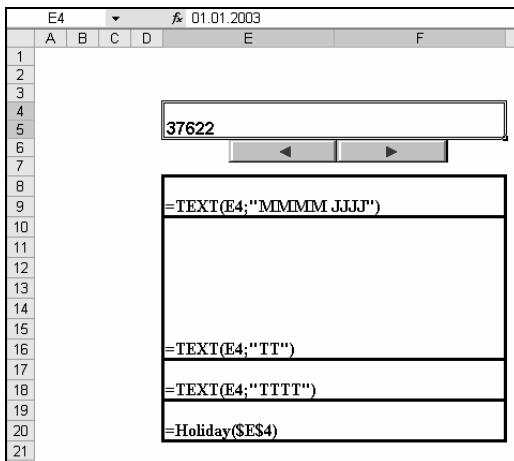


Abbildung 69: Tageskalender mit Formeln

Das Datum verändern Sie durch eine Datumseingabe in E4 oder durch Anklicken des Drehfeldes. Dadurch springt das Datum jeweils einen Tag vor oder zurück. Das Drehfeld `SpinButton` wurde aus der Symbolleiste `STEUERLEMENT-TOOLBOX` eingefügt.

Im Eigenschaften-Fenster dieses Drehfeldes wurde der `LINKEDCELL`-Eigenschaft der Zellbezug E4 zugewiesen. Dadurch wird der Wert des Drehfeldes in die Zelle E4 geschrieben und umgekehrt wird bei Eingabe in Zelle E4 der Wert `Value` des Drehfeldes angeglichen.

Die Eigenschaft `Min` enthält den Wert 37622, was dem 01.01.2003 entspricht. `Max` beträgt 74510, entsprechend dem 31.12.2103. Durch diese beiden Werte wird die Unter- und Obergrenze des

Hinweis

Eigenschaften von Steuerelementen lassen sich nur im Entwurfsmodus einstellen. Aktivieren Sie den Entwurfsmodus durch Anklicken des türkisfarbenen Dreiecks in der Symbolleiste STEUERLEMENT-TOOLBOX. Wählen Sie ein bereits eingefügtes Steuerelement durch Anklicken aus und klicken Sie auf das zweite Symbol in der Symbolleiste. Dadurch wird das Eigenschaften-Fenster angezeigt.

Die Funktionalität des Drehfeldes erreichen Sie wieder durch Deaktivierung des Entwurfsmodus.

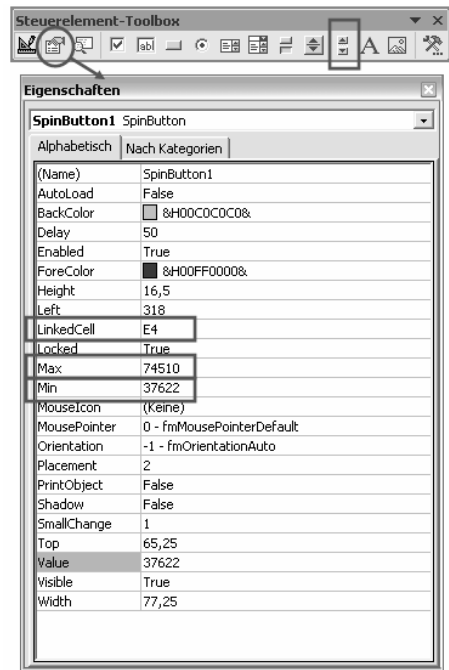


Abbildung 70: Eigenschaftsfenster Drehfeld

Drehfeldes bestimmt. Über die weiteren Eigenschaften können Sie die Farben, Größe, Name etc. bestimmen. Hier wird Ihrer Kreativität kaum Grenzen gesetzt.

In der untersten Zelle des Tagesblattes werden die per Code hinterlegten Feiertage über die `Holiday`-Funktion ausgegeben. Beim Funktionsaufruf wird der Bezug `=$E$4`, inklusive dem enthaltenen Datum aus Zelle E4 von der Formel an die Funktion übergeben. Die Variable `dDate` enthält dadurch das Datum der Zelle E4 und wird in der folgenden `Select Case`-Anweisung mit vorgegebenen Daten verglichen. Mit den Codezeilen

```
Case Is = DateAdd("D", 1, datEastern)
    Holiday = "Ostermontag"
```

wird ein Datumswert mit dem Inhalt der Variablen `dDate` verglichen. Bei Übereinstimmung wird ein definierter Text an die Funktion übergeben und die Anweisung verlassen. Dieser Text wird nach Beendigung der Funktion in der aufrufenden Zelle als Ergebnis der `Holiday`-Funktion angezeigt.

Grundlagen

Allgemein

Datum
Zeit

ungen

Steuerelemente

Befehle
leisten

Objekt

Diagramm

Ereignisse

User-Form

Web/
Mail

Extern

Gemischtes

Spezial

Sie können beliebig viele solcher Case-Blöcke innerhalb der Select Case-Anweisungen angeben. In unserem Beispiel haben wir unter anderem die Tage der Zeitumstellung (Sommerzeit) und das Datum des Muttertages aufgenommen. Ebenso können Sie auch individuelle Anlässe wie Geburtstage angeben. Der folgende Case-Block gibt am 03.12. eines Jahres den Hinweistext »Geburtstag« zurück.

```
Case Is = DateSerial(intYear, 12, 3)
    Holiday = "Geburtstag"
```

Der nachfolgende Code befindet sich in der angegebenen Beispieldatei. In dem Tabellenblatt TODAY wird der jeweils aktuelle Tag auf einem Tageskalender angezeigt. Hierbei wird ebenfalls die Holiday-Funktion über den nachstehenden Code angewandt.

```
'=====
' Auf CD      Buchdaten\Beispiele\03_Datum_Zeit\
' Dateiname  03_05_HolidaysVBA.xls
' Tabelle    Daily
' Tabelle    Today
' Modul      mdl_02_FunctionHoliday
'=====
```

```
Function Holiday(dDate As Date) As String
    Dim intYear As Integer
    Dim datEastern As Date

    intYear = Year(dDate)
    datEastern = EASTER(intYear)

    Select Case dDate
        Case Is = DateSerial(intYear, 1, 1)
            Holiday = "NeuJahr"
        Case Is = DateSerial(intYear, 1, 6)
            Holiday = "Dreikönig"
        Case Is = DateAdd("D", -48, datEastern)
            Holiday = "Rosenmontag"
        Case Is = DateSerial(intYear, 4, 0) - _
            (Weekday(DateSerial(intYear, 4, 0), 2) Mod 7)
            Holiday = "Beginn Sommerzeit"
        Case Is = DateAdd("D", -2, datEastern)
            Holiday = "Karfreitag"
        Case Is = datEastern
            Holiday = "Ostersonntag"
        Case Is = DateAdd("D", 1, datEastern)
            Holiday = "Ostermontag"
        Case Is = DateSerial(intYear, 5, 1)
            Holiday = "Tag der Arbeit"
        Case Is = IIf(DateSerial(intYear, 5, 15) - _
            (Weekday(DateSerial(intYear, 5, 1), 2) Mod 7) = _
            datEastern + 49, DateSerial(intYear, 5, 15) - _
            (Weekday(DateSerial(intYear, 5, 1), 2) Mod 7) - _
            7, DateSerial(intYear, 5, 15) - _
```

```

        (Weekday(DateSerial(intYear, 5, 1), 2) Mod 7))
    Holiday = "Muttertag"
Case Is = DateAdd("D", 39, datEastern)
    Holiday = "Christi Himmelfahrt"
Case Is = DateAdd("D", 49, datEastern)
    Holiday = "Pfingstsonntag"
Case Is = DateAdd("D", 50, datEastern)
    Holiday = "Pfingstmontag"
Case Is = DateAdd("D", 60, datEastern)
    Holiday = "Fronleichnam"
Case Is = DateSerial(intYear, 8, 15)
    Holiday = "Maria Himmelfahrt"
Case Is = DateSerial(intYear, 10, 3)
    Holiday = "Deutsche Einheit"
Case Is = DateSerial(intYear, 11, 0) - (Weekday( _
    DateSerial(intYear, 11, 0), 2) Mod 7)
    Holiday = "Ende Sommerzeit"
Case Is = DateSerial(intYear, 11, 22) - _
    (DateSerial(intYear, 11, 18) Mod 7)
    Holiday = "Buß- und Betttag"
Case Is = DateSerial(intYear, 10, 31)
    Holiday = "Reformationstag"
Case Is = DateSerial(intYear, 11, 1)
    Holiday = "Allerheiligen"
Case Is = DateSerial(intYear, 12, 3)
    Holiday = "Geburtstag"
Case Is = DateSerial(intYear, 12, 24) - _
    (Weekday(DateSerial(intYear, 12, 24), 2) Mod 7) - 21
    Holiday = "1. Advent"
Case Is = DateSerial(intYear, 12, 24) - _
    (Weekday(DateSerial(intYear, 12, 24), 2) Mod 7) - 14
    Holiday = "2. Advent"
Case Is = DateSerial(intYear, 12, 24) - _
    (Weekday(DateSerial(intYear, 12, 24), 2) Mod 7) - 7
    Holiday = "3. Advent"
Case Is = DateSerial(intYear, 12, 24) - _
    (Weekday(DateSerial(intYear, 12, 24), 2) Mod 7)
    Holiday = "4. Advent"
Case Is = DateSerial(intYear, 12, 24)
    Holiday = "Heilig Abend"
Case Is = DateSerial(intYear, 12, 25)
    Holiday = "erster Weihnachtstag"
Case Is = DateSerial(intYear, 12, 26)
    Holiday = "zweiter Weihnachtstag"
Case Is = DateSerial(intYear, 12, 31)
    Holiday = "Silvester"
Case Else
    Holiday = ""
End Select
End Function
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

88 100-jähriger Kalender und Arbeitszeittabelle

In der Beispieldatei haben wir einen 100-jährigen Kalender erstellt. Durch Auswahl eines Monats und Jahres über zwei Drehfelder werden die Tage eines Monats mitsamt den dazugehörigen Kalenderwochen und Feiertagen dargestellt.

Über das Change-Ereignis der Drehfelder wird die Prozedur `Sheetname` aufgerufen. Die `SpinButton_Change`-Ereignisse werden im Codebereich des Tabellenblattes gespeichert.

```
'=====
' Auf CD      Buchdaten\Beispiele\03_Datum_Zeit\
' Dateiname   03_06_Calendar.xls
' Ereignis    Tabelle2(Januar 03)
'=====

Private Sub SpinButton1_Change()
    Sheetname
End Sub

Private Sub SpinButton2_Change()
    Sheetname
End Sub
```

Diese Prozeduren rufen das Makro `Sheetname` zur Beschriftung des Tabellenregisters auf, welches in einem Standardmodul hinterlegt ist. Wählen Sie in einem weiteren Blatt das gleiche Datum wie in einem vorherigen Tabellenblatt aus, würde die Prozedur versuchen, einen bereits verwendeten Namen diesem Tabellenblatt zuzuweisen, was zu einer Fehlermeldung führt.

Für diesen Fall fangen wir die Fehlermeldung mit einer `On Error GoTo`-Anweisung ab und benennen das Tabellenblatt zusätzlich mit seiner Indexnummerierung.

```
'=====
' Auf CD      Buchdaten\Beispiele\03_Datum_Zeit\
' Dateiname   03_06_Calendar.xls
' Modul       mdl_01_Sheetname
'=====

Sub Sheetname()
    On Error GoTo ErrorHandler
    ActiveSheet.Name = Format(Range("C5"), "MMMM YY")
    Exit Sub
ErrorHandler:
    ActiveSheet.Name = _
        Format(Range("C5"), "MMMM YY") & "_" & ActiveSheet.Index
End Sub
```

Über die bedingte Formatierung werden Sonn- und Feiertage in den Spalten A und C hervorgehoben.

Auf der Basis dieses Kalenders lässt sich leicht eine Arbeitszeittabelle erstellen, wie in unserem Beispiel.

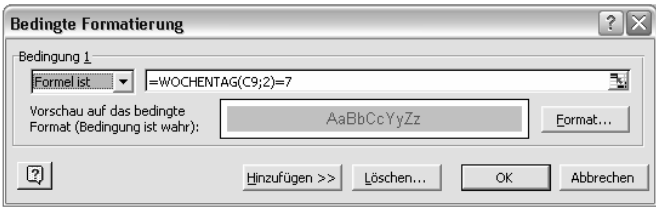


Abbildung 71: Sonntage mit bedingter Formatierung hervorheben

	A	B	C	D	E	F	G	H	I	J	K
1	Januar 2003										
2											
3											
4	Feiertag	KW	Datum	Arbeits Beginn	Pause	Arbeits Ende	Gesamt Std.	Nacht Std.	Sonntag Std.	Feiertag Std.	Spesen EUR
5	Neujahr	01	Mi 01.01	08:00	00:30	17:00	08:30			08:30	6,00 €
6			Do 02.01	08:00	01:00	07:59	22:59	09:00			12,00 €
7			Fr 03.01	08:00		19:00	11:00				6,00 €
8			Sa 04.01	19:00	01:00	04:00	08:00	07:00	04:00		
9			So 05.01								
10		02	Mo 06.01								
11			Di 07.01								
12			Mi 08.01								
13			Fr 14.01								
14			Sa 15.01								
15			So 16.01								
16		03	Mo 20.01								
17			Di 21.01								
18			Mi 22.01								
19			Do 23.01								
20			Fr 24.01								
21			Sa 25.01								
22			So 26.01								
23		04	Mo 27.01								
24			Di 28.01								
25			Mi 29.01								
26			Do 30.01								
27			Fr 31.01								
28	Gesamt						50:29	16:00	04:00	08:30	24,00 €
29	Lohnberechnung						504,83 €	40,00 €	20,00 €	85,00 €	24,00 €
30	Bruttolohn						673,83 €				

Abbildung 72: Arbeitszeittabelle

In der Tabelle werden nur der Arbeitsbeginn, das Arbeitsende sowie die Pausen eingetragen. Die Ergebnisse werden über Formeln ausgegeben, welche in den Zellen der Spalten G bis K hinterlegt sind.

Bei einer Arbeitszeit von mehr als 24 Stunden soll ein entsprechender Spesensatz gewährt werden. Dies erreichen Sie, indem Sie als Arbeitsbeginn zum Beispiel 8:00 und -ende 7:59 ohne Pause eingeben. Da dies eine große Fehlerquelle darstellt, wurde über DATEN | GÜLTIGKEIT | BENUTZERDEFINIERT die Formel `=D5<>$F5` eingegeben, so dass der Beginn und Ende der Arbeitszeit nicht identisch sein dürfen. Weitere Informationen zum Thema Gültigkeit finden Sie in der Kategorie »Auswertungen«.

In dem Tabellenblatt DATENBANK der Beispieldatei hinterlegen Sie alle veränderbaren Angaben für eine individuelle Berechnung. Unter Zuhilfenahme der `Easter`-Funktion aus Rezept 86, welche im Modul `mdl_02_Easter` hinterlegt ist, erstellen Sie eine Liste der Feiertage entsprechend Ihres Bundeslandes.

Die Nachtstunden werden anhand einer vorgegebenen Anfangs- und Endzeit berechnet. Ebenso die Höhe der Spesen, des Lohnes und der Zuschläge. Die hinterlegten Werte werden in den Formeln eingelesen. Die Nachtzeit zum Beispiel wird nur für die Zeit berechnet, die zwischen den vorgegebenen Uhrzeiten liegt. In unserem Beispiel also zwischen 20:00 Uhr und 06:00 Uhr.

	A	B	C	D	E
1					
2		Jahr			
3		2003			
4		Feiertage		Lohn und Zuschläge	
5		1. Januar 2003	Neu.Jahr	Stundenlohn	10,00 €
6				Nachtzuschlag	25%
7		18. April 2003	Karfreitag	Sonntagszuschlag	50%
8		20. April 2003	Ostersonntag	Feiertagszuschlag	100%
9		21. April 2003	Ostermontag		
10					
11		1. Mai 2003	Tag der Arbeit		
12				Nacht Arbeitszeit	
13		29. Mai 2003	Christi Himmelfahrt		
14		8. Juni 2003	Pfingstsonntag	Beginn	20:00
15		9. Juni 2003	Pfingstmontag	Ende	06:00
16					
17		19. Juni 2003	Fronleichnam		
18		3. Oktober 2003	Tag der dt. Einheit	Spesen	
19				Gesamtzeit >	
20					0,00 €
21		25. Dezember 2003	1. Weihnachtstag	08:00	6,00 €
22		26. Dezember 2003	2. Weihnachtstag	14:00	12,00 €
23		1. Januar 2004	Neu.Jahr	23:59	24,00 €
24					

Abbildung 73: Datenbank für Arbeitszeittabelle

Für die Feiertagsliste und die in den Formeln verwendeten Werte wurden in dieser Tabelle den jeweiligen Zellen und Bereichen Namen zugewiesen, um bei der Formelerstellung einen guten Überblick zu behalten.

Eine Gesamtübersicht aller im Tabellenblatt vorhandenen Namen erhalten Sie über den Menüpunkt EINFÜGEN | NAMEN | EINFÜGEN | LISTE EINFÜGEN. Im Tabellenblatt wird diese Auflistung ab der aktuellen Zellposition ausgegeben:

Name	Bezug
Druckbereich	=Datenbank!\$B\$1:\$E\$23
Feiertage	=Datenbank!\$B\$5:\$C\$23
Feiertagszuschlag	=Datenbank!\$E\$9
Jahr	= 'Januar 03' ! \$G\$2
Monat	= 'Januar 03' ! \$D\$3
Nacht_Beginn	=Datenbank!\$E\$14
Nacht_End	=Datenbank!\$E\$15
Nachtzuschlag	=Datenbank!\$E\$7
Sonntagszuschlag	=Datenbank!\$E\$8
Spesen	=Datenbank!\$D\$20:\$E\$23
Stundenlohn	=Datenbank!\$E\$6

Tabelle 16: Erstellte Namensliste aller vorhandener Namen

Nachfolgende Tabelle zeigt die verwendeten Formeln bezogen auf die Zeile 5.

Spalte	Überschrift	Formel
A	Feiertag	=WENN(ISTFEHLER(SVERWEIS(C5;Feiertage;2;0)))>><<; SVERWEIS(C5;Feiertage;2;0))
B	KW	=WENN(ISTFEHLER(WeekNumberDIN(C5)))>><<; WeekNumberDIN(C5))
C5	Datum	=DATUM(Jahr;Monat;1)
C6 – C35	Datum	=WENN(\$C5=>><<;>><<;WENN(MONAT(\$C\$5)=MONAT(\$C5+1); \$C5+1;>><<))
G	Gesamt Std.	=WENN(ODER(D5=>><<;F5=>><<))>><<; WENN(REST(F5-D5;1)-E5<0;0;REST(F5-D5;1)-E5))
H	Nacht Std.	=WENN(G5=>><<;>><<;MAX(0;MIN(F5+WENN(F5<=D5;1;0); Nacht_Ende+1)-MAX(D5;Nacht_Beginn))+ MAX(0;MIN(Nacht_Ende;F5)-D5)-E5)
I	Sonntag Std.	=WENN(G5=>><<;>><<;WENN(WOCHENTAG(C5;2)=7; WENN(H5<>><<;1-D5-E5;F5-D5-E5);0)+ WENN(C6=>><<;0;WENN(UND(WOCHEN- TAG(C6;2)=7;H5<>><<); F5;0)))
J	Feiertag Std.	=WENN(G5=>><<;>><<;WENN(A5<>><<; WENN(H5<>><<;1-D5-E5;F5-D5-E5);0)+ WENN(C6=>><<;0;WENN(UND(A6<>><<;H5<>><<);F5;0)))
K	Spesen EUR	=WENN(G5=>><<;>><<;SVERWEIS(G5;Spesen;2;2))

Tabelle 17: Formeln der Arbeitszeittabelle

Diese Beispieldatei ist nur für ein Kalenderjahr konzipiert, da für die Berechnung der Feiertage im Tabellenblatt DATENBANK das ausgewählte Jahr des CALENDAR-Blattes herangezogen wird. Sie können aber innerhalb der Datei das CALENDAR-Blatt kopieren und in der Kopie einen anderen Monat auswählen.

89 Lohn berechnen

In der Beispieldatei wird unterhalb der Arbeitszeittabelle der Lohn der einzelnen Spalten sowie der Gesamtlohn berechnet.

G3B	=G3B*Stundenlohn*24										
	A	B	C	D	E	F	G	H	I	J	K
1											
4	Feiertag	KW	Datum	Arbeits Beginn	Pause	Arbeits Ende	Gesamt Std.	Nacht Std.	Sonntag Std.	Feiertag Std.	Spesen EUR
29			Sa 25.01.								
30			So 26.01.								
31		05	Mo 27.01.								
32			Di 28.01.								
33			Mi 29.01.								
34			Do 30.01.								
35			Fr 31.01.								
36	Gesamt						50:29	16:00	04:00	08:30	24,00 €
37											
38	Lohnberechnung						504,83 €	40,00 €	20,00 €	85,00 €	24,00 €
39	Bruttolohn						673,83 €				

Abbildung 74: Lohnberechnung

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

Die Zelle, welche den Stundensatz enthält, wurde mit dem Namen STUNDENLOHN benannt. Für die Lohnberechnung in Zelle G38 werden die errechneten Gesamtstunden aus G36 mit dem hinterlegten Stundensatz und der Zahl 24 multipliziert.

```
=G36*Stundenlohn*24      ' liefert 504,83 _
```

Diese Formel entspricht den Werten: 50:29*10*24. Die Ergebniszellen der Lohnberechnungen sind mit dem Währungsformat formatiert. Bei der Zuschlagsberechnung für die Nacht-, Sonntags- und Feiertagsstunden wird zusätzlich der angegebene Prozentsatz aus der DATENBANK berücksichtigt. Am Beispiel der Nachtstunden wird folgende Formel angewandt.

```
=H36*Stundenlohn*24*Nachtzuschlag      ' liefert 40,00 _
```

Zur Verdeutlichung stellen wir die dazugehörigen Werte noch einmal dar:

```
16:00*10*24*25%
```

90 Minuszeiten berechnen

Wenn es um die Berechnung von Uhrzeiten geht, sorgt Excel für einige Verwirrung, wenn die Zeitdifferenz einen *Minuswert* ergibt. Subtrahiert man eine größere Uhrzeit von einer niedrigeren, erscheinen als Ergebnis standardmäßig Rauten (#), oder auch Gitterzäune genannt, in der Zelle. Die Ursache liegt entweder darin, dass Excel das bereits verwendete Format der Quellzellen hh:mm übernimmt, oder der Anwender dieses Format angibt, um einen Uhrzeitwert zu erhalten. Ändert man das Format auf STANDARD oder ZAHL, wird der errechnete Wert korrekt als Dezimalzahl angezeigt.

	C2		fx: =B2-A2		
	A	B	C	D	E
1	Start	End	Differenz	Formel	Format
2	08:00	07:00	#####	=B2-A2	hh:mm
3	08:00	07:00	-0,04	=B2-A3	Zahl; 2 Nachkommastellen
4					

Abbildung 75: Minuszeiten berechnen

Hinweis

Excel hat also Probleme damit, negative Uhrzeiten im UHRZEITFORMAT darzustellen und nicht mit der Berechnung von Uhrzeiten.

Bei der Zeitberechnung und deren Darstellung kann man mit VBA durch Vorgabe des Formates eine korrekte Ausgabe der Zeitdifferenz im Textformat erzielen.

```
'=====
' Auf CD      Buchdaten\Beispiele\03_Datum_Zeit\
' Dateiname   03_07_MinusTime.xls
' Tabelle     Function_NegativeTime
' Modul       mdl_01_MinusTime
```



```
' =====  
  
Sub MinusTime()  
    MsgBox "B2-A2=" & _  
        vbNewLine & _  
        Format(Range("B2") - Range("A2"), "- hh:mm")  
  
    Range("G2") = Format(Range("B2") - Range("A2"), "- hh:mm")  
End Sub
```

Hier sehen Sie die richtige Ausgabe der Zeitdifferenz im Meldungsfenster und im Textformat in der Zelle G2. Ein Weiterrechnen ist mit diesem Textwert allerdings nicht ohne weiteres möglich.

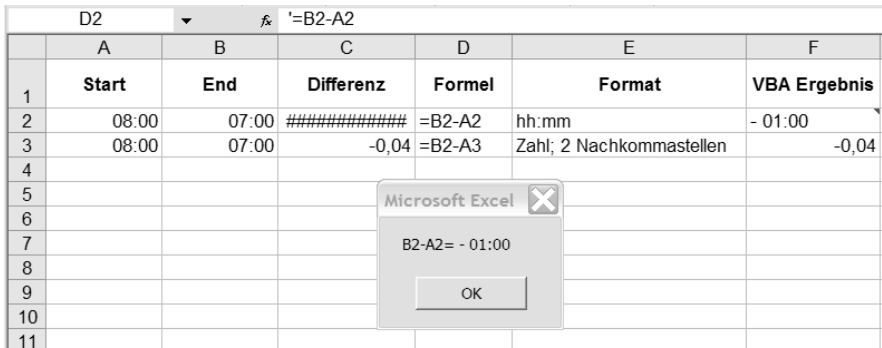


Abbildung 76: Negative Zeitberechnung mit VBA

Mit einigen verschachtelten Tabellenfunktionen erzielen Sie das gleiche Ergebnis: eine negative Zeitdarstellung im Textformat. Da dieses Ergebnis im Textformat vorliegt, kann man damit nicht ohne weiteres weiterrechnen.

```
= "-" & TEXT(STUNDE((B2-A2)*-1); "00") & ":" & TEXT(MINUTE((B2-A2)*-1); "00")
```

oder

```
=TEXT(ABS(B2-A2); "- hh:mm")
```

91 Negative Zeitberechnung mit Funktion

Aufgrund der obigen Feststellung, dass VBA negative Zeiten berechnen und im Uhrzeitformat darstellen kann, verwenden wir zur Berechnung im Tabellenblatt eine benutzerdefinierte Funktion. Diese erspart Ihnen die komplizierte Eingabe der oben gezeigten Formellösungen.

Die Zeitdifferenz zwischen B2 und A2 wird nicht mehr mit einer Formel, sondern mit der Funktion =NegativeTime(B2;A2) berechnet.

```
' =====  
  
' Auf CD Buchdaten\Beispiele\03_Datum_Zeit\  
' Dateiname 03_07_MinusTime.xls
```

Grundlagen

Allgemein

Datum
Zeit

Formeln

Steuerungselemente

Befehle
ausführen

Objekte

Diagramme

Ereignisse

User-Formen

Web/Mail

Extern

Gemischtes

Spezial

```
' Tabelle      Function_NegativeTime
' Modul       mdl_02_Function_NegativeTime
' =====

Function NegativeTime(rng1 As Range, rng2 As Range)
    If rng1 < rng2 Then
        NegativeTime = Format(rng1 - rng2, "- hh:mm")
    Else
        NegativeTime = Format(rng1 - rng2, "hh:mm")
    End If
End Function
```

Hinweis

Diese Funktion liefert in einer Zelle das korrekte Ergebnis einer negativen Zeitdifferenz im *Textformat*. Das bedeutet, dass mit diesem Ergebnis keine weiteren Berechnungen durchgeführt werden können. Ist die Endzeit größer als die Anfangszeit, so gibt die Funktion das positive Ergebnis ebenfalls im *Textformat* aus.

92 1904-Datumswerte für Minuszeiten

Die einfachste Möglichkeit für die richtige negative Zeitberechnung und Darstellung ist die Umstellung der 1904-DATUMSWERTE über das Menü EXTRAS | OPTIONEN Registerkarte BERECHNEN. Aktiviert man dort das Kontrollkästchen 1904-DATUMSWERTE, kann Excel Minuszeiten anzeigen und berechnen. Diese Einstellung gilt für die gesamte Arbeitsmappe und behält auch bei erneutem Öffnen die gewählte Einstellung.

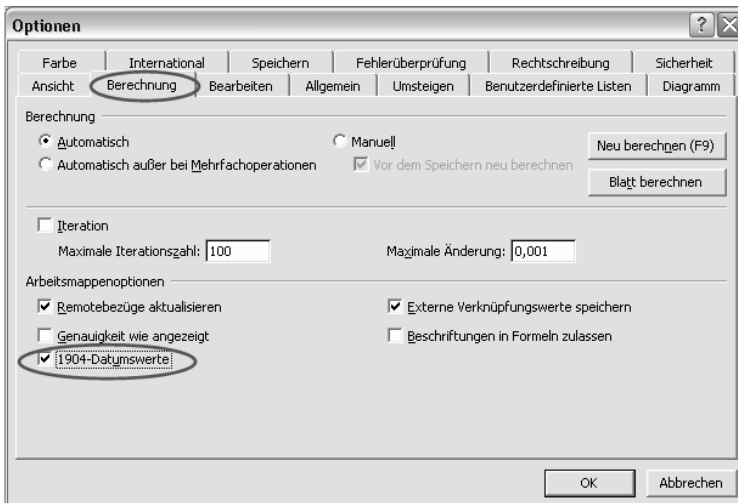


Abbildung 77: Umstellen 1904-Datumswerte

Diese Methode ist zwar einfach, birgt allerdings dann ihre Tücken, wenn sich in der aktiven Arbeitsmappe außer Zeiten bereits Datumswerte befinden, oder die Zelleingaben via VBA manipuliert werden. Die vorhandenen Datumswerte werden bei der Umstellung um vier Jahre und einen Tag hochgezählt.

Wert	1900-Datumswerte	1904-Datumswerte
-3	#####	-04.01.1904
-2	#####	-03.01.1904
-1	#####	-02.01.1904
0	00.01.1900	01.01.1904
1	01.01.1900	02.01.1904
2	02.01.1900	03.01.1904
3	03.01.1900	04.01.1904
37653	01.02.2003	02.02.2007

Tabelle 18: Datumswerte bei verschiedenen Datumssystemen

Bei normaler Datumsberechnung im 1900-Datumssystem wird die Zahl der seit dem 31.12.1899 vergangenen Tage gezählt, bei der 1904-Datumsberechnung die Tage, die ab dem 01.01.1904 ver-
gangen sind.

In der Beispieldatei haben wir im Tabellenblatt 1904-DATUMSWERTE einige Testdaten bereitge-
stellt. Über eine Umschaltfläche `ToggleButton` wird das Datumssystem gewechselt. So haben Sie
einen direkten Vergleich zwischen den beiden Datumssystemen.

Als Besonderheit des Makros zur Umstellung des Datumssystems wird über die `IIf`-Funktion die
Beschriftung `Caption` der Umschaltfläche entsprechend dem eingestellten Datumssystem geändert.

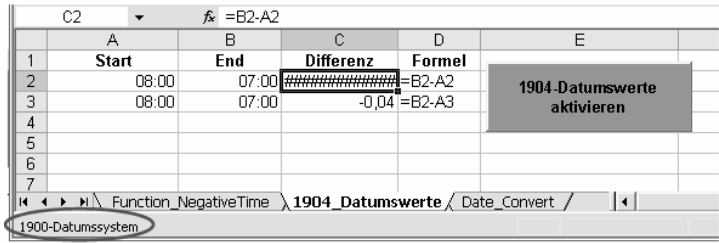


Abbildung 78: 1904-Datumswerte deaktiviert

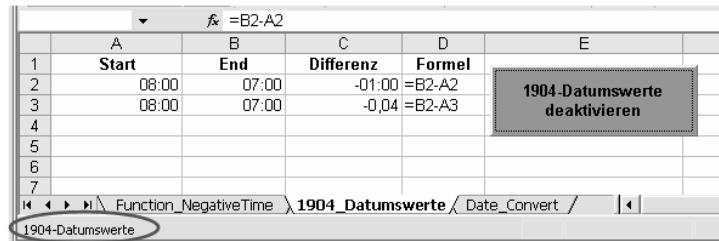


Abbildung 79: 1904-Datumswerte aktiviert

```
'=====
' Auf CD   Buchdaten\Beispiele\03_Datum_Zeit\
' Dateiname 03_07_MinusTime.xls
```

```
' Tabelle      1904_Datumswerte
' Ereignis     Tabelle2(1904_Datumswerte)
' =====

Private Sub ToggleButton1_Click()
    ActiveWorkbook.Date1904 = Not ActiveWorkbook.Date1904

    ToggleButton1.Caption = _
        "1904-Datumswerte" & vbCrLf & _
        IIf(Application.ActiveWorkbook.Date1904, _
            "deaktivieren", "aktivieren")
    InfoDateSystem
End Sub
```

Die Syntax zur Aktivierung des 1904-Datumssystems lautet:

```
ActiveWorkbook.Date1904 = True
```

Verwenden Sie die Eigenschaft `False` für die Deaktivierung des 1904-Datumssystems, also die Einstellung des 1900-Datumssystems.

93 Anzeige des eingestellten Datumssystems in Statuszeile

Woran erkennen Sie nun, nach welchem Datumssystem Excel die aktive Arbeitsmappe berechnet? Entweder subtrahieren Sie zwei Uhrzeiten und erkennen an den Gitterzäunen bzw. an der korrekten Ausgabe der negativen Zeitdifferenz die aktuelle Einstellung, oder Sie klicken sich wieder durch das Menü EXTRAS | OPTIONEN Registerkarte BERECHNUNG und schauen im Abschnitt ARBEITSMAPPEN-OPTIONEN nach, ob in dem Kontrollkästchen 1904-DATUMSWERTE ein Häkchen steht oder nicht.

Eine elegantere Möglichkeit bietet die Anzeige des Datumssystems in der STATUSZEILE von Excel. Üblicherweise wird diese von Excel verwaltet, um Informationen anzuzeigen. Der Standardtext lautet BEREIT.

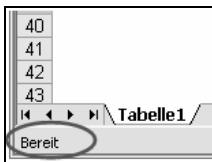


Abbildung 80: Standardtext der Statuszeile

In der Beispieldatei zeigen wir Ihnen das jeweils eingestellte Datumssystem als Hinweistext in der Statuszeile an. Das Auslesen der Einstellung sowie das Schreiben in die Statuszeile erfolgt über die Prozedur `InfoDateSystem`. Wenn die Statusleiste wieder von Microsoft Excel gesteuert werden soll, setzen Sie die `StatusBar`-Eigenschaft mit der Prozedur `StatusBarReset` wieder zurück.

```
' =====
' Auf CD      Buchdaten\Beispiele\03_Datum_Zeit\
' Dateiname   03_07_MinusTime.xls
```

```
' Tabelle      1904_Datumswerte
' Modul       mdl_03_StatusBar
' =====

Sub InfoDateSystem()
    Application.StatusBar = _
        IIf(Application.ActiveWorkbook.Date1904, _
            "1904-Datumswerte", "normales Datumssystem")
End Sub

Sub StatusBarReset()
    Application.StatusBar = False
End Sub
```

Die Anzeige des Datumssystems soll nur in dieser Beispieldatei geschehen. Daraus folgt, dass Sie Excel mitteilen müssen, wann die gewünschte Information in der Statuszeile angezeigt werden soll, und wann Excel wieder die Kontrolle übernimmt.

Dazu verwenden Sie ein paar der Workbook-Ereignis-Prozeduren, welche in der Kategorie »Ereignisse« noch genauer beschrieben werden. Tritt eines der Ereignisse ein, wird die jeweils angegebene Prozedur aufgerufen und ausgeführt.

Die Umstellung des Datumssystems selbst löst kein Ereignis aus. Um dennoch jederzeit das eingestellte Datumssystem in der Statuszeile ausgeben zu können, verwenden wir das Workbook_SheetSelectionChange-Ereignis. Jede Änderung der Zellmarkierung auf jedem Tabellenblatt der Datei löst dieses Ereignis aus. Dadurch wird sichergestellt, dass die ordnungsgemäße Anzeige des Datumssystems zeitnah nach der Umstellung erfolgt.

```
' =====
' Auf CD      Buchdaten\Beispiele\03_Datum_Zeit\
' Dateiname   03_07_MinusTime.xls
' Ereignis    DieseArbeitsmappe
' =====

Private Sub Workbook_Activate()
    InfoDateSystem
End Sub

Private Sub Workbook_SheetSelectionChange _
    (ByVal Sh As Object, ByVal Target As Range)
    InfoDateSystem
End Sub

Private Sub Workbook_Deactivate()
    StatusBarReset
End Sub
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

94 Eingabe von Minuszeiten

Die Eingabe einer negativen Uhrzeit erreichen Sie in beiden Datumssystemen durch den Zelleintrag

```
"-10:00"
```

Folgende Prozedur schreibt eine negative Zeit im Textformat in die Zelle E10.

```
' =====
' Auf CD      Buchdaten\Beispiele\03_Datum_Zeit\
' Dateiname   03_07_MinusTime.xls
' Tabelle     Function_NegativeTime
' Modul       mdl_04_WriteMinusTime
' =====

Sub WriteMinusTime()
    Range("E10") = "-10:00"
End Sub
```

Hinweis

Beachten Sie bitte, dass es sich bei diesem Zellwert um einen Wert im Textformat handelt. Eine Berechnung ist mit diesem Zelleninhalt nicht möglich. Diese Möglichkeit der negativen Zeiteingabe in eine Zelle ist nur dafür gedacht, Minuszeiten in einer Zelle *anzuzeigen*.

95 1904-Datumswerte mit VBA kopieren

In der Zelle A12 der Beispieldatei, Tabelle 1904_DATUMSWERTE, steht das Datum 01.01.2003. Dieses Datum wird bei der Umstellung auf das 1904-Datumssystem auf den 02.01.2007 geändert, was exakt 1462 Tagen entspricht.

Angenommen, die 1904-DATUMSWERTE sind aktiviert und Sie wollen das Datum aus A12 mit einem einfachen Befehl in eine weitere Zelle A13 schreiben. Dazu können Sie folgende Anweisung geben:

```
Range("A13") = Range("A12")
```

In der folgenden Prozedur stellen wir zunächst die 1904-Datumswerte ein und ändern passend dazu die Beschriftung der Umschaltfläche, zur Umstellung des Datumssystems. Im Anschluss daran wird der Zelle A13 der Wert der Zelle A12 zugewiesen. Das Ergebnis sehen Sie anschließend im Tabellenblatt und in einem Meldungsfenster:

```
' =====
' Auf CD      Buchdaten\Beispiele\03_Datum_Zeit\
' Dateiname   03_07_MinusTime.xls
' Tabelle     1904_Datumswerte
' Modul       mdl_05_CopyDate
' =====

Sub CopyDateFalse()
    ActiveWorkbook.Date1904 = True
```

```
ActiveSheet.CommandButton1.Caption = _
    "1904-Datumswerte" & vbCr & "deaktivieren"

Range("A13") = Range("A12")

MsgBox "A12= " & Range("A12") & _
    vbCr & _
    "A13= " & Range("A13") _
    , , "CopyDateFalse"

End Sub
```

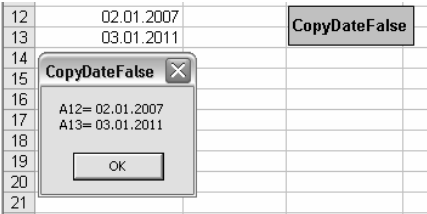


Abbildung 81: Falscher Datentransfer

Sie sehen, dass Excel nicht das Datum der Zelle A12 eins zu eins nach A13 transferiert hat, sondern entsprechend der aktivierten 1904-Datumswerte vier Jahre und einen Tag dazuaddiert hat.

Wie kann das passieren? Ist das ein bereits mehrfach beschriebener Bug?

Nein, es ist kein Bug, die Erklärung dazu ist einfach: Die Default-Einstellung des Range-Objektes ist Value. Wird einem Range-Objekt keine Eigenschaft zugewiesen, gibt es den Wert des angegebenen Bezugs zurück. Leider rechnet Excel in unserem Fall dem gefundenen Wert aufgrund der aktivierten 1904-Datumswerte vier Jahre und einen Tag hinzu.

Die Lösung des Problems liegt auf der Hand: schreiben Sie obige Prozedur mit expliziter Zuweisung der gewünschten Value-Eigenschaft für das Range-Objekt.

```
Range("A13").Value = Range("A12").Value
```

Dadurch wird der tatsächliche Wert des Range korrekt von der Zelle A12 in die Zelle A13 transferiert:

```
'=====
' Auf CD      Buchdaten\Beispiele\03_Datum_Zeit\
' Dateiname   03_07_MinusTime.xls
' Tabelle     1904_Datumswerte
' Modul       mdl_05_CopyDate
'=====

Sub CopyDateTrue()
    ActiveWorkbook.Date1904 = True

    ActiveSheet.CommandButton1.Caption = _
        "1904-Datumswerte" & vbCr & "deaktivieren"
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

```
Range("A13").Value = Range("A12").Value
```

```
MsgBox "A12= " & Range("A12").Value & _  
vbCr & _  
"A13= " & Range("A13").Value _  
, , "CopyDateTrue"
```

```
End Sub
```

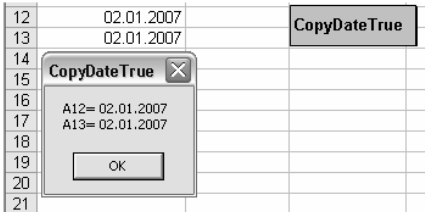


Abbildung 82: Richtiger Datentransfer

96 Datumswerte aus anderem Datumssystem importieren

Für den Fall, dass Sie Daten aus anderen Arbeitsmappen importiert haben, welche in einem anderen als dem aktuellen Datumssystem gespeichert waren, konvertieren Sie die Datumswerte entweder von den 1900- zu den 1904-Datumswerten oder umgekehrt.

Haben Sie bereits einen umfangreichen Bestand an Datumswerten in Ihrer Tabelle und ändern Sie dann das Datumssystem auf 1904-Datumswerte, können Sie mit der ersten Prozedur `ConvertInto1904` alle vorhandenen Daten wieder berichtigen, also um vier Jahre und einen Tag zurückrechnen. Dieser Zeitraum entspricht einem Zahlenwert von 1462. Durch Verwendung der `SpecialCells`-Methode werden dabei nur reine Zahlenwerte neu berechnet. Ergebnisse, welche aus Formeln resultieren, bleiben dabei unberücksichtigt. Die Prozedur `ConvertFrom1904` zählt zu jedem Datumswert im benutzten Bereich 1462 Tage hinzu, um die entstandene Differenz durch die Umstellung des Datumssystems auszugleichen.

Die folgenden Prozeduren erledigen im benutzten Bereich des aktiven Tabellenblattes die Konvertierungsarbeit.

```
'=====
' Auf CD      Buchdaten\Beispiele\03_Datum_Zeit\
' Dateiname   03_07_MinusTime.xls
' Tabelle     Date_Convert
' Modul       mdl_06_DateConvert
'=====

' Feste Datumswerte Konvertieren _
'   von normalem Datumssystem zu 1904-Datumswerten

Sub ConvertInto1904()
    Dim rngArea As Range
    Dim rngCell As Range
```

```
On Error Resume Next
' Verweis auf Zahlenzellen im aktiven Tabellenblatt
Set rngArea = ActiveSheet.UsedRange _
                .SpecialCells(xlCellTypeConstants, 1)
' Verlasse Prozedur, wenn keine Zahlenzellen vorhanden sind
If Err.Number > 0 Then Exit Sub

' Fehlerroutine zurücksetzen
On Error GoTo 0

' Ereignisse deaktivieren
Application.EnableEvents = False

For Each rngCell In rngArea
    If IsDate(rngCell.Value) Then
        ' Datumswerte zurückrechnen
        rngCell.Value = rngCell.Value - 1462
    End If
Next

' Ereignisse aktivieren
Application.EnableEvents = True
End Sub

' Feste Datumswerte Konvertieren _
  von 1904-Datumswerten zu normalem Datumssystem

Sub ConvertFrom1904()
    Dim rngArea As Range
    Dim rngCell As Range

    On Error Resume Next
    Set rngArea = ActiveSheet.UsedRange _
                .SpecialCells(xlCellTypeConstants, 1)
    If Err.Number > 0 Then Exit Sub

    On Error GoTo 0
    Application.EnableEvents = False
    For Each rngCell In rngArea
        If IsDate(rngCell.Value) Then
            rngCell.Value = rngCell.Value + 1462
        End If
    Next
    Application.EnableEvents = True
End Sub
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

97 Datums-Funktionen bei 1904-Datumswerten

Formeln mit Datumsberechnungen reagieren anders als fixe Datumsangaben bei der Umstellung des Datumssystems. Die Formel =HEUTE() liefert in beiden Systemen den gleichen Wert: das Datum der Systemeinstellung. Dieses Verhalten betrifft auch weitere Funktionen der Kategorie DATUM & ZEIT im Funktions-Assistenten, wie zum Beispiel die Jetzt- und die Datum-Funktion.

Wurde das Datumssystem auf 1904-Datumswerte oder umgekehrt umgestellt, gleichen die folgenden Prozeduren die Zeitdifferenz wieder aus.

```
'=====
' Auf CD      Buchdaten\Beispiele\03_Datum_Zeit\
' Dateiname  03_07_MinusTime.xls
' Tabelle    Date_Convert
' Modul      mdl_06_DateConvert
'=====
```

```
' Formel-Datumswerte Konvertieren _
'   von normalem Datumssystem zu 1904-Datumswerten
```

```
Sub ConvertDateFormularInto1904()
    Dim rngArea As Range
    Dim rngCell As Range

    On Error Resume Next
    ' Verweis auf Formelzellen im aktiven Tabellenblatt
    Set rngArea = ActiveSheet.UsedRange _
        .SpecialCells(xlCellTypeFormulas, 23)
    If Err.Number > 0 Then Exit Sub

    On Error GoTo 0
    Application.EnableEvents = False
    For Each rngCell In rngArea
        If IsDate(rngCell.Value) And _
            Right(rngCell.Formula, 5) = "+1462" Then

            rngCell.Formula = _
                Left(rngCell.Formula, Len(rngCell.Formula) - 5)
        End If
    Next
    Application.EnableEvents = True
End Sub
```

```
' Formel-Datumswerte Konvertieren _
'   von 1904-Datumswerten zu normalem Datumssystem
```

```
Sub ConvertDateFormularFrom1904()
    Dim rngArea As Range
    Dim rngCell As Range

    On Error Resume Next
    ' Verweis auf Formelzellen im aktiven Tabellenblatt
    Set rngArea = ActiveSheet.UsedRange _
```

```
                .SpecialCells(xlCellTypeFormulas, 23)
If Err.Number > 0 Then Exit Sub

On Error GoTo 0
Application.EnableEvents = False
For Each rngCell In rngArea
    If IsDate(rngCell.Value) Then
        rngCell.Formula = rngCell.Formula & "+1462"
    End If
Next
Application.EnableEvents = True
End Sub
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

98 Berechnung von Datumsdifferenzen

Wenn Sie die Anzahl der Tage, Monate oder Jahre zwischen zwei Datumsangaben mit einer komfortablen Möglichkeit berechnen möchten, können Sie die Tabellen-Funktion DATEDIF verwenden. Diese Funktion liefert die Anzahl der Tage, Monate oder Jahre zwischen einem Ausgangs- und einem Enddatum.

Obwohl diese Funktion aus Kompatibilitätsgründen mit Lotus 1-2-3 seit einigen Jahren in Microsoft Excel zur Verfügung gestellt wird, wurde sie erstmals in der in Excel integrierten Online-Hilfe, Version Excel 2000, beschrieben. In Excel 2002 wird diese Funktion wiederum nicht mehr dokumentiert. Aus diesem Grund geben wir hier noch einmal eine genauere Beschreibung.

Die DATEDIF-Funktion hat die folgende Syntax:

```
=DATEDIF(Ausgangsdatum;Enddatum;Einheit)
```

Der dritte Parameter der Funktion ist der Informationstyp, welcher die Einheit angibt, die zurückgegeben werden soll. Diese Einheit wird über die englische Kurzschreibweise vorgenommen. Die Online-Hilfe in Excel 2000 nennt hier fälschlicherweise die deutschen Abkürzungen. Zum Beispiel wird dort J für Jahre angegeben, richtig ist aber Y für Year.

Datumsangaben können als Textzeichenfolgen in Anführungszeichen, als fortlaufende Zahlen oder als Ergebnis anderer Formeln oder Funktionen eingegeben werden.

Ausgangsdatum ist das erste Datum oder Startdatum des Zeitraums.

Enddatum ist ein Datum, welches das letzte Datum oder Enddatum des Zeitraums angibt.

Die folgenden Einheiten stehen Ihnen dabei zur Verfügung. Die Ergebnisse wurden aus nachstehender Formel berechnet:

```
=DATEDIF("15.06.2000";"01.02.2003";"Einheit")
```

Einheit	Ergebnis	Rückgabewert
»Y«	2	Die Anzahl der vollständigen Jahre im Zeitraum
»M«	31	Die Anzahl der vollständigen Monate im Zeitraum
»D«	961	Die Anzahl der Tage im Zeitraum
»MD«	17	Die Differenz zwischen den Tagen in Anfangsdatum und Enddatum Die Monate und Jahre der Datumsangaben werden ignoriert
»YM«	7	Die Differenz zwischen den Monaten in Anfangsdatum und Enddatum Die Tage und Jahre der Datumsangaben werden ignoriert.
»YD«	231	Die Differenz zwischen den Tagen in Anfangsdatum und Enddatum Die Jahre der Datumsangaben werden ignoriert

Tabelle 19: Informationstypen der DATEDIF-Tabellenfunktion

Es sind auch Kombinationen der DATEDIF-Funktion innerhalb einer Formel möglich:

```
=DATEDIF("15.06.2000";"01.02.2003";"Y") & " Jahre und " &  
DATEDIF("15.06.2000";"01.02.2003";"YM") & " Monate" & " und "&  
DATEDIF("15.06.2000";"01.02.2003";"MD") & " Tage"
```

ergibt 2 Jahre und 7 Monate und 17 Tage.

Bei der Berechnung durch die DATEDIF-Funktion werden nur ganze Jahre bzw. Monate berücksichtigt. Der Zeitraum zwischen dem 15.01.2003 und 14.02.2003 ergibt 0 Monate. Erst wenn der Tag des Ausgangsdatums erreicht oder überschritten wird, zählt die DATEDIF-Funktion einen Monat weiter.

```
=DATEDIF("15.01.2003";"15.02.2003";"M") ' ergibt 1 Monat
```

Ausgangsdatum	Enddatum	Einheit	Differenz
15.01.2002	14.01.2003	Y	0 Jahre
15.01.2002	15.01.2003	Y	1 Jahr
15.01.2003	14.02.2003	YM	0 Monate
15.01.2003	15.02.2003	YM	1 Monat
15.01.2003	14.01.2003	D	#ZAHL!
15.01.2003	15.01.2003	D	0 Tage
15.01.2003	16.01.2003	D	1 Tag

Tabelle 20: DATEDIF-Berechnungen

In der Tabelle 20 wird ersichtlich, dass das Enddatum größer als das Ausgangsdatum sein muss, da die Formel ansonsten eine Fehlermeldung als Ergebnis liefert.

99 Zeitdifferenzen mit VBA DateDiff

Mit der DateDiff-Funktion, welche in VBA zur Verfügung steht, können Sie ermitteln, wie viele Zeitintervalle sich zwischen zwei Daten oder Zeiten befinden.

Die Syntax zu dieser Funktion lautet:

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

Die ersten drei Parameter interval, date1 und date2 sind vergleichbar mit den Parametern der bereits genannten DATEDIF-Tabellenfunktion. Der vierte Parameter firstdayofweek ist optional und erwartet als Argument eine Zahl, die den ersten Tag der Woche angibt. Ist dieser Wert nicht angegeben, so wird Sonntag angenommen. Der letzte Parameter firstweekofyear ist ebenfalls optional und erwartet eine Zahl, welche die erste Woche des Jahres angibt. Ist dieser Wert nicht angegeben, wird die Woche mit dem 1. Januar als die erste Woche angenommen.

Folgende Einstellungen sind für das Argument interval möglich:

Einstellung	Beschreibung
yyyy	Jahr
q	Quartal
m	Monat
y	Tag des Jahres
d	Tag
w	Wochentag
ww	Woche
h	Stunde
n	Minute
s	Sekunde

Tabelle 21: Interval-Argumente der DateDiff-Funktion

Achtung

Die Annahme, dass die DateDiff-Funktion in der VBA-Programmierung das Gegenstück zu der Tabellenfunktion DATEDIF darstellt, ist falsch.

Mit der VBA-Funktion DateDiff wird ermittelt, wie viele Zeitintervalle sich zwischen zwei Daten befinden. Die Zeitdifferenz zwischen dem 31.01. zum 01.02. gilt deswegen als ganzer Monat, während die viel längere Zeitdifferenz vom 1.1. zum 31.1. keinen Monat ergibt.

DateDiff bestimmt die Monatsdifferenz nicht im Sinne einer Tagezählung, sondern durch Vergleich der Monats-Zahlen. Also ist ergänzend der Tag hinzuzunehmen.

Vergleichen wir die Ergebnisse der beiden Funktionen DATEDIF und DateDiff, wird deutlich, dass diese beiden Funktionen sich nicht nur durch ein f mehr oder weniger unterscheiden, sondern auch unterschiedlich rechnen. In der Beispieldatei finden Sie die Testdaten und Ergebnisse unse-

Grund-lagen

Allge-mein

Datu-Zeit

tungen

Steuer-elemen

Befehl-leisten

Objekt

Dia-gramm

Ereig-nisse

User-Form

Web/Mail

Extern

Ge-mischtes

Specia

rer Berechnungen. Ausgangssituation sind zwei Datumswerte in A2 und B2 der Tabelle DATEDIFCONTRADATEDIFF. In der Zelle A2 steht das Datum 15.06.2000, in B2 der 01.01.2003. Die Ergebnisse der Excel-Funktion wurden mit oben bereits erwähnten Formeln berechnet. Die Werte der VBA-Funktion haben wir mit dem nachstehenden Makro in die einzelnen Zellen geschrieben.

```
' =====
' Auf CD      Buchdaten\Beispiele\03_Datum_Zeit\
' Dateiname   03_08_DATEDIFandDateDiff.xls
' Tabelle     DATEDIFcontraDateDiff
' Modul       mdl_01_VBADateDiff
' =====
```

```
Sub VBADateDiff()
    Dim DateBegin As Date
    Dim DateEnd As Date
    Dim intI As Integer
    Dim varInterval As Variant

    DateBegin = Range("A2").Value
    DateEnd = Range("B2").Value
    varInterval = Array("yyyy", "m", "d")

    For intI = 0 To 2
        Cells(intI + 2, 5) = _
            DateDiff(varInterval(intI), DateBegin, DateEnd)

        Cells(intI + 2, 6) = varInterval(intI)
    Next
End Sub
```

C2		=DATEDIF(A2,B2;"Y")				
	A	B	C	D	E	F
1	Begin	End	Excel	Einheit	VBA	Interval
2	15.06.2000	01.02.2003	2	"Y"	3	yyyy
3			31	"M"	32	m
4			961	"D"	961	d
5			17	"MD"	keine Berechnung möglich	
6			7	"YM"	keine Berechnung möglich	
7			231	"YD"	keine Berechnung möglich	

Abbildung 83: Vergleichsergebnisse DATEDIF / DateDiff

Per Makro sieht ein Vergleich der beiden Funktionen, welche sich von der Schreibweise her nur durch ein zusätzliches *f* unterscheiden, folgendermaßen aus:

```
' =====
' Auf CD      Buchdaten\Beispiele\03_Datum_Zeit\
' Dateiname   03_08_DATEDIFandDateDiff.xls
```

```
' Tabelle    DATEDIFcontraDateDiff
' Modul      mdl_02_DateSpanYear
' =====

Sub DateSpanYear()
    Dim intVBA As Integer

    'Ergebnis aus VBA-Funktion an Variable übergeben
    intVBA = DateDiff( _
        Interval:="yyyy", _
        Date1:=Range("A2").Value, _
        Date2:=Range("B2").Value)

    With Range("IV1")
        'Tabellenfunktion temporär in Zelle schreiben
        .Formula = "=DateDif(A2,B2,""Y"")"

        MsgBox "Im Zeitraum vom " & Range("A2").Value & _
            " bis " & Range("B2").Value _
            & " liegen" & _
            vbNewLine & _
            "Excel: " & .Value & " Jahre" & _
            vbNewLine & _
            "VBA:  " & intVBA & " Jahre", , _
            "DATEDIF contra DateDiff"

        'Temporäre Berechnung löschen
        .ClearContents
    End With
End Sub
```

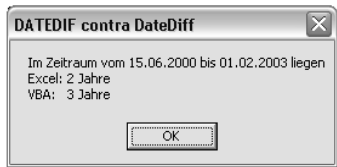


Abbildung 84: Vergleichsberechnung

100 VBA-Ersatz-Prozedur für die DateDif-Tabellenfunktion

Die Meldung aus Abbildung 84 hat uns überzeugt, eine Ersatzlösung für die VBA-Funktion DateDiff zu finden, um auch unter VBA das gleiche Ergebnis wie das der Tabellenfunktion DATEDIF zu erhalten.

Eine einfache Möglichkeit, eine Tabellenfunktion unter VBA zu verwenden, besteht über das Application.WorksheetFunction-Objekt. Folgende Syntax wird für die Tabellenfunktion =SUMME("A1:A5") unter VBA angewandt.

```
Range("D2") = Application.WorksheetFunction.Sum(Range("A1:A5"))
```

Grund-lagen

Allge-mein

Datu-Zeit

tungen

Steuer-elemen

Befehl-leisten

Objekt

Dia-gramm

Ereig-nisse

User-Form

Web/Mail

Extern

Ge-misch-tes

Specia

Genauso wie die Tabellenfunktion DATEDIF nicht in der Hilfe dokumentiert wird, erscheint sie auch nicht in der INTELLISENSE-Liste des Application.WorksheetFunction-Objektes (siehe Abbildung 85) und steht somit unter VBA nicht zur Verfügung.

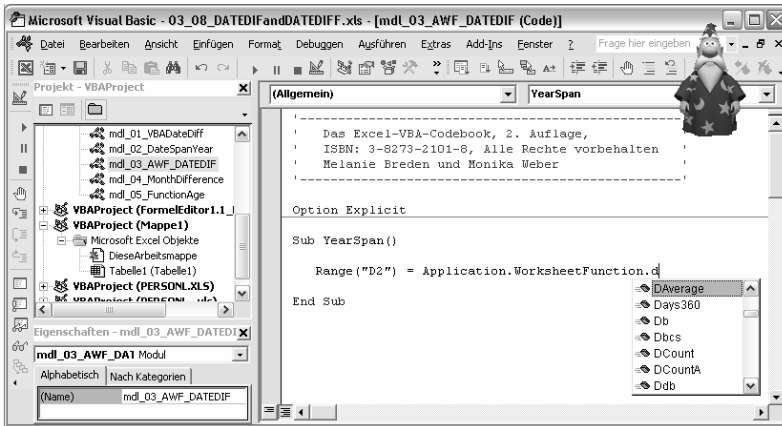


Abbildung 85: Methoden des Application.WorksheetFunction-Objekts über die INTELLISENSE-Auflistung

Da uns die Tabellenfunktion DATEDIF in VBA nicht zur Verfügung steht, bedienen Sie sich eines einfachen Tricks: Schreiben Sie in einer Prozedur die Tabellenfunktion DATEDIF zur Berechnung einer Zeitdifferenz in die gewünschte Zelle und wandeln Sie dann das Formelergebnis in einen festen Wert um.

In der Tabelle AWF.DATEDIF der Beispieldatei wurden die nachfolgend aufgeführten Prozeduren angewandt.

```
'=====
' Auf CD      Buchdaten\Beispiele\03_Datum_Zeit\
' Dateiname   03_08_DATEDIFandDateDiff.xls
' Tabelle     AWF.DATEDIF
' Modul       mdl_03_AWF_DATEDIF
'=====
```

```
Sub YearSpan()
    With Range("D2")
        .Formula = "=DateDif(A2,B2,""Y"")"
        .Value = .Value
    End With
End Sub
```

```
Sub MonthSpan()
    With Range("D3")
        .Formula = "=DateDif(A2,B2,""M"")"
        .Value = .Value
    End With
End Sub
```



```
Sub DaySpan()  
    With Range("D4")  
        .Formula = "=DateDif(A2,B2,""D"")"  
        .Value = .Value  
    End With  
End Sub  
  
Sub MonthDaySpan()  
    With Range("D5")  
        .Formula = "=DateDif(A2,B2,""MD"")"  
        .Value = .Value  
    End With  
End Sub  
  
Sub YearMonthSpan()  
    With Range("D6")  
        .Formula = "=DateDif(A2,B2,""YM"")"  
        .Value = .Value  
    End With  
End Sub  
  
Sub YearDaySpan()  
    With Range("D7")  
        .Formula = "=DateDif(A2,B2,""YD"")"  
        .Value = .Value  
    End With  
End Sub
```

Es ist ebenso möglich, ohne die DATEDIF-Funktion eine Zeitdifferenz zu berechnen. Das folgende Makro zeigt dies am Beispiel einer Monatsdifferenz.

```
' =====  
' Auf CD      Buchdaten\Beispiele\03_Datum_Zeit\  
' Dateiname  03_08_DATEDIFandDateDiff.xls  
' Tabelle    AWF.DATEDIF  
' Modul      mdl_04_MonthDifference  
' =====  
  
Sub MonthDifference()  
    Dim datDate1 As Date  
    Dim datDate2 As Date  
    Dim bytTemp As Byte  
    Dim intDiffMonth As Integer  
  
    datDate1 = Worksheets("AWF.DATEDIF").Range("A2").Value  
    datDate2 = Worksheets("AWF.DATEDIF").Range("B2").Value  
  
    If Day(datDate2) < Day(datDate1) Then bytTemp = 1  
  
    intDiffMonth = (Year(datDate2) - Year(datDate1)) * 12 _  
        + Month(datDate2) - Month(datDate1) - bytTemp
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

```
MsgBox "Zwischen dem " & Range("A2").Value & _
      " und dem " & Range("B2").Value & _
      vbNewLine & _
      "liegen " & intDiffMonth & " Monate.", , _
      "Monatsberechnung ohne DATEDIF-Funktion"
```

```
End Sub
```

101 Funktion, um Lebensalter zu berechnen

Im Zusammenhang mit der `Datediff`-Funktion möchten wir Ihnen noch eine benutzerdefinierte Funktion vorstellen, die Ihnen bei der Altersberechnung behilflich sein kann.

Schreiben Sie die Funktion im VBA-Editor in ein Modul. In der Funktion wird berücksichtigt, ob der Geburtstag im aktuellen Jahr bereits stattgefunden hat oder nicht. In der Beispieldatei Tabelle `FUNCTION_AGE` wird in Zelle C8 das Geburtsdatum eingegeben. Der Aufruf in einer Zelle erfolgt über die Tabellen-Formeln `=Age(Bezug)` oder `=Age(Bezug; [Deadline])`.

Interessant an der Funktion ist der zweite optionale Parameter, `Deadline`, den Sie auf Wunsch verwenden können, aber nicht müssen. Standardmäßig wird das Alter auf Basis des aktuellen Datums berechnet. Über den Parameter `Deadline` können Sie auch ein bestimmtes Enddatum angeben. Folgende Formel liefert das Alter der Person am 01.01.2003 (Geburtsdatum in Zelle C8):

```
=Age(C8;DATWERT("01.01.2003"))
```

Ist der Parameter `Bezug` leer oder enthält er keinen gültigen Datumswert, dann wird ein Leerstring an die `Age`-Funktion zurückgegeben und die Funktion beendet. Mit der `IsMissing`-Funktion wird in der Prozedur überprüft, ob der optionale Parameter `Deadline` in der Formel angegeben wurde. Fehlt dieses Argument, wird das aktuelle Datum aus der Systemeinstellung als Endzeit zur Berechnung herangezogen.

Die Berechnung des Lebensalters soll taggenau erfolgen. Unter Berücksichtigung des Jahres der Endzeit wird kontrolliert, ob das Startdatum größer oder kleiner als das Enddatum ist. Dazu verwenden wir die `DateSerial`-Funktion. In dieser Funktion wird aus dem Tag und Monat des Startdatums und dem Jahr des Enddatums ein Vergleichsdatum erzeugt:

```
DateSerial(Year(Deadline), Month(Birthday), Day(Birthday))
```

Mit diesem Vergleichsdatum lässt sich nun überprüfen, ob der angegebene Geburtstag in dem Jahr des Enddatums schon stattgefunden hat oder ob er noch vor dem Enddatum liegt. Ist das Vergleichsdatum größer als das Enddatum, dann liegt der Geburtstag noch vor uns. Dementsprechend wird von dem Ergebnis der `Datediff`-Funktion ein Jahr abgezogen. Wenn das Vergleichsdatum kleiner als das Enddatum ist, wurde der Geburtstag in dem Jahr schon gefeiert. In diesem Fall stimmt die Berechnung der VBA-Funktion `Datediff`.

In unserer Beispieldatei wird die `Age`-Funktion in der Zelle C12 über die Formel `=Age(C8)` aufgerufen. Sobald Sie ein Datum in die Zelle C8 eingeben, erfolgt die Berechnung mit anschließender Ausgabe des Ergebnisses in der Zelle C12.

```
'=====
' Auf CD      Buchdaten\Beispiele\03_Datum_Zeit\
' Dateiname  03_08_DATEDIFandDateDiff.xls
' Tabelle    Function_Age
' Modul      mdl_05_FunctionAge
'=====
```

```
Function Age(Birthday, Optional Deadline)
    If IsEmpty(Birthday) Or Not IsDate(Birthday) Then
        Age = ""
        Exit Function
    End If

    If IsMissingDeadline) Then Deadline = Date

    If DateSerial(Year(Deadline), Month(Birthday), _
        Day(Birthday)) > Deadline Then
        Age = DateDiff("yyy", Birthday, Deadline) - 1
    Else
        Age = DateDiff("yyy", Birthday, Deadline)
    End If
End Function
```

102 Stoppuhr mit Excel

In diesem Rezept zeigen wir Ihnen, wie man in Excel eine Stoppuhr erstellt. Bei Klick auf die Schaltfläche START wird in einem Feld die abgelaufene Zeit im Sekundentakt bis zum nächsten Klicken auf die Schaltfläche STOP mitgeschnitten.



Abbildung 86: Stoppuhr

In dem Code zu obigen Beispiel bedienen wir uns der OnTime-Methode.

Hinweis

Eine ausführliche Beschreibung mit anschaulichen Beispielen zu der OnTime-Methode finden Sie in der Kategorie »Ereignisse«.

Die Darstellung der Zeit erfolgt hier nicht in einer Zelle, sondern wird in einem Bezeichnungsfeld LABEL angezeigt. In der Beispieldatei haben wir noch einige kleinere Spielereien integriert. Über das Workbook_Open-Ereignis sowie bei Aktivierung der START/STOP-Schaltfläche werden die verwendeten Steuerelemente unterschiedlich formatiert und beschriftet.

Grundlagen
Allgemein

Datum
Zeit

Funktionen

Steuerelemente

Befehle
leisten

Objekt

Diagramm

Ereignisse

User-Form

Web/Mail

Extern

Gemischtes

Spezial

```
'=====
' Auf CD      Buchdaten\Beispiele\03_Datum_Zeit\
' Dateiname   03_09_StopWatch.xls
' Ereignis    DieseArbeitsmappe
'=====

Private Sub Workbook_Open()
    ' Fenstergröße
    ActiveWindow.WindowState = xlNormal
    With ActiveWindow
        .Width = 195
        .Height = 197
    End With

    With Worksheets(1)
        .Label1.Visible = False
        .CmbStartStop.Caption = "K l i c k m i c h"
        .CmbStartStop.BackColor = &HC000&
    End With

    ' Bei eingestelltem Blattschutz kann im Tabellenblatt
    ' nichts markiert werden
    Worksheets(1).EnableSelection = xlNoSelection
End Sub
```

Für den START bzw. STOP der Zeitmessung verwenden wir für beide Funktionen eine Befehls-schaltfläche `COMMANDBUTTON`. Beide Elemente, `LABEL` und `COMMANDBUTTON`, wurden aus der `STEUERELEMENT-TOOLBOX`-Symbolleiste eingefügt.

```
'=====
' Auf CD      Buchdaten\Beispiele\03_Datum_Zeit\
' Dateiname   03_09_StopWatch.xls
' Ereignis    Tabelle1(StopWatch)
'=====

Private Sub CmbStartStop_Click()
    If CmbStartStop.Caption = "S t a r t" Or _
        CmbStartStop.Caption = "K l i c k m i c h" Then
        DateTimeNow = Time
        CmbStartStop.Caption = "S t o p"
        CmbStartStop.BackColor = &H80FF&
        Label1.Visible = True

        ' Aufruf des Makros zur Zeitanzeige
        ShowTime
    Else
        Label1.Caption = Format(Time - DateTimeNow, "hh:mm:ss")
        CmbStartStop.Caption = "S t a r t"
        CmbStartStop.BackColor = &HC000&
```

```
        ' Aufruf des Makros, um die Zeitanzeige zu beenden
        StopTime
    End If
End Sub
```

Über die `OnTime`-Methode wird die Prozedur `ShowTime` im Sekundentakt erneut aufgerufen, um die angezeigte Zeit im Bezeichnungsfeld `Label1` um eine Sekunde zu erhöhen.

```
'=====
' Auf CD      Buchdaten\Beispiele\03_Datum_Zeit\
' Dateiname  03_09_StopWatch.xls
' Modul      mdl_01_StopWatch
'=====

Public TimeNow As Date
Public ET      As Variant

Sub ShowTime()
    ThisWorkbook.Sheets(1).Label1.Caption = _
        Format(Time - TimeNow, "hh:mm:ss")

    ET = Now + TimeValue("00:00:01")

    Application.OnTime ET, "ShowTime"
End Sub
```

Im folgenden Makro wird die `OnTime`-Einstellung aus der vorherigen Prozedur gelöscht.

```
'=====
' Auf CD      Buchdaten\Beispiele\03_Datum_Zeit\
' Dateiname  03_09_StopWatch.xls
' Modul      mdl_01_StopWatch
'=====

Sub StopTime()
    On Error Resume Next
    Application.OnTime EarliestTime:=ET, Procedure:="ShowTime", _
        Schedule:=False
End Sub
```

Beim Beenden der Beispieldatei wird die Prozedur `StopTime` im `Workbook_BeforeClose`-Ereignis aufgerufen. Sie setzt die Aktualisierung der Zeitanzeige zurück. Die `Saved`-Eigenschaft wird auf `True` gesetzt. Dadurch wird die Datei, ohne sie zu speichern und ohne den Anwender zum Speichern aufzufordern, geschlossen.

```
'=====
' Auf CD      Buchdaten\Beispiele\03_Datum_Zeit\
' Dateiname  03_09_StopWatch.xls
' Ereignis   DieseArbeitsmappe
'=====
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

```
Private Sub Workbook_BeforeClose(Cancel As Boolean)
    StopTime
    ThisWorkbook.Saved = True
End Sub
```

103 Excel kann warten

Bei der Verarbeitung eines aufgerufenen Makros werden die einzelnen Codezeilen einer Prozedur schrittweise, Zeile für Zeile, abgearbeitet. Mit der `Wait`-Methode wird das aktuell ausgeführte Makro bis zu einem angegebenen Zeitpunkt angehalten.

Im ersten Beispiel wird über die `Wait`-Methode der Zeitpunkt festgelegt, zu dem das Makro fortgesetzt und dann das Meldungsfenster angezeigt wird.

```
'=====
' Auf CD      Buchdaten\Beispiele\03_Datum_Zeit\
' Dateiname   03_10_Wait.xls
' Modul       mdl_01_Wait
'=====

Sub MessageIn10Seconds()
    If Application.Wait(Now + TimeValue("0:00:10")) Then
        MsgBox "10 Sekunden sind abgelaufen"
    End If
End Sub
```

104 Signaltöne erzeugen

Im nächsten Beispiel wollen wir mit der `Beep`-Anweisung fünf aufeinander folgende Signaltöne über den Lautsprecher des Computers abspielen.

Mit folgendem Code werden zwar in einer `For...Next`-Anweisung fünf Signaltöne erzeugt, aber beim Makrodurchlauf ist nur ein Ton zu hören. Die Begründung liegt darin, dass der Schleifendurchlauf zu schnell ist, um fünf getrennte Töne hörbar wahrzunehmen.

```
'=====
' Auf CD      Buchdaten\Beispiele\03_Datum_Zeit\
' Dateiname   03_10_Wait.xls
' Modul       mdl_01_Wait
'=====

Sub OneBeep()
    Dim bytCounter As Byte

    For bytCounter = 1 To 5
        Beep
    Next bytCounter
End Sub
```

Lassen Sie Excel jeweils nach jeder Tonausgabe eine Sekunde lang warten, bevor die Schleife erneut durchlaufen wird. Dadurch werden fünf einzelne Signaltöne hörbar erzeugt.

```
'=====
' Auf CD      Buchdaten\Beispiele\03_Datum_Zeit\
' Dateiname   03_10_Wait.xls
' Modul       mdl_01_Wait
'=====

Sub FiveBeeps()
    Dim bytCounter As Byte

    For bytCounter = 1 To 5
        Beep
        Application.Wait (Now + TimeValue("0:00:01"))
    Next bytCounter
End Sub
```

105 Töne bei aktuellem Datum ausgeben

Im folgenden Beispiel wird die aktive Zelle auf das heutige Datum überprüft. Bei Übereinstimmung werden drei Signaltöne ausgegeben. Hat die aktive Zelle nicht den heutigen Datumswert, erfolgt eine Fehlermeldung. Es kann vorkommen, dass Excel zu Beginn der Schleife zeitlich ins Stolpern gerät und die Tonausgabe ungleichmäßig erfolgt. Diesen Effekt haben Sie eventuell im vorherigen Beispiel feststellen können. Aus diesem Grund halten wir das Makro eine Sekunde lang an, bevor die Schleife begonnen wird. Dadurch sind die drei angeforderten Töne in gleichmäßigen Abständen zu hören.

```
'=====
' Auf CD      Buchdaten\Beispiele\03_Datum_Zeit\
' Dateiname   03_10_Wait.xls
' Modul       mdl_01_Wait
'=====

Sub BeepMsgBox()
    Dim bytCounter As Byte

    If ActiveCell.Value = Date Then
        Application.Wait (Now + TimeValue("0:00:01"))

        For bytCounter = 1 To 3
            Beep
            Application.Wait (Now + TimeValue("0:00:01"))
        Next bytCounter

    Else
        MsgBox "Die ausgewählte Zelle enthält nicht den " & Date
    End If
End Sub
```

Grundlagen

Allgemein

Datum und Zeit

Funktionen

Steuerungselemente

Befehle

Objekte

Diagramme

Ereignisse

User-Formen

Web/Mail

Extern

Gemischtes

Spezial

106 Meldung zeitbegrenzt anzeigen

Im folgenden Beispiel wird ein Text für eine definierte Zeitspanne in einer `UserForm` angezeigt. In unserem Beispiel enthält das Formular keine Schaltflächen. Das `UserForm` zeigt nur einen Text sowie die verbleibende Zeit bis zur selbstständigen Schließung der Meldung an. Das `UserForm` wird nach zehn Sekunden automatisch geschlossen.

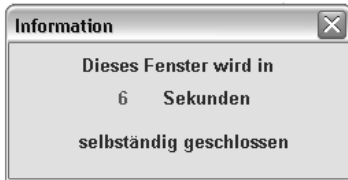


Abbildung 87: Selbstschließende Meldung

Der Aufruf des `UserForm` erfolgt über eine Schaltfläche auf dem Tabellenblatt.

```
'=====
' Auf CD      Buchdaten\Beispiele\03_Datum_Zeit\
' Dateiname   03_10_Wait.xls
' Modul       mdl_02_CallInfo
'=====
```

```
Sub CallInfo()
    frm_01_Info.Show
End Sub
```

Beim Aufruf der `UserForm` wird im `UserForm_Activate`-Ereignis über die `Wait`-Methode innerhalb einer `For...Next`-Anweisung die Zeit bis zur Schließung der Form abgearbeitet. Da die Schleife zehn Mal durchlaufen wird, beträgt die Anzeigendauer dementsprechend zehn Sekunden.

Hinweis

Die `Wait`-Methode unterbricht alle Aktivitäten von Microsoft Excel und verhindert unter Umständen auch das Ausführen sonstiger Vorgänge auf dem Computer. Hintergrundprozesse, wie Drucken und Neuberechnung, werden jedoch fortgesetzt.

Durch Verwendung der `DoEvents`-Funktion wird die Steuerung an das Betriebssystem übergeben, damit andere Ereignisse verarbeitet werden können.

Durch die `DoEvents`-Funktion wird sichergestellt, dass die Aufschrift des Bezeichnungsfeldes `Label2` bei jedem Schleifendurchlauf aktualisiert wird.

Für die Zeitanzeige im Tabellenblatt verwenden wir ein `BEZEICHNUNGSFELD` aus der Formular-WERKZEUGSAMMLUNG. In dessen `Caption`-Eigenschaft wird bei jedem Schleifendurchlauf die verbleibende Restlaufzeit der Schleife geschrieben. Direkt im Anschluss an die Schleife wird das Formular wieder geschlossen.


```
' =====  
' Auf CD      Buchdaten\Beispiele\03_Datum_Zeit\  
' Dateiname  03_10_Wait.xls  
' Formular   frm_01_Info  
' =====
```

```
Private Sub UserForm_Activate()  
    Dim intI As Integer  
  
    For intI = 1 To 10  
        DoEvents  
        Label2.Caption = 11 - intI  
        Application.Wait Now + TimeValue("00:00:01")  
    Next  
  
    Unload Me  
End Sub
```

Tipp

Nähere Informationen und Beispiele zu den Steuerelementen des Formulars finden Sie in der Kategorie »UserForm«

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

Auswertungen

In dieser Kategorie finden Sie Beispiele, die Ihnen zeigen, wie Sie VBA anwenden können, um Ihre Daten auszuwerten. Sie werden erfahren, wie Sie Bezüge speichern können, wie Sie Tabellenblätter vergleichen und wie Formeln ermittelt werden können. Es werden zudem die Themen Autofilter, Spezialfilter, Gültigkeit, Sortieren und Pivot-Tabellen behandelt.

In Excel können bedingte Formatierungen verwendet werden. Excel stellt allerdings maximal drei Bedingungen zur Verfügung. Sie erfahren in dieser Kategorie, wie per VBA diese Einschränkung umgangen werden kann.

107 Bezüge archivieren

Durch das Löschen eines Tabellenblattes verlieren Formeln, welche sich auf dieses Blatt bezogen haben, ihren Bezug. In den betroffenen Zellen erscheint die Fehlermeldung `#Bezug!`.

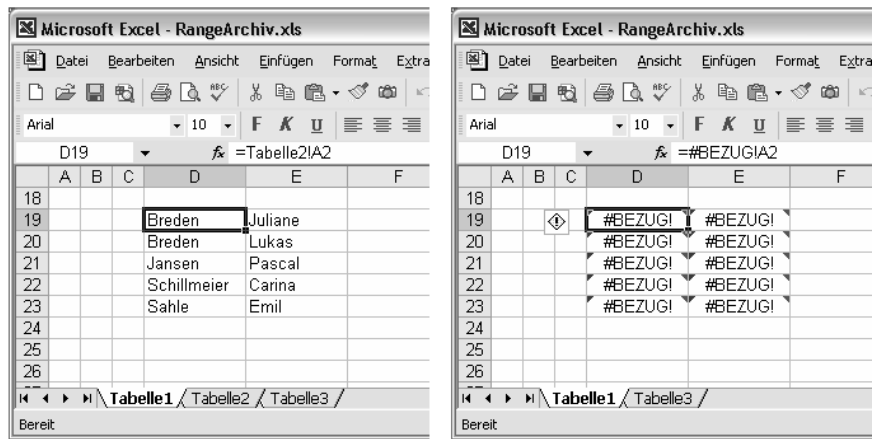


Abbildung 88: Formelzellen mit verlorenem Bezug

Hinweis Es ist nicht möglich, das Löschen eines Tabellenblattes rückgängig zu machen. Dabei spielt es keine Rolle, ob ein Blatt manuell oder über VBA gelöscht wurde. Sie können lediglich die Arbeitsmappe nach dem Löschvorgang ohne zu speichern schließen. Beim erneuten Öffnen sind die Bezüge wieder vorhanden, da das Tabellenblatt in der Datei vorhanden und somit der Bezug hergestellt ist.

Aber es gibt einen Weg, wie Excel sich die Formeln merken kann: Sie werden einfach in den Kommentartext der Zelle geschrieben. Im folgenden Beispiel schreiben wir anfangs die Anweisung `On Error Resume Next`. Dadurch werden mögliche Fehler in der Prozedur zunächst ignoriert. Im Laufe der Prozedur werden evtl. aufgetretene Fehler über das `Err`-Objekt verarbeitet. Anschließend geben Sie in eine `InputBox` den Namen des zu löschenden Tabellenblattes ein. Diese Eingabe wird in der Objekt-Variable `DelSheet` gespeichert. In einer `If`-Abfrage wird überprüft, ob die

Grundlagen

Allgemein

Datum
Zeit

tungen

Steuerelemente

Befehle
leisten

Objekte

Diagramm

Ereignisse

User-Form

Web/
Mail

Extern

Gemischtes

Spezial

Variable `DelSheet` gefüllt ist. Diese kann nur gefüllt sein, wenn der eingegebene Name einem vorhandenen Tabellenblattnamen der Arbeitsmappe entspricht. Die Prozedur zeigt eine Fehlermeldung, wenn das Blatt nicht vorhanden ist, und wird anschließend mit `Exit Sub` beendet.

Wurde der Blattname identifiziert, werden in einer `For ... Each`-Schleife alle Tabellenblätter der aktiven Arbeitsmappe durchlaufen, welche nicht den Namen des zu löschenden Blattes tragen. Bezüge können nur in Formeln enthalten sein. Dementsprechend werden in einer weiteren `For ... Each`-Schleife alle Zellen des Schleifenblattes überprüft, welche eine Formel enthalten. Dies erreichen wir über das `Type-Argument xlCellTypeFormulas` der `SpecialCells`-Methode. Unmittelbar vor dem Schleifenaufruf werden mit der Anweisung `Err.Clear` mögliche Fehler aus dem Speicher gelöscht. Wenn ein Tabellenblatt keine Formeln enthält, würde die Anweisung

```
For Each rngCell In Ws.UsedRange.SpecialCells(xlCellTypeFormulas)
```

einen Fehler verursachen. Dieser Fehler wird durch die anfängliche `On Error Resume Next`-Anweisung jedoch erst einmal übergangen. Über die Abfrage `If Err.Number = 0` wird ermittelt, ob der Schleifenaufruf einen Fehler verursacht hat. Dieser wäre darauf zurückzuführen, dass in dem zu durchsuchenden Tabellenblatt keine Formeln enthalten sind. Wurde kein Fehler festgestellt, beträgt der Wert von `Err.Number` 0.

In der nun folgenden `InStr`-Funktion wird ermittelt, ob der Name des Tabellenblattes `DelSheet` in der Formel der gerade überprüften Zelle enthalten ist. Die `InStr`-Funktion gibt die Position des ersten Auftretens der gesuchten Zeichenfolge wieder. Beträgt dieser Wert größer 0, wird in der nächsten Prozedurzeile ein möglicherweise vorhandener Kommentar gelöscht. Excel würde einen Fehler ausgeben, wenn Sie versuchen, zu einem vorhandenen Kommentar einen weiteren hinzuzufügen. Hierzu verwenden wir eine `If`-Abfrage:

```
If Not rngCell.Comment Is Nothing Then rngCell.Comment.Delete
```

Die Syntax `rngCell.FormulaLocal` gibt die Formel der Zelle unter Verwendung von A1-Bezügen in der Sprache des Benutzers zurück. Diese wird dem Kommentar als Text zugewiesen. Über die `AutoSize`-Eigenschaft wird die Größe des Kommentar-Rahmens automatisch dem Formeltext angepasst und der Kommentar im Tabellenblatt ausgeblendet.

Tipp

Weitere Informationen und Beispiele zu Kommentaren finden Sie in der Kategorie »Einstieg«

Nachdem alle Formelzellen in allen Tabellenblättern der Arbeitsmappe abgearbeitet wurden, erstellen wir vorsichtshalber eine Sicherheitskopie. Das Tabellenblatt `DelSheet` wird kopiert und erhält den Namen des zu löschenden Blattes mit dem Zusatzhinweis & `"_copy"`. Die Warnmeldung, bevor ein Tabellenblatt gelöscht wird, umgehen Sie mit der `DisplayAlerts`-Eigenschaft. Diese wird vor dem Löschen des Blattes auf `False` gesetzt und im Anschluss daran wieder auf `True`. Zu guter Letzt wird noch das anfänglich aktive Tabellenblatt selektiert und die Bildschirmaktualisierung wieder eingeschaltet.

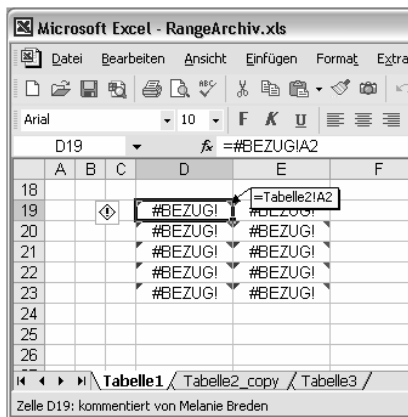


Abbildung 89: Formeln in Kommentaren archivieren

```
' =====
' Auf CD      Buchdaten\Beispiele\04_Auswertungen\
' Dateiname  04_01_RangeArchiv.xls
' Tabelle     Tabelle1
' Modul      mdl_01_RememberFormula
' =====

Public Sub RememberFormula()
    Dim MySheet As Worksheet
    Dim DelSheet As Worksheet
    Dim Ws As Worksheet
    Dim rngCell As Range

    ' aktives Tabellenblatt an Variable übergeben
    Set MySheet = ActiveSheet

    ' Fehlermeldungen in Prozedur zunächst ignorieren und
    ' später verarbeiten
    On Error Resume Next

    ' Eingabe, welches Tabellenblatt gelöscht werden soll
    Set DelSheet = Worksheets(Application.InputBox _
        ("Welche Tabelle soll gelöscht werden?", _
        "Bezüge festhalten", "Tabelle2", , , , 2))

    ' Prüfung, ob Tabellenblatt zur Eingabe existiert
    If DelSheet Is Nothing Then
        MsgBox "Der eingegebene Tabellenname existiert nicht."
        Exit Sub
    End If

    ' Bildschirmaktualisierung deaktivieren
    Application.ScreenUpdating = False

    ' Schleife über alle Tabellenblätter der aktiven Arbeitsmappe
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

```

For Each Ws In ActiveWorkbook.Worksheets
    ' Schleife weiter durchlaufen, wenn Blatt Ws
    ' nicht das zu löschende Blatt ist
    If Ws.Name <> DelSheet.Name Then
        ' mögliche Fehler werden aus dem Speicher gelöscht
        Err.Clear
        ' Schleife über alle Zellen welche Formeln enthalten
        For Each rngCell In Ws.UsedRange. _
            SpecialCells(xlCellTypeFormulas)
            ' Prüfen, ob Fehler verursacht wurde
            If Err.Number = 0 Then
                ' Prüfen, ob Blattname in Formel enthalten ist
                If InStr(rngCell.Formula, DelSheet.Name) > 0 Then
                    ' evtl. vorhandenen Kommentar löschen
                    If Not rngCell.Comment Is Nothing Then _
                        rngCell.Comment.Delete
                    ' Kommentar hinzufügen
                    With rngCell.AddComment
                        ' Formel in Kommentar schreiben
                        .Text rngCell.FormulaLocal
                        ' automatische Größe des Kommentars
                        .Shape.TextFrame.AutoSize = True
                        ' Kommentar ausblenden
                        .Visible = False
                    End With
                End If
            End If
        Next
    End If
Next

' Sicherheitsblatt erstellen
DelSheet.Copy After:=DelSheet
ActiveSheet.Name = DelSheet.Name & "_copy"

' Blatt löschen
Application.DisplayAlerts = False
DelSheet.Delete
Application.DisplayAlerts = True

' Ausgangsblatt selektieren
MySheet.Select
' Bildschirmaktualisierung aktivieren
Application.ScreenUpdating = True
End Sub

```

108 Bezug-Fehler reparieren

In unserer vorherigen Beispiel-Prozedur `RememberFormula` wurde von einem gelöschten Tabellenblatt vorab eine Sicherheitskopie erstellt. Diese trägt den Blattnamen des ursprünglichen Bezugsblattes mit dem Zusatz »_copy«. Sie können den Zusatz löschen, um den ursprünglichen Tabellenblattnamen zu erhalten. Excel ist dadurch dennoch nicht in der Lage, die entstandenen

fehlenden Bezüge in den Formeln wieder herzustellen. Jeder fehlende Bezug wird in Formeln durch den Ausdruck `#Bezug!` ersetzt. Dementsprechend kann keine eindeutige Zuordnung zum ursprünglichen Bezug hergestellt werden.

Wie Sie Formeln in Kommentaren hinterlegen, haben Sie im vorherigen Beispiel gesehen. In der nachfolgenden Prozedur `ResetFormulas` schreiben wir die Formeln aus den Kommentaren in die dazugehörigen Zellen zurück. Diese geben dann als Ergebnis die Berechnung aus der Formel wieder. Die abgearbeiteten Kommentare werden gelöscht. Der dazugehörige Code ähnelt dem vorherigen zum Speichern von Formeln in Kommentaren. Aufgrund dessen beschreiben wir nachfolgend nur die Unterschiede der beiden Prozeduren.

Bei jedem Zurückschreiben der Formeln in eine Zelle wird eine Neuberechnung der gesamten Arbeitsmappe ausgelöst. Das verlangsamt die Laufzeit des Makros unnötig. Bevor die erste `For...Each`-Schleife aufgerufen wird, stellen wir deshalb für die Laufzeit der Prozedur die Berechnungsart über die `Calculation`-Eigenschaft `xlManual` auf `manuell`. Am Ende der Prozedur übergeben wir die Handhabung der Neuberechnung mit `.Calculation = xlCalculationAutomatic` wieder an Excel. Es erfolgt dann sogleich eine Neuberechnung.

Es sollen alle Formeln, welche in Kommentaren hinterlegt sind und dem gewünschten Tabellenblattnamen aus der `InputBox` entsprechen, in Ihre Zellen zurückgeschrieben werden. Folglich werden in der `For...Each`-Schleife alle Zellen, welche Kommentare enthalten, überprüft. Dies erreichen wir über das `Type`-Argument `xlCellTypeComments` der `SpecialCells`-Methode. Die Formeln aus den Kommentar-Texten werden mit `rngCell.Formula = .Comment.Text` in die einzelnen Zellen geschrieben.

```
'=====
' Auf CD      Buchdaten\Beispiele\04_Auswertungen\
' Dateiname  04_01_RangeArchiv.xls
' Tabelle     Tabelle1
' Modul      mdl_02_ResetFormulas
'=====
```

```
Public Sub ResetFormulas()
    Dim DelSheet As Worksheet
    Dim Ws       As Worksheet
    Dim rngCell  As Range

    On Error Resume Next
    Set DelSheet = _
        Worksheets(Application.InputBox _
        ("Welcher Tabellenname soll aktualisiert werden?", _
        "Bezüge wiederherstellen", "Tabelle2", , , , 2))
    If DelSheet Is Nothing Then
        MsgBox "Die eingegebene Tabelle existiert nicht." & _
            vbCrLf & vbCrLf & _
            "Entfernen Sie evtl. den Zusatz ""_copy"" " & _
            "des Tabellenblattes."
    End If

    Exit Sub

    With Application
        .ScreenUpdating = False
        .Calculation = xlManual
```

Grund-
lagenAllge-
meinDatum
Zeit

tionen

Steuer
elemenBefehl
leisten

Objekt

Dia-
grammEreig-
nisseUser-
FormWeb/
Mail

Extern

Ge-
misch
tes

Specia

```

End With

For Each Ws In ActiveWorkbook.Worksheets
    If Ws.Name <> DelSheet.Name Then
        Err.Clear
        ' Schleife über alle Zellen welche Kommentare enthalten
        For Each rngCell In Ws.UsedRange. _
            SpecialCells(xlCellTypeComments)
            If Err.Number = 0 Then
                ' Prüfen, ob Blattname in Kommentar-Formel
                ' enthalten ist
                If InStr(rngCell.Comment.Text, DelSheet.Name) > 0 _
                    Then
                    With rngCell
                        ' Formel aus Kommentar in Zelle schreiben
                        .Formula = .Comment.Text
                        .Comment.Delete
                    End With
                End If
            End If
        Next
    End If
Next

With Application
    .ScreenUpdating = True
    .Calculation = xlCalculationAutomatic
End With
End Sub

```

109 Tabellenblätter vergleichen (Fehlerausgabe in einem separaten Tabellenblatt)

Wir möchten Ihnen zwei verschiedene Rezepte zum Vergleich von Tabellenblättern vorstellen. Bei beiden Rezepten werden zwei Tabellenblätter miteinander verglichen. Ziel ist es, zu ermitteln, in welchen Zellen Abweichungen bestehen.

In diesem Rezept soll die Ausgabe der ungleichen Daten in einem neuen Tabellenblatt erfolgen. Es werden in dem Tabellenblatt sowohl die Zelladressen als auch die ungleichen Daten beider Tabellenblätter ausgegeben. Die Daten werden in die Spalten A-D geschrieben. Die Zelladressen werden zudem mit Hyperlinks versehen, so dass bequem per Mausklick in die Zelle mit ungleichem Inhalt gesprungen werden kann.

Direkt nach der Variablendeklaration wird die Bildschirmaktualisierung `ScreenUpdating` ausgeschaltet. Damit verhindern wir ein Flackern des Bildschirms während des Ablaufs der Prozedur. Zudem wird die Geschwindigkeit erhöht. Warnmeldungen, wie sie beim Löschen eines Tabellenblattes angezeigt werden, werden über die `DisplayAlerts`-Eigenschaft deaktiviert. Am Ende der Prozedur werden beide Eigenschaften wieder aktiviert.

In einem zweiten Schritt wird geprüft, ob bereits ein Tabellenblatt mit dem Namen `ERRORREPORT` in der Mappe enthalten ist. Dies könnte der Fall sein, wenn die Prozedur bereits einmal ausgeführt wurde. Das Blatt wird gelöscht und direkt im Anschluss neu erstellt.

	A	B	C	D	E
1	Tabelle1		Tabelle2		
2	A33	Monika	A33	Monika	
3	B2	Breen	B2	Breden	
4	C19	07.05.1972	C19	07.05.1973	
5	D14	Autho	D14	Author	
6	D25	501	D25	500	
7	F25	8801	F25	8800	
8					

Abbildung 90: Ausgabe der unterschiedlichen Zellen

Danach erfolgt die Referenzierung Set der Objekte, die wiederholt verwendet werden. Am Ende der Prozedur werden die Objekte wieder freigegeben mit Set = Nothing.

Nach der Referenzierung wird in If-Entscheidungen ermittelt, welches Tabellenblatt mehr Zeilen oder Spalten umfasst. Dies um sicherzustellen, dass jede Zelle beider Blätter auf Unstimmigkeiten überprüft wird, auch wenn ein Tabellenblatt mehr Zeilen oder Spalten enthalten sollte als das andere. Der jeweils höhere Wert wird an die Variablen intMaxRow (Zeilen) und intMaxCol (Spalten) übergeben.

Vor dem Eintritt in die For...Next-Schleife setzen wir den Wert des Zählers i auf 1. Dieser Zähler wird verwendet, um im neuen Tabellenblatt ERRORREPORT die ungleichen Daten jeweils fortlaufend untereinander auszugeben. Vor dem Austritt aus der Schleife wird der Zähler jeweils um den Wert 1 erhöht.

Zwei ineinander verschachtelte For-Schleifen bilden den eigentlichen Kern der Prozedur, denn hier werden Zelle für Zelle die beiden Tabellenblätter verglichen. Sollte eine Unstimmigkeit gefunden werden (strCompWS1 <> strCompWS2), wird die With-Anweisung ausgeführt. In der With-Anweisung werden Hyperlinks und ungleiche Daten an die Spalten A-D des Tabellenblattes ERRORREPORT übergeben (siehe Abbildung 90).

Nachdem alle Daten verglichen worden sind und die beiden For-Schleifen verlassen wurden, werden die Spaltenüberschriften eingefügt. In der Spalte A und B stehen die Werte des ersten Tabellenblattes. In der Spaltenüberschrift soll der Name dieses Tabellenblattes stehen. Die Schrift wird im Stil »Fett« Bold formatiert. Wir weisen eine grüne Hintergrundfarbe Interior.ColorIndex zu. Der Inhalt der Zellen wird zentriert (HorizontalAlignment = xlCenter). Dasselbe erfolgt mit den Spalten C und D, mit dem Unterschied, dass wir hier eine gelbe Hintergrundfarbe zuweisen. Die ersten beiden und die zweiten beiden Zellen werden miteinander verbunden (Merge).

Wenn keine Unterschiede auf den beiden Tabellenblättern gefunden werden, wird eine entsprechende Meldung MsgBox auf dem Bildschirm angezeigt. Das Tabellenblatt ERRORREPORT wird in diesem Falle nicht benötigt und kann gelöscht werden.

```
' =====  
' Auf CD      Buchdaten\Beispiele\04_Auswertungen\  
' Dateiname  04_02_CompareTwoSheets.xls  
' Modul      mdl_01_CompareTwoSheets  
' =====
```

```
Sub CompareTwoSheets()  
    Dim ws As Worksheet  
    Dim ws1 As Worksheet, ws2 As Worksheet  
    Dim wsErrors As Worksheet
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

```

Dim objws1Row As Object, objws1Col As Object
Dim objws2Row As Object, objws2Col As Object
Dim intMaxRow As Integer, intMaxCol As Integer
Dim intCol As Integer, intRow As Integer
Dim strCompWS1 As String, strCompWS2 As String
Dim i As Integer

' Bildschirmaktualisierung und Warnungen ausschalten
With Application
    .ScreenUpdating = False
    .DisplayAlerts = False
End With

' Falls Tabellenblätter mit dem Namen "ErrorReport" oder
' "Adressen" vorhanden sind, diese löschen
For Each ws In Worksheets
    If ws.Name = "ErrorReport" Then
        ws.Delete
    End If
Next ws

' Tabellenblatt "ErrorReport" für die Auswertung einfügen
Worksheets.Add Before:=Worksheets(1)
Worksheets(1).Name = "ErrorReport"

' Referenzieren
Set wsErrors = Worksheets(1)
Set ws1 = Worksheets(2)
Set ws2 = Worksheets(3)

Set objws1Row = ws1.UsedRange.Rows
Set objws1Col = ws1.UsedRange.Columns
Set objws2Row = ws2.UsedRange.Rows
Set objws2Col = ws2.UsedRange.Columns

' Maximale Zeilenzahl ermitteln
If objws1Row.Count > objws2Row.Count Then
    intMaxRow = objws1Row.Count
Else
    intMaxRow = objws2Row.Count
End If

' Maximale Spaltenzahl ermitteln
If objws1Col.Count > objws2Col.Count Then
    intMaxCol = objws1Col.Count
Else
    intMaxCol = objws2Col.Count
End If

i = 1
' Jede Zelle der beiden Tabellenblätter vergleichen
For intCol = 1 To intMaxCol
    For intRow = 1 To intMaxRow

```

```
strCompWS1 = ws1.Cells(intRow, intCol)
strCompWS2 = ws2.Cells(intRow, intCol)

If strCompWS1 <> strCompWS2 Then

    ' Adressen und Werte von fehlerhaften Zellen
    ' in Tabellenblatt "ErrorReport" übertragen
    With wsErrors
        ' Hyperlinks und Werte des ersten Tabellenblattes
        .Hyperlinks.Add _
            Anchor:=.Cells(i, 1), _
            Address="", _
            SubAddress="" & ws1.Name & "!" & _
                ws1.Cells(intRow, intCol). _
                    Address(False, False), _
                    TextToDisplay:=Cells(intRow, intCol). _
                        Address(False, False)
        .Cells(i, 2) = strCompWS1

        ' Hyperlinks und Werte des zweiten Tabellenblattes
        .Hyperlinks.Add _
            Anchor:=.Cells(i, 3), _
            Address="", _
            SubAddress="" & ws2.Name & "!" & _
                ws1.Cells(intRow, intCol). _
                    Address(False, False), _
                    TextToDisplay:=Cells(intRow, intCol). _
                        Address(False, False)
        .Cells(i, 4) = strCompWS2
    End With
    i = i + 1

End If

Next intRow
Next intCol

' Spaltenüberschriften einfügen und formatieren
With wsErrors
    .Rows(1).Insert
    With .Range("A1")
        .Value = ws1.Name
        .Font.Bold = True
        .Interior.ColorIndex = 4
        .HorizontalAlignment = xlCenter
    End With

    With .Range("C1")
        .Value = ws2.Name
        .Font.Bold = True
        .Interior.ColorIndex = 6
        .HorizontalAlignment = xlCenter
    End With
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

```

        .Range("A1:B1").Merge
        .Range("C1:D1").Merge
    End With

    ' Wenn keine Unstimmigkeiten gefunden werden, dann
    ' eine entsprechende Meldung ausgeben und Hilfsblatt entfernen
    If i = 1 Then
        MsgBox "Es wurden keine Unstimmigkeiten gefunden"
        wsErrors.Delete
    End If

    ' Bildschirmaktualisierung und Warnungen einschalten
    With Application
        .ScreenUpdating = True
        .DisplayAlerts = True
    End With

    ' Objekte wieder freigeben
    Set wsErrors = Nothing
    Set ws1 = Nothing
    Set ws2 = Nothing

    Set objws1Row = Nothing
    Set objws1Col = Nothing
    Set objws2Row = Nothing
    Set objws2Col = Nothing
End Sub

```

110 Tabellenblätter vergleichen (Fehlerausgabe in einer separaten Mappe)

Wie bereits im vorangegangenen Rezept, werden hier zwei Tabellenblätter miteinander verglichen. Diesmal soll die Originaldatei jedoch unberührt bleiben. Wir erzeugen eine Kopie der Mappe und kreisen dort die ungleichen Werte rot ein. Jeder Zelle, die eine Abweichung aufweist, wird zudem ein Kommentar hinzugefügt, der das Gegenstück des Inhalts der Zelle der anderen Tabelle reflektiert.

	A	B	C	D	E	F
1	Melanie	Bredon		Melanie	Bredon	
2	Monika	Weber		Monika	Weber	
3			500		200	-100
4			800		1100	1100
5	Melanie	Bredon		Melanie	Bredon	
6	Monika	Weber		Monika	Weber	
7			1100		1500	2600
8			1400		2100	3500
9	Melanie	Bredon		Melanie	Bredon	
10	Monika	Weber		Monika	Weber	
11			1700		2700	4400
12			2000		3300	5300
13	Melanie	Bredon		Melanie	Bredon	

Abbildung 91: Ungleiche Daten eingekreist und mit Kommentar versehen

Die Variablendeklaration und die Referenzierung verhalten sich wie im vorangegangenen Rezept. In einer With-Anweisung werden Bildschirmaktualisierung und Warnmeldungen deaktiviert. Am Ende der Prozedur wird beides wieder aktiviert.

An die Variablen strOriginalFile und strCopyFile werden die Dateinamen der Originaldatei und der Kopie übergeben. In einer For-Schleife wird geprüft, ob eine Datei mit dem Namen der Kopie offen ist. Dies könnte der Fall sein, wenn die Prozedur bereits einmal durchlaufen wurde. In diesem Fall wird die Datei geschlossen. Nach der For-Schleife wird eine Kopie der Originaldatei erstellt, damit die Originaldatei unberührt bleibt. Die Markierungen der ungleichen Stellen der beiden Tabellenblätter erfolgen in der Kopie.

Die If-Entscheidungen entsprechen dem vorangegangenen Rezept. In der darauf folgenden For-Schleife werden die ungleichen Zellen eingekreist. An dieser Stelle werden wir auf eine Beschreibung der Kreise verzichten, da sich die Kategorie »Objekte« eigens dem Thema Autoformen, Grafiken usw. zuwendet. In der For-Schleife wird zudem jeder ungleichen Zelle ein Kommentar eingefügt (AddComment). In der Kommentabox wird der ungleiche Wert der anderen Tabelle angezeigt.

```
' =====
' Auf CD      Buchdaten\Beispiele\04_Auswertungen\
' Dateiname  04_03_CompareTwoSheetsCircle.xls
' Modul      mdl_01_CompareTwoSheetsCircle
' =====

Sub CompareTwoSheetsCircle()
    Dim wb As Workbook
    Dim strOriginalFile As String, strCopyFile As String
    Dim ws1 As Worksheet, ws2 As Worksheet
    Dim objws1Row As Object, objws1Col As Object
    Dim objws2Row As Object, objws2Col As Object
    Dim intMaxRow As Integer, intMaxCol As Integer
    Dim intCol As Integer, intRow As Integer
    Dim strCompWS1 As String, strCompWS2 As String

    ' Referenzierung
    Set ws1 = Worksheets(1)
    Set ws2 = Worksheets(2)

    Set objws1Row = ws1.UsedRange.Rows
    Set objws1Col = ws1.UsedRange.Columns
    Set objws2Row = ws2.UsedRange.Rows
    Set objws2Col = ws2.UsedRange.Columns

    ' Bildschirmaktualisierung und Warnungen ausschalten
    With Application
        .ScreenUpdating = False
        .DisplayAlerts = False
    End With

    ' Pfad und Dateiangaben von Original und Kopie
    strOriginalFile = ThisWorkbook.FullName
    strCopyFile = "ErrorReport.xls"
```

Grundlagen
Allgemein
Datum und Zeit
Formeln
Diagramme
Objekte
Diagramm
Ereignisse
User-Form
Web/Mail
Extern
Gemischtes
Spezial


```
        .Fill.Visible = msoFalse
        .Line.ForeColor.SchemeColor = 10
    End With
    .Cells(intRow, intCol).AddComment strCompWS1
End With

End If
Next intRow
Next intCol

' Die Originaldatei wieder öffnen
Workbooks.Open strOriginalFile

' Kopie aktivieren
Workbooks(strCopyFile).Activate

' Bildschirmaktualisierung und Warnungen einschalten
With Application
    .ScreenUpdating = True
    .DisplayAlerts = True
End With

' Objekte wieder freigeben
Set ws1 = Nothing
Set ws2 = Nothing

Set objws1Row = Nothing
Set objws1Col = Nothing
Set objws2Row = Nothing
Set objws2Col = Nothing
End Sub
```

Mittels der folgenden Prozedur können Sie im aktiven Tabellenblatt alle roten Kreise entfernen und den Inhalt der Kommentare an die Zellen übergeben. Alle vorhandenen Kommentare werden danach ebenfalls gelöscht.

```
' =====
' Auf CD      Buchdaten\Beispiele\04_Auswertungen\
' Dateiname   04_03_CompareTwoSheetsCircle.xls
' Modul       mdl_02_TakeOverComments
' =====

Sub TakeOverComments()
    Dim shp As Shape
    Dim c As Range

    ' Rote Kreise entfernen
    For Each shp In ActiveSheet.Shapes
        If shp.Type = msoAutoShape Then
            shp.Delete
        End If
    Next shp
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

```

' Inhalt von Kommentaren übernehmen und Kommentare löschen
For Each c In ActiveSheet.UsedRange.Cells
    If c.Comment Is Nothing Then
    Else
        c.Value = c.Comment.Text
        c.Comment.Delete
    End If
Next c
End Sub

```

111 Formeln in einem Tabellenblatt ermitteln

Wenn ein Tabellenblatt viele Formeln enthält, kann das Ganze leicht unübersichtlich werden. Es wäre oftmals hilfreich, wenn man zu einem solchen Tabellenblatt die enthaltenen Formeln und Werte in einem separaten Tabellenblatt ausgeben könnte. Natürlich können Sie über das Menü EXTRAS | OPTIONEN, Registerkarte ANSICHT die FORMELN aktivieren. Der Nachteil dabei ist, dass nur *entweder* die Werte *oder* die Formeln angezeigt werden können.

Dieses Rezept ist so aufgebaut, dass die Prozedur von dem Tabellenblatt aus gestartet wird, dessen Formeln ausgegeben werden sollen. Für die Ausgabe wird ein neues Tabellenblatt mit dem Namen GETFORMULAS erzeugt. Damit wird verhindert, dass vorhandene Werte überschrieben werden.

In der Spalte A werden die Zellbezüge ausgegeben und in der Spalte B die zugehörigen Formeln. In den Spalten D und E werden die im Tabellenblatt vorhandenen Werte sowie deren Zellbezüge aufgelistet. In den Spalten G und H werden die nicht berechenbaren Werte, also der vorhandene Text ausgegeben sowie dessen Zellbezüge.

	A	B	C	D	E	F	G	H
1	Support							
2	B12	=SVERWEIS(A12;Produktel\$A\$4:\$B\$16;2)		B5	90	A1	SVerweis mit Vergleich	
3	E12	=SVERWEIS(C12;\$A\$4:\$D\$8;VERGLEICH(B12;\$A\$4:\$D\$4;0);0)		C5	100	B3	Honorar pro Lektion	
4	B13	=SVERWEIS(A13;Produktel\$A\$4:\$B\$16;2)		D5	90	A4	Kursleiter	
5	E13	=SVERWEIS(C13;\$A\$4:\$D\$8;VERGLEICH(B13;\$A\$4:\$D\$4;0);0)		B6	80	B4	Lehrgänge	
6	B14	=SVERWEIS(A14;Produktel\$A\$4:\$B\$16;2)		C6	95	C4	Firmenkurs	
7	E14	=SVERWEIS(C14;\$A\$4:\$D\$8;VERGLEICH(B14;\$A\$4:\$D\$4;0);0)		D6	85	D4	Tageskurs	
8	B15	=SVERWEIS(A15;Produktel\$A\$4:\$B\$16;2)		B7	85	A5	Weber	
9	E15	=SVERWEIS(C15;\$A\$4:\$D\$8;VERGLEICH(B15;\$A\$4:\$D\$4;0);0)		C7	95	A6	Breden	
10	B16	=SVERWEIS(A16;Produktel\$A\$4:\$B\$16;2)		D7	90	A7	Can	
11	E16	=SVERWEIS(C16;\$A\$4:\$D\$8;VERGLEICH(B16;\$A\$4:\$D\$4;0);0)		B8	95	A8	Umlas	
12	B17	=SVERWEIS(A17;Produktel\$A\$4:\$B\$16;2)		C8	105	A11	Produkt	
13	E17	=SVERWEIS(C17;\$A\$4:\$D\$8;VERGLEICH(B17;\$A\$4:\$D\$4;0);0)		D8	95	B11	Kategorie	
14				D12	4	C11	Kursleiter	
15				D13	8	D11	Lektionen à 50 Min.	
16				D14	2	F11	Honorar am Lektion	

Abbildung 92: Formeln, Werte und Text in einem separaten Tabellenblatt ausgeben

Nach der Variablendeklaration wird in einer For-Each-Schleife geprüft, ob bereits ein Tabellenblatt mit dem Namen GETFORMULAS in der Mappe vorhanden ist. Die könnte dann der Fall sein, wenn die Prozedur bereits einmal ausgeführt wurde. Falls ein solches Tabellenblatt vorhanden ist, wird es gelöscht. Nach der Schleife wird ein neues Tabellenblatt mit dem Namen GETFORMULAS eingefügt.

In der nächsten For-Schleife werden alle Zellen im benutzten Bereich durchlaufen. In einer If-Entscheidung wird geprüft, ob es sich beim Inhalt der Zelle um eine Formel `FormulaLocal`, einen Wert `IsNumeric` oder einen Text `Not IsNumeric` handelt. Entsprechend dem Ergebnis der Entscheidung werden Zelladresse und Zellinhalt in die entsprechenden Spalten im neuen Tabellenblatt geschrieben.

Nach der Schleife wird in die Zelle A1 der Name des Tabellenblattes geschrieben, dessen Werte ermittelt wurden. Die Zelle wird zudem mit einem Hyperlink versehen, damit per Klick darauf auf das Herkunftsblatt gewechselt werden kann.

```
' =====
' Auf CD      Buchdaten\Beispiele\04_Auswertungen\
' Dateiname  04_04_GetFormulas.xls
' Modul      mdl_01_GetFormulas
' =====

Sub SearchFormulas()
    Dim strActSht As String
    Dim ws As Worksheet
    Dim wsFormula As Worksheet
    Dim wsActSht As Worksheet
    Dim i As Integer
    Dim c As Range
    Dim strAdr As String

    strActSht = ActiveSheet.Name

    ' Allfällig vorhandenes Tabellenblatt "GetFormulas" entfernen
    For Each ws In Worksheets
        If ws.Name = "GetFormulas" Then
            Application.DisplayAlerts = False
            ws.Delete
            Application.DisplayAlerts = True
        End If
    Next ws

    ' Tabellenblatt "GetFormulas" neu erzeugen
    Worksheets.Add Before:=Worksheets(1)
    Worksheets(1).Name = "GetFormulas"

    ' Objekte referenzieren
    Set wsFormula = Worksheets("GetFormulas")
    Set wsActSht = Worksheets(strActSht)

    wsFormula.Range("A1").Value = strActSht

    i = 0

    ' Jede Zelle im benutzten Bereich prüfen
    For Each c In wsActSht.UsedRange
        strAdr = c.Address(False, False)
        With wsFormula

            ' Formeln ermitteln
            If c.HasFormula Then
                .Cells(Rows.Count, 1).End(xlUp). _
                    Offset(1, 0) = strAdr
                .Cells(Rows.Count, 2).End(xlUp). _
                    Offset(1, 0) = " " & c.FormulaLocal
            End If
        End With
    Next c
End Sub
```

Grundlagen

Allgemein

Datum
Zeit

Funktionen

Steuerelemente

Befehle
leisten

Objekte

Diagramme

Ereignisse

User-Formen

Web/Mail

Extern

Gemischtes

Spezial

```

        i = i + 1

        ' Zahlen ermitteln
    ElseIf IsNumeric(c) And c <> "" Then
        .Cells(Rows.Count, 4).End(xlUp). _
            Offset(1, 0) = strAdr
        .Cells(Rows.Count, 5).End(xlUp). _
            Offset(1, 0) = " " & c

        ' Text ermitteln
    ElseIf IsNumeric(c) = False And c <> "" Then
        .Cells(Rows.Count, 7).End(xlUp). _
            Offset(1, 0) = strAdr
        .Cells(Rows.Count, 8).End(xlUp). _
            Offset(1, 0) = " " & c
    End If

End With
Next c

' Tabellenblatt "GetFormulas" mit einem Hyperlink zurück
' zur Ursprungstabelle versehen
With wsFormula
    .Range("A1").Hyperlinks.Add _
        Anchor:=Selection, _
        Address:="", _
        SubAddress:="" & wsFormula.Range("A1").Value & "!A1", _
        TextToDisplay:=[A1].Value
    .Range("A:H").Columns.AutoFit
End With

' Referenzierte Objekte wieder freigeben
Set wsFormula = Nothing
Set wsActSht = Nothing
End Sub

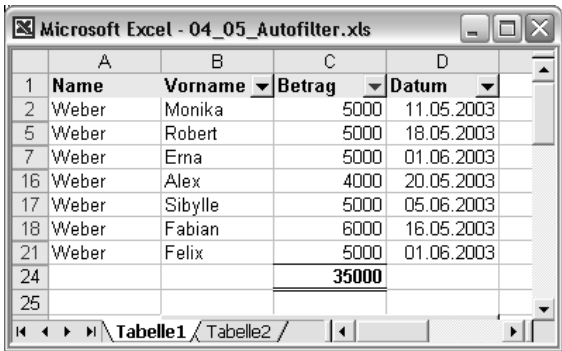
```

112 Einfacher Autofilter

Wenn Sie einen Autofilter mit VBA programmieren möchten, dann benutzen Sie am besten den Makro-Rekorder, zeichnen die einzelnen Schritte auf und passen dann die Prozedur an. Auf diese Weise ist es am einfachsten, die benötigten Objekte, Methoden und Eigenschaften zu erhalten. Den Autofilter finden Sie unter dem Menüpunkt DATEN | FILTER | AUTOFILTER.

Dass ein Autofilter gesetzt ist, ist an den Pfeilen zu erkennen, die neben den Überschriften angezeigt werden. Wenn Filterdaten ausgewählt wurden, dann sind die Filterpfeile blau, ansonsten schwarz.

In unserem Beispiel verwenden wir eine Tabelle mit vier Spalten (A bis D). In der Spalte A sollen nur Felder gefiltert werden, die den Namen »Weber« enthalten. Sie haben zudem die Möglichkeit, den Filterpfeil nur dieser einen Spalte per VBA auszublenden (`VisibleDropDown:=False`). Auf dem manuellen Wege ist es nicht möglich, einen einzelnen Filterpfeil auszublenden.



	A	B	C	D
1	Name	Vorname	Betrag	Datum
2	Weber	Monika	5000	11.05.2003
5	Weber	Robert	5000	18.05.2003
7	Weber	Erna	5000	01.06.2003
16	Weber	Alex	4000	20.05.2003
17	Weber	Sibylle	5000	05.06.2003
18	Weber	Fabian	6000	16.05.2003
21	Weber	Felix	5000	01.06.2003
24			35000	
25				

Abbildung 93: Gefilterte Daten

In der Spalte C werden nur Daten gefiltert, die einen Betrag größer als 3'000 *Criteria1* und kleiner als 5'000 *Criteria2* aufweisen. Es werden somit für die Spalte C zwei Kriterien verwendet. Um die Werte einzugrenzen, verwenden wir als Operator die Konstante `x1And`.

Ein Autofilter löscht keine Datensätze, sondern blendet sie lediglich aus. Im letzten Teil unserer Prozedur werden die gefilterten Daten ins zweite Tabellenblatt kopiert. Damit sind in der Kopie *nur* die gefilterten Daten enthalten. Vor dem Kopiervorgang werden allfällig enthaltene Daten im zweiten Tabellenblatt entfernt `Clear`. Diesen Vorgang können Sie auch manuell erreichen, indem Sie die Funktionen Kopieren, Einfügen und Löschen verwenden.

```
'=====
' Auf CD      Buchdaten\Beispiele\04_Auswertungen\
' Dateiname  04_05_Autofilter.xls
' Modul      mdl_01_SimpleAutofilter
'=====

Sub SimpleAutofilter()
    Dim ws1 As Worksheet
    Dim ws2 As Worksheet

    Set ws1 = Worksheets(1)
    Set ws2 = Worksheets(2)

    With ws1.Range("A1")
        ' Autofilter auf erstes Filterfeld
        ' DropDown-Pfeil ausblenden
        .AutoFilter Field:=1, _
            Criteria1:="Weber", _
            VisibleDropDown:=False

        ' Autofilter auf drittes Filterfeld
        .AutoFilter Field:=3, _
            Criteria1:=">3000", _
            Operator:=x1And, _
            Criteria2:="<8000"

        ' Gefilterten Bereich nach Tabelle2 kopieren
        ws2.UsedRange.Clear
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

```
.CurrentRegion.Copy
With ws2
    .Paste Destination:=.Range("A1")
End With
Application.CutCopyMode = False
End With

Set ws1 = Nothing
Set ws2 = Nothing
End Sub
```

Es stehen insgesamt sechs Konstanten zur Verfügung:

Konstante	Index	Beschreibung
xlAnd	1	Und
xlBottom10Items	4	Die untersten x Elemente
xlBottom10Percent	6	Die untersten x Prozent
xlOr	2	Oder
xlTop10Items	3	Die obersten x Elemente
xlTop10Percent	5	Die untersten x Prozent

Tabelle 22: Konstanten für den Autofilter

Innerhalb der Kriterien stehen folgende Operatoren zur Verfügung:

Operator	Beschreibung
den genauen Wert eingeben (z.B. Criteria1:=»Weber«	entspricht
<>	entspricht nicht
>	ist größer als
>=	ist größer oder gleich
<	ist kleiner als
<=	ist kleiner oder gleich
=W*	beginnt mit dem Buchstaben »W«
<>W*	beginnt nicht mit dem Buchstaben »W«
=*W	endet mit dem Buchstaben »W«
<>*W	endet nicht mit dem Buchstaben »W«
=*W*	enthält den Buchstaben »W«
<>*W*	enthält nicht den Buchstaben »W«

Tabelle 23: Operatoren für Kriterien

Hinweis

Im DropDown-Menü des Autofilters werden maximal 1000 eindeutige Einträge angezeigt. Dies ist eine Einschränkung von Excel. Der Autofilter selbst funktioniert korrekt, auch wenn das DropDown-Menü nur 1000 Einträge umfasst. Falls Sie nach einem Wert filtern möchten, der nicht im DropDown-Menü angezeigt wird, klicken Sie auf einen der Pfeile und wählen aus dem sich öffnenden Menü den Eintrag (BENUTZERDEFINIERT). Hier können Sie jeden beliebigen Suchbegriff eingeben, auch jene, welche das DropDown-Menü des Autofilters nicht anzeigt.

Pro Filter-Feld können nur zwei Kriterien verwendet werden. Wenn Sie mit mehr Kriterien arbeiten möchten, dann verwenden Sie den Spezialfilter. Beispiele zum Spezialfilter finden Sie auf den folgenden Seiten.

Tipp

Wenn Sie gefilterte Daten berechnen möchten, reicht es nicht aus, beispielsweise die Funktion »Summe« zu verwenden, denn dabei würden in eine Berechnung auch die nicht sichtbaren Daten mit einbezogen. Um gefilterte Daten zu berechnen, verwenden Sie die Funktion »Teilergebnis«.

Innerhalb der Funktion »Teilergebnis« wird in der runden Klammer an erster Stelle angegeben, welche Funktion verwendet werden soll. Danach wird der Bereich festgelegt. Um eine Summe zu bilden, könnte die Formel wie folgt aussehen:

=TEILERGEBNIS(9;C2:C21)

Es stehen insgesamt 11 Funktionen zur Verfügung:

Wert	Funktion
1	MITTELWERT
2	ANZAHL
3	ANZAHL2
4	MAX
5	MIN
6	PRODUKT
7	STABW
8	STABWN
9	SUMME
10	VARIANZ
11	VARIANZEN

Tabelle 24: Werte für die Funktion »Teilergebnis«

113 Autofiltern nach Datum

Wenn Sie nach einem Datum filtern möchten, dann müssen Sie bedenken, dass in VBA das Datum im US-Format verwendet wird: MM/TT/YYYY und nicht wie bei uns das Format: TT.MM.JJJJ. Dementsprechend muss auch das amerikanische Datumsformat im VBA-Code eingegeben werden:

Grundlagen

Allgemein

Datum und Zeit

tungen

Steuerelemente

Befehlsliste

Objekt

Diagramm

Ereignisse

User-Form

Web/Mail

Extern

Gemischtes

Spezial

```

' =====
' Auf CD      Buchdaten\Beispiele\04_Auswertungen\
' Dateiname   04_05_Autofilter.xls
' Modul       mdl_02_AutofilterDate
' =====

Sub AutofilterDate1()
    With Worksheets(1).Range("A1")
        ' Autofilter auf Datumswerte
        .AutoFilter Field:=4, _
            Criteria1:=">=05/01/2003", _
            Operator:="x1And", _
            Criteria2:="<=05/15/2003"
    End With
End Sub

```

Wenn Sie nicht mit der Schreibweise mm/tt/jjjj arbeiten möchten, dann können wir Ihnen an dieser Stelle auch eine Alternative aufzeigen. Schreiben Sie das gewünschte Datum in eine beliebige Zelle. In unserem Beispiel ist es die Zelle F1. Innerhalb des Autofilters verwenden wir als erstes Kriterium »größer gleich« (>=) und verketteten dieses mit der Zelle. Wichtig ist dabei, dass zusätzlich die Eigenschaft Value2 verwendet wird. Als zweites Kriterium verwenden wir »kleiner gleich« und verketteten es ebenfalls mit der Zelle und der Eigenschaft Value2.

	A	B	C	D	E	F
	Name	Vorname	Betrag	Datum		
1						01.05.2003
2	Weber	Monika	5000	11.05.2003		15.05.2003
3	Breden	Melanie	3000	15.05.2003		
4	Can	Zulfu	2000	01.05.2003		
14	Schmid	Kurt	9000	11.05.2003		
19	Weber	Ramona	9000	11.05.2003		
24			28000			

Abbildung 94: Filtern nach Datum

Wenn Sie einen Datumsbereich ausgeben möchten, z.B. 01.05.2003 bis 15.05.2003, dann schreiben Sie die beiden Daten in zwei verschiedene Zellen (F1 und F2).

```

' =====
' Auf CD      Buchdaten\Beispiele\04_Auswertungen\
' Dateiname   04_05_Autofilter.xls
' Modul       mdl_02_AutofilterDate
' =====

Sub AutofilterDate2()
    With Worksheets(1).Range("A1")
        ' Autofilter auf Datumswerte
        .AutoFilter Field:=4, _
            Criteria1:=">=" & Range("F1").Value2, _
            Operator:="x1And", _
            Criteria2:="<=" & Range("F2").Value2
    End With
End Sub

```

114 Autofilter deaktivieren

Sie haben die Möglichkeit, den Autofilter zu deaktivieren, indem Sie die Eigenschaft `AutoFilterMode` auf den Wert `False` setzen. Um den Autofilter manuell zu deaktivieren, klicken Sie zuerst in eine Spalte, die den Autofilter enthält, und wählen dann den Menüpunkt **DATEN | FILTER | AUTO-FILTER**.

```
'=====
' Auf CD      Buchdaten\Beispiele\04_Auswertungen\
' Dateiname   04_05_Autofilter.xls
' Modul       mdl_03_AutofilterOff
'=====

Sub AutofilterOff()
    ActiveSheet.AutoFilterMode = False
End Sub
```

115 Spezialfilter mit mehr als zwei Kriterien

Der Spezialfilter eignet sich vor allem dann, wenn mehr als zwei Kriterien gefiltert werden sollen. Das Arbeiten mit dem Spezialfilter ist zu Beginn etwas gewöhnungsbedürftig. Wir möchten Ihnen deshalb anhand eines kleinen Beispiels zeigen, wie Sie ihn manuell, also ohne VBA, verwenden können.

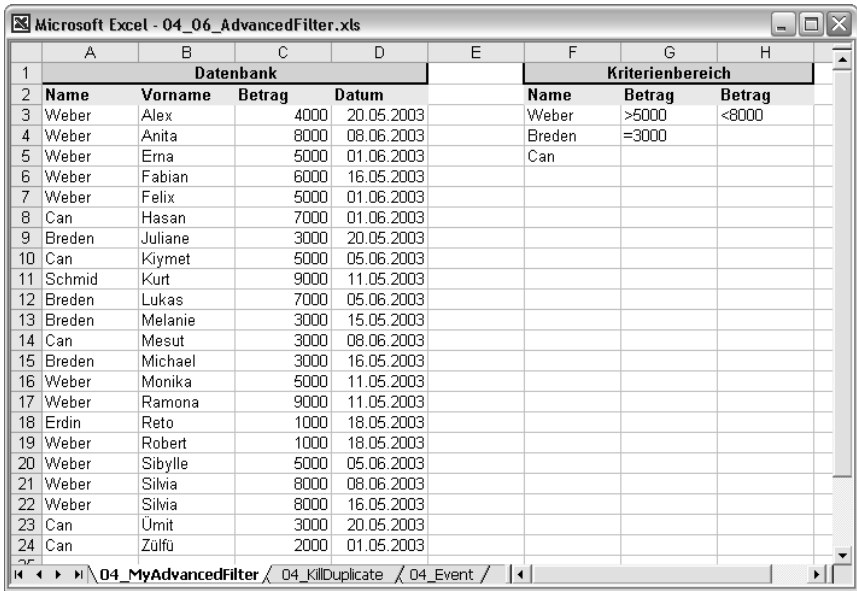


Abbildung 95: Spezialfilter

1. Bauen Sie wie gewohnt Ihre Datenbank auf.
2. Schreiben Sie die zu filternden Kriterien in separate Spalten (siehe Abbildung 95). Damit der Spezialfilter die Felder identifizieren kann, müssen die zu filternden Überschriften übereinstimmen.

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

3. Tragen Sie unterhalb der Überschriften (Spalte F bis H) die gewünschten Filterkriterien ein. In unserem Beispiel sollen drei Namen gefiltert werden. Beachten Sie, dass die Überschrift BETRAG zweimal aufgeführt ist. Auf diese Weise können die Werte wahlweise eingegrenzt werden. Wenn Sie ein Gleichheitszeichen (=) verwenden, dann setzen Sie ein Hochkomma (') voran, denn sonst geht Excel davon aus, dass Sie in dem Feld etwas berechnen möchten.
4. Aktivieren Sie eine Zelle innerhalb der Datenbank und starten Sie den Spezialfilter. Sie finden den Spezialfilter unter dem Menüpunkt DATEN | FILTER | SPEZIALFILTER.
5. In dem Dialogfenster, das sich nun öffnet, können Sie wählen, ob die gefilterten Daten an der gleichen Stelle oder in einem anderen Bereich ausgegeben werden sollen. Wenn Sie die zweite Option wählen, dann wird das Feld KOPIEREN NACH aktiv und es kann zusätzlich eine Zielzelle eingegeben werden. Der LISTENBEREICH ist der Bereich der Datenbank. Der KRITERIENBEREICH ist der Bereich, in welchem Sie die Filterkriterien eingetragen haben. Achten Sie darauf, die Überschriftenzeile in der Bereichsangabe zu integrieren. Sie können zusätzlich das Kontrollkästchen KEINE DUPLIKATE aktivieren (darauf werden wir in einem separaten Rezept zurückkommen).



Abbildung 96: Dialogfenster für den Spezialfilter

6. Bestätigen Sie die Eingaben, indem Sie auf die Schaltfläche OK klicken.

Die gefilterten Daten werden in diesem Beispiel in einem separaten Bereich (ab Zelle A30) ausgegeben.

Hinweis

Wenn Sie die Option LISTE AN GLEICHER STELLE FILTERN gewählt hätten, dann wäre das Verhalten ähnlich dem Autofilter. Wenn Sie später alle Spalten wieder einblenden möchten, dann wählen Sie den Menüpunkt DATEN | FILTER | ALLE ANZEIGEN.

Wenn Sie den Makro-Rekorder die Schritte aufzeichnen lassen, dann erhalten Sie den unten stehenden Code in einem Standardmodul des aktiven VBA-Projektes.

Die Methode `AdvancedFilter` steht für den Spezialfilter. Mittels `Action` und der Konstanten `xlFilterCopy` legen Sie fest, dass eine Kopie der Datenbank erstellt werden soll. In diesem Falle ist zwingend ein Ausgabebereich `CopyToRange` erforderlich. Alternativ zu `xlFilterCopy` können Sie die Konstante `xlFilterInPlace` verwenden. Die Daten werden dann an gleicher Stelle ausgegeben und der `CopyToRange`-Befehl wäre nicht erforderlich. Die Anweisung `CriteriaRange` steht für den Kriterienbereich.


```
'=====
' Auf CD      Buchdaten\Beispiele\04_Auswertungen\
' Dateiname  04_06_AdvancedFilter.xls
' Tabelle    04_MyAdvancedFilter
' Modul      mdl_01_MyAdvancedFilter
'=====

Sub MyAdvancedFilter()
    Range("A2:D24").AdvancedFilter _
        Action:=xlFilterCopy, _
        CriteriaRange:=Range("F2:G5"), _
        CopyToRange:=Range("A30"), _
        Unique:=False
End Sub
```

Tipp

Sie können die gefilterten Daten auch auf einem anderen Tabellenblatt ausgeben. Das funktioniert allerdings nur, wenn der Spezialfilter von dem Tabellenblatt aus gestartet wird, in dem die Daten ausgegeben werden sollen.
Wenn Sie den Spezialfilter vom Quelldatenblatt her starten und als Ausgabe ein anderes Tabellenblatt wählen, dann wird eine wenig aussagekräftige Warnmeldung angezeigt:



Abbildung 97: Warnmeldung Spezialfilter

116 Doppelte Einträge ausblenden

Der Spezialfilter eignet sich ausgezeichnet, um doppelte Datensätze aus einer Datenbank zu entfernen. In unseren Beispielen wollen wir die Datensätze jedoch lediglich ausblenden. Sie können die gefilterten Daten natürlich auch an einer anderen Stelle oder auf einem anderen Tabellenblatt ausgeben. Die Beschreibung dazu finden Sie im vorangegangenen Rezept.

In unserem ersten Code werden wir mit `Unique:=True` festlegen, dass in dem angegebenen Datenbereich doppelte Datensätze ausgeblendet werden.

```
'=====
' Auf CD      Buchdaten\Beispiele\04_Auswertungen\
' Dateiname  04_06_AdvancedFilter.xls
' Tabelle    04_KillDuplicate
' Modul      mdl_02_KillDuplicate
'=====

Sub KillDuplicateRecord()
    Range("A2:D46").AdvancedFilter _
        Action:=xlFilterInPlace, _
        Unique:=True
End Sub
```

Grundlagen
Allgemein

Datum
Zeit

tungen

Steuerelemente

Befehle
leisten

Objekt

Diagramm

Ereignisse

User-Form

Web/Mail

Extern

Gemischtes

Spezial

Unser zweiter Code zeigt, dass der Filter auch auf eine einzelne Spalte gesetzt werden kann. Es wird nur im angegebenen Bereich der Spalte A geprüft, ob doppelte Datensätze vorhanden sind. Wenn dies zutrifft, werden die Duplikate ausgeblendet. In Bezug auf unsere Beispielmappe bedeutet das, dass nur fünf Namen stehen bleiben.

```
'=====
' Auf CD      Buchdaten\Beispiele\04_Auswertungen\
' Dateiname   04_06_AdvancedFilter.xls
' Tabelle     04_KillDuplicate
' Modul       mdl_02_KillDuplicate
'=====

Sub KillDuplicateColumnA()
    Range("A2:A46").AdvancedFilter _
        Action:=xlFilterInPlace, _
        Unique:=True
End Sub
```

117 Spezialfilter: Alle Datensätze wieder einblenden

Um alle Datensätze, die durch den Spezialfilter ausgeblendet wurden, wieder einzublenden, verwenden Sie die Methode `ShowAllData`. Damit der Debugger nicht gestartet wird, wenn kein gefilterter Bereich vorhanden ist, verwenden Sie zusätzlich eine If-Entscheidung, die prüft, ob ein aktiver Filtermodus vorhanden ist (`FilterMode = True`).

```
'=====
' Auf CD      Buchdaten\Beispiele\04_Auswertungen\
' Dateiname   04_06_AdvancedFilter.xls
' Tabelle     04_KillDuplicate
' Modul       mdl_03_ShowAllMyData
'=====

Sub ShowAllMyData()
    With ActiveSheet
        If .FilterMode = True Then
            .ShowAllData
        End If
    End With
End Sub
```

118 Spezialfilter als Ereignisprozedur

Schön wäre es natürlich, wenn die gefilterten Daten sofort ausgegeben werden könnten, ohne dass eigens eine Prozedur gestartet werden muss. Warum auch nicht? Wir werden uns einer Ereignis-Prozedur bedienen, die direkt an das Tabellenblatt `04_EVENT` gebunden ist. Das Ereignis soll reagieren, wenn sich ein Zelleninhalt im Bereich `A3:E4` des angegebenen Tabellenblattes (`04_EVENT`) verändert. Dazu verwenden wir das `Worksheet_Change`-Ereignis des Tabellenblattes `04_EVENT`. Mehr zum Thema Ereignisprozeduren erfahren Sie in der Kategorie »Ereignisse«.

Im Rezept 113 haben wir darauf hingewiesen, dass auch über andere Tabellenblätter hinweg gefiltert werden kann. Dazu muss der Spezialfilter von dem Tabellenblatt aus gestartet werden, in dem

die Daten ausgegeben werden sollen. Genau dies werden wir hier tun. Wir verwenden zwei Tabellenblätter. Im Tabellenblatt 04_DATABASE befindet sich die Datenbank, in 04_EVENT werden die gefilterten Daten ausgegeben.

Filter-Kriterien:				
Name	Vorname	Betrag	Betrag	Datum
C				01.05.2003
W	R	>4000	<10000	11.05.2003

Name	Vorname	Betrag	Datum
Can	Mesut	3000	01.05.2003
Weber	Ramona	9000	11.05.2003
Weber	Robert	5000	11.05.2003
Can	Zülfü	2000	01.05.2003

Name	Vorname	Betrag	Datum
Weber	Alex	4000	20.05.2003
Weber	Anita	8000	08.06.2003
Weber	Erna	5000	01.06.2003
Weber	Fabian	6000	16.05.2003
Weber	Felix	5000	01.06.2003
Can	Hasan	7000	01.06.2003
Breden	Juliane	3000	20.05.2003
Can	Kiymet	5000	05.06.2003
Schmid	Kurt	9000	11.05.2003
Breden	Lukas	7000	05.06.2003
Breden	Melanie	3000	15.05.2003
Can	Mesut	3000	01.05.2003

Abbildung 98: Spezialfilter über zwei Tabellenblätter als Ereignisprozedur

Eine saubere Referenzierung nach der Variablendeklaration ermöglicht uns einen guten Überblick über den Spezialfilter, da innerhalb des Codes nicht jedes Mal der gesamte Befehl ausgeschrieben werden muss. Der Filter ist so aufgebaut, dass von den gefilterten Datensätzen aus 04_DATABASE jeweils eine Kopie auf dem Tabellenblatt 04_EVENT der gleichen Excel-Arbeitsmappe ausgegeben wird (Action:=xlFilterCopy). Als Kriterium CriteriaRange verwenden wir den Bereich A2:E4 der Tabelle 04_EVENT. Der Ausgabebereich CopyToRange befindet sich im Tabellenblatt 04_EVENT im Bereich A7:D8. Wenn Sie die Anzeige von Duplikaten unterdrücken möchten, dann ändern Sie bei Unique den Wert auf True.

Falls Sie mehr als zwei Zeilen für die Eingabe der Kriterien benötigen, dann können Sie weitere Zeilen einfügen. Denken Sie dabei daran, dass auch die Bereiche der Prozedur angepasst werden müssen. Sie können wahlweise auch mit benannten Bereichen arbeiten.

```

' =====
' Auf CD      Buchdaten\Beispiele\04_Auswertungen\
' Dateiname  04_06_AdvancedFilter.xls
' Tabelle    04_Event und 04_Database
' Ereignis   Tabelle4(04_Event)
' =====

```

```

Private Sub Worksheet_Change(ByVal Target As Range)
    Dim rngDB As Range
    Dim rngFilter As Range
    Dim rngOutput As Range

    Set rngDB = Worksheets("04_Database").Range("A1:D23")
    Set rngFilter = Worksheets("04_Event").Range("A2:E4")
    Set rngOutput = Worksheets("04_Event").Range("A7:D8")

    If Not (Application.Intersect(rngFilter, Target) Is Nothing) Then
        rngDB.AdvancedFilter _
            Action:=xlFilterCopy, _
            CriteriaRange:=rngFilter, _

```

Grundlagen

Allgemein

Datum
Zeit

tungen

Steuer
elementeBefehl
leisten

Objekt

Diagramm

Ereignisse

User-
FormWeb/
Mail

Extern

Gemischtes

Spezial

```

        CopyToRange:=rngOutput, _
        Unique:=False
    End If

    Set rngDB = Nothing
    Set rngFilter = Nothing
    Set rngOutput = Nothing
End Sub

```

119 Gültigkeit

In Excel können Sie Gültigkeitsprüfungen für Zellen festlegen. Sie können beispielsweise die Eingabe so einschränken, dass nur Werte im Bereich von 1 bis 50 in die Zelle eingegeben werden können. Falls ein anderer Wert in die Zelle eingegeben wird, dann erscheint eine Fehlermeldung.

Sie finden die Gültigkeit unter dem Menüpunkt DATEN | GÜLTIGKEIT. Es stehen drei Registerkarten in dem Dialogfeld zur Verfügung. In der Registerkarte EINSTELLUNGEN können Sie die Einschränkung festlegen. Es stehen zwei DropDown-Menüs zur Verfügung. Im ersten DropDown ZULASSEN können Sie wählen, was zugelassen werden soll. Beispielsweise nur eine GANZZAHL oder ein DATUM usw. Im zweiten DropDown-Feld DATEN können Sie eine weitere Auswahl treffen, zum Beispiel ZWISCHEN. In den beiden Eingabefeldern MINIMUM und MAXIMUM können Sie den kleinsten und größten zugelassenen Wert eingeben.

In der Registerkarte EINGABEMELDUNG können Sie einen TITEL und eine EINGABEMELDUNG hinterlegen. Beim Aktivieren der Zelle erscheint dieser selbst definierte Text. Sinnvollerweise könnten Sie z.B. die Werte anzeigen lassen, die zugelassen sind.

In der Registerkarte FEHLERMELDUNG haben Sie die Möglichkeit, einen TITEL und FEHLERMELDUNG zu hinterlegen. Diese Meldung erscheint, wenn ein nicht zugelassener Wert in die Zelle eingegeben wurde. Sie können zudem wählen, welcher Typ von Icon in der Fehlermeldung angezeigt werden soll.

In der nachfolgenden Prozedur wird im Bereich A1:B5 jeder Zelle die gleiche Gültigkeit zugewiesen. Zu Beginn der Prozedur werden vorhandene Gültigkeiten mit der Delete-Methode gelöscht. Danach wird mit der Add-Methode eine neue Gültigkeit erstellt. Die angegebene Konstante im Type-Argument ist eine Ganzzahl. Bei der Warnmeldung AlertStyle soll das Stopp-Symbol verwendet werden. Als Operator wurde ZWISCHEN gewählt. Mittels Formula1 wird der Minimalwert festgelegt und mittels Formula2 der Maximalwert. Leere Zellen werden mittels IgnoreBlank ignoriert. Ein DropDown-Pfeil InCellDropdown soll angezeigt werden. Im zweitletzten Codeblock wird die Eingabemeldung aufbereitet und im letzten Codeblock die Fehlermeldung, die bei falscher Eingabe angezeigt werden soll.

```

' =====
' Auf CD      Buchdaten\Beispiele\04_Auswertungen\
' Dateiname  04_07_DataValidation.xls
' Tabelle     04_MyValidation
' Modul      mdl_01_MyDataValidation
' =====

Sub MyDataValidation()
    With Range("A1:B5").Validation
        ' Evtl. vorhandene Gültigkeit löschen

```

```
.Delete

' Neue Gültigkeit erstellen
.Add _
    Type:=xlValidateWholeNumber, _
    AlertStyle:=xlValidAlertStop, _
    Operator:=xlBetween, _
    Formula1:="1", _
    Formula2:="50"

' Leere Zellen sollen ignoriert werden
.IgnoreBlank = True

' Das DropDown soll sichtbar sein
.InCellDropdown = True

' Eingabe-Meldung anzeigen
.InputTitle = "Zugelassen"
.InputMessage = "Zugelassener Wert: 1 bis 50."
.ShowInput = True

' Bei falscher Eingabe eine Fehlermeldung anzeigen
.ErrorTitle = "Falsche Eingabe"
.ErrorMessage = "Sie haben nicht einen Wert " & _
                "zwischen 1 und 50 eingegeben."
.ShowError = True
End With
End Sub
```

Um die Gültigkeit im Bereich A1:B5 zu entfernen, verwenden Sie die Anweisung `Validation.Delete`:

```
' =====
' Auf CD      Buchdaten\Beispiele\04_Auswertungen\
' Dateiname  04_07_DataValidation.xls
' Tabelle     04_MyValidation
' Modul      mdl_02_DeleteValidation
' =====

Sub DeleteValidation()
    Range("A1:B5").Validation.Delete
End Sub
```

Den nachfolgenden Tabellen können Sie die verfügbaren Konstanten zu den Gültigkeitstypen entnehmen:

Type	Index	Beschreibung
xlValidateCustom	7	Benutzerdefiniert
xlValidateDate	4	Datum

Tabelle 25: Gültigkeitstyp

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

Type	Index	Beschreibung
xlValidateDecimal	2	Dezimal
xlValidateInputOnly	0	Jeden Wert
xlValidateList	3	Liste
xlValidateTextLength	6	Textlänge
xlValidateTime	5	Zeit
xlValidateWholeNumber	1	Ganze Zahl

Tabelle 25: Gültigkeitstyp (Forts.)

AlertStyle	Index	Beschreibung
xlValidAlertInformation	3	Weißes Sprechblasen-Symbol mit blauem Ausrufezeichen
xlValidAlertStop	1	Roter Kreis mit weißem Kreuz
xlValidAlertWarning	2	Gelbes Dreieck mit schwarzem Ausrufezeichen

Tabelle 26: Symbol-Typ (Icon)

Operator	Index	Beschreibung
xlBetween	1	Zwischen
xlEqual	3	Gleich
xlGreater	5	Größer als
xlGreaterEqual	7	Größer oder gleich
xlLess	6	Kleiner als
xlLessEqual	8	Kleiner oder gleich
xlNotBetween	2	Nicht zwischen
xlNotEqual	4	Ungleich

Tabelle 27: Operatoren

120 Gültigkeit auf ein anderes Tabellenblatt

Wenn Sie mit einer Gültigkeitsliste arbeiten, sich die Liste jedoch auf einem anderen Tabellenblatt oder in einer anderen Mappe befindet, müssen Sie den Bereich, der als Liste gilt, zuerst benennen (EINFÜGEN | NAME | DEFINIEREN). Ansonsten wird es nicht möglich sein, eine tabellenübergreifende Gültigkeit zu erstellen.

In unserem Beispiel verwenden wir zwei Tabellenblätter. Im Tabellenblatt 04_VALIDATIONLIST befindet sich die anzuzeigende Liste. Im Tabellenblatt 04_VALIDATION soll das Zellen-DropDown über die Gültigkeit-Liste angezeigt werden.

Benennen Sie zuerst den Bereich in der Tabelle, in der sich die Liste befindet. Weisen Sie den Namen `MeineListe` zu. Im VBA-Code weisen wir dem Bereich A1:B2 des Tabellenblatts 04_VALIDATION die DropDown-Felder zu. Wir verweisen dabei auf den Namen `MeineListe`. Achten Sie darauf, dass dem Namen ebenso wie bei der manuellen Eingabe ein Gleichheitszeichen (=) vorangestellt werden muss.

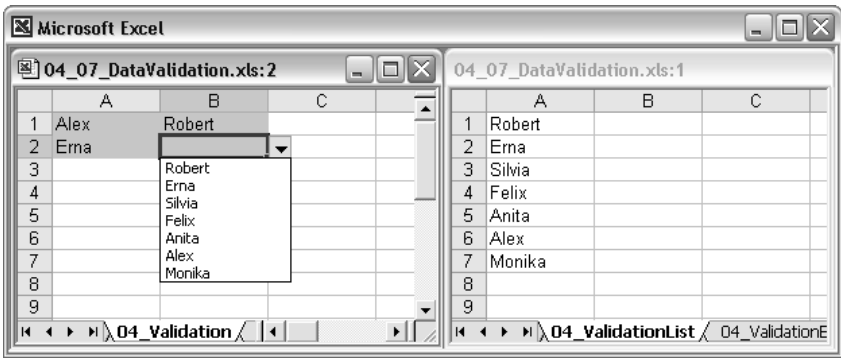


Abbildung 99: Gültigkeit auf ein anderes Tabellenblatt

Hinweis Gültigkeiten können nicht nur auf andere Tabellenblätter, sondern auch auf andere Mappen erstellt werden. Dabei muss allerdings in der Gültigkeit der Mappenname mit angegeben werden. Zum Beispiel: `= [Mappe2.xls]Tabelle1!$A$1:$A$7`

```
'=====
' Auf CD      Buchdaten\Beispiele\04_Auswertungen\
' Dateiname   04_07_DataValidation.xls
' Tabelle     04_Validation und 04_ValidationList
' Modul       mdl_03_ValidationList
'=====

Sub ValidationList()
    ' Gültigkeit erstellen
    With Worksheets("04_Validation").Range("A1:B2").Validation
        .Delete
        .Add _
            Type:=xlValidateList, _
            Formula1:="=MeineListe"
    End With
End Sub
```

Hinweis Wenn Sie eine Gültigkeit-Liste erstellen und die Daten direkt in das Feld QUELLE eingeben, dann bedenken Sie, dass die Anzahl der Zeichen auf 255 beschränkt ist. Erstellen Sie alternativ die Daten in einer separaten Spalte und geben Sie in der QUELLE lediglich den Bezug ein. Sie können die Spalte dann wahlweise ausblenden, damit die Datenquelle nicht sichtbar ist.

121 Gültigkeit als Ereignis-Prozedur

Die Gültigkeitsprüfung ist zwar sehr nützlich, hat jedoch einen entscheidenden Nachteil. Gehen wir einmal davon aus, dass Sie in der Zelle A1 die Eingabe auf eine Ganzzahl zwischen 1 und 50 beschränkt haben. Nun kopieren Sie den Inhalt der Zelle B1, welche einen Wert von 100 aufweist, in die Zelle A1. Die Zelle A1 wird mit dem ungültigen Wert aus Zelle B1 überschrieben. Die Gültigkeit ist nicht mehr vorhanden, denn sie wurde beim Kopieren ebenfalls überschrieben.

- Grundlagen
- Allgemein
- Datum und Zeit
- Formeln
- Statistik
- Diagramme
- Ergebnisse
- User-Formen
- Web/Mail
- Extern
- Gemischtes
- Spezial

Verhindern können Sie dies nicht, aber Sie können eine Ereignisprozedur erstellen, die die Gültigkeit in dem Bereich umgehend wieder erstellt, wenn sie aus irgendeinem Grund nicht mehr vorhanden sein sollte. Wir verwenden dabei das `Worksheet_Change`-Ereignis des entsprechenden Tabellenblattes. Mehr zum Thema Ereignis-Prozeduren erfahren Sie in der Kategorie »Ereignisse«. Zu Beginn der Ereignis-Prozedur wird festgelegt, dass die Prozedur nur ausgeführt werden soll, wenn eine Veränderung im Bereich A1:B5 stattfindet. Danach wird ein `Errorhandler`-Verzweigung eingefügt. In der ersten `If`-Entscheidung wird geprüft, ob in der aktiven Zelle (`Target`) bereits eine Gültigkeit vorhanden ist. Sollte dies zutreffen, verlassen wir die Prozedur. Damit stellen wir sicher, dass eine bestehende Gültigkeit nicht durch die neue überschrieben wird. Falls in der Zelle keine Gültigkeit vorhanden ist, wird ein Fehler ausgelöst. Diesen Fehler machen wir uns zunutze. Beim Nichtvorhandensein einer Gültigkeit werden wir nach Eintritt in die Sprungmarke `Errorhandler` eine neue Gültigkeit erstellen.

Es soll nur eine Ganzzahl im Bereich von 1 bis 50 zugelassen werden. Falls nun der Fall eintreten sollte, dass eine vorhandene Gültigkeit überschrieben wird, indem ein unerlaubter Wert in die Zelle kopiert wird, müssen wir prüfen, ob der Wert gültig ist. Dies geschieht in der zweiten `If`-Anweisung. Wenn es sich um einen ungültigen Wert handeln sollte, wird eine entsprechende Meldung auf dem Bildschirm angezeigt. Der ungültige Wert wird mittels `ClearContents` gelöscht und die Zelle für eine Neueingabe wieder aktiviert. Da die Ereignis-Prozedur bei diesen beiden Codezeilen einen Fehler bzw. eine Endlosschleife auslösen würde, werden zu Beginn der Prozedur alle Ereignisse mit `EnableEvents = False` deaktiviert. Direkt nach dem Ausführen der beiden Codezeilen werden die Ereignisse wieder aktiviert. Nun müssen wir nur noch die gewünschte Gültigkeit einfügen. Der dazu verwendete Code entspricht dem Beispiel aus dem Rezept 117.

```
'=====
' Auf CD      Buchdaten\Beispiele\04_Auswertungen\
' Dateiname   04_07_DataValidation.xls
' Tabelle     04_ValidationEvent
' Ereignis    Tabelle4(04_ValidationEvent)
'=====
```

```
Private Sub Worksheet_Change(ByVal Target As Range)
    ' Bereich festlegen
    If Intersect(Target, Range("A1:B5")) Is Nothing _
        Or Target.Count > 1 Then
        Exit Sub
    End If

    On Error GoTo ErrorHandler

    ' Wenn bereits eine Gültigkeit vorhanden ist,
    ' dann die Prozedur verlassen
    If Target.Validation.Type >= 0 Then
        Exit Sub
    End If
```

Errorhandler:

```
With Target
    ' Prüfen, ob der eingegebene Wert im gültigen Bereich liegt
    If .Value < 1 Or .Value > 50 Then
        MsgBox "Sie haben nicht einen Wert " & _
```

```

        "zwischen 1 und 50 eingegeben.", _
        Title:="Falsche Eingabe"

' Bei ungültiger Eingabe, Zellinhalt löschen und
' Zelle wieder aktivieren
Application.EnableEvents = False
.ClearContents
.Activate
Application.EnableEvents = True
End If

' Gültigkeit erstellen
With .Validation
.Delete
.Add _
    Type:=xlValidateWholeNumber, _
    AlertStyle:=xlValidAlertStop, _
    Operator:=xlBetween, _
    Formula1:="1", _
    Formula2:="50"
.IgnoreBlank = True
.InCellDropdown = True
.InputTitle = "Zugelassen"
.InputMessage = "Zugelassener Wert: 1 bis 50."
.ShowInput = True
.ErrorTitle = "Falsche Eingabe"
.ErrorMessage = "Sie haben nicht einen Wert " & _
    "zwischen 1 und 50 eingegeben."
.ShowError = True
End With
End With
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

122 Eine Pivot-Tabelle erzeugen

Pivot-Tabellen sind ein sehr nützliches Werkzeug, mit dem Sie umfangreiche Daten schnell auswerten können. Mit einer Pivot-Tabelle können Sie die Daten auf einfachste Weise dynamisch gestalten und umgestalten. Oftmals liegen die Daten in rohen Tabellen vor und es ist z.B. nur schwer zu erkennen, wie viele Früchte durch welchen Kunden gekauft wurden (siehe linkes Tabellenblatt der Abbildung 100).

Um eine Pivot-Tabelle entsprechend der Tabelle 1 aus Abbildung 100 zu erstellen, setzen Sie zuerst den Cursor in die auszuwertende Tabelle und wählen dann den Menüpunkt DATEN | PIVOT-TABLE- UND PIVOTCHART-BERICHT. Der Pivot-Tabellen-Assistent wird gestartet. Klicken Sie im ersten Schritt des Assistenten auf WEITER. Sofern Sie den Cursor vor dem Start des Assistenten bereits in die auszuwertende Tabelle gesetzt haben, können Sie nun wiederum auf die Schaltfläche WEITER klicken. Ansonsten geben Sie manuell den richtigen Bereich ein.

Im letzten Schritt des Assistenten können Sie wählen, ob die Pivot-Tabelle in einem neuen Tabellenblatt oder im bestehenden Arbeitsblatt erstellt werden soll. Wenn Sie die zweite Option auswählen, dann müssen Sie zusätzlich den gewünschten Zielbereich angeben. Im letzten Schritt des Assistenten finden Sie zudem die Schaltfläche LAYOUT. Ein Klick auf diese Schaltfläche öffnet ein weiteres Dialogfenster. Hier können Sie die Anordnung Ihrer Felder vornehmen. Die Einstellung, die Sie hier vornehmen, kann später jederzeit umgestellt werden.

The screenshot shows two Excel windows. The left window, '04_08_PivotTable.xls:2', displays a data table with columns A to F. The right window, '04_08_PivotTable.xls:1', shows a PivotTable with 'Datum' and 'Kunde' in the row labels, 'Artikel' in the column labels, and 'Summe von Total' as the value field. A 'PivotTable-Feldliste' (PivotTable Field List) task pane is open on the right, listing the available fields: Datum, Kunde, Artikel, Anzahl, Preis, and Total.

	A	B	C	D	E	F
1	Datum	Kunde	Artikel	Anzahl	Preis	Total
2	17.05.2003	Weber	Gemüse	50	30	1500
3	17.05.2003	Can	Früchte	100	20	2000
4	19.05.2003	Weber	Non-Food	35	10	350
5	20.05.2003	Can	Gemüse	25	30	750
6	20.05.2003	Breden	Gemüse	83	30	2490
7	20.05.2003	Breden	Non-Food	105	10	1050
8	21.05.2003	Weber	Früchte	29	20	580
9	21.05.2003	Breden	Non-Food	55	10	550
10	21.05.2003	Weber	Gemüse	35	30	1050
11	22.05.2003	Can	Non-Food	150	10	1500
12	23.05.2003	Breden	Gemüse	70	30	2100
13	24.05.2003	Weber	Non-Food	50	10	500
14	26.05.2003	Can	Non-Food	75	10	750
15	26.05.2003	Breden	Gemüse	80	30	2400
16	27.05.2003	Weber	Non-Food	90	10	900
17	28.05.2003	Can	Früchte	58	20	1160
18	29.05.2003	Weber	Gemüse	89	30	2670
19	29.05.2003	Breden	Gemüse	93	30	2790
20	29.05.2003	Breden	Früchte	49	20	980
21	30.05.2003	Can	Früchte	35	20	700
22	30.05.2003	Breden	Non-Food	94	10	940

	A	B	C	D	E
1	Datum	(Alle)			
2					
3	Summe von Total	Kunde			
4	Artikel	Breden	Can	Weber	Gesamtergebnis
5	Früchte	2140	3860	580	6580
6	Gemüse	9780	2460	5220	17460
7	Non-Food	2540	2250	2130	6920
8	Gesamtergebnis	14460	8570	7930	30960

Abbildung 100: Basistabelle und Pivot-Tabelle

In der Mitte des Dialogfensters sind vier Felder zu erkennen:

- ▶ Seite
- ▶ Zeile
- ▶ Spalte
- ▶ Daten

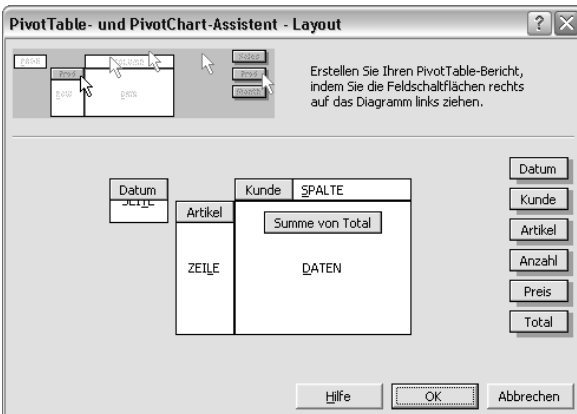


Abbildung 101: Layout der Pivot-Tabelle

Rechterhand sind verschiedene Felder aufgeführt. Jedes der Felder entspricht einer Spaltenüberschrift Ihrer Tabelle. Diese Überschriften können Sie nun mit gedrückter linker Maustaste in die Pivot-Tabellenfelder ziehen. Es können pro Feld auch mehrere Überschriften verwendet werden. Im Feld DATEN müssen immer numerische Werte enthalten sein, denn hier finden die Berechnungen statt.

Nachdem die Felder wie gewünscht positioniert wurden, können Sie das Dialogfenster schließen, indem Sie die Schaltfläche OK anklicken. Sie gelangen zurück zum Assistenten und können auch diesen über die Schaltfläche FERTIG STELLEN beenden. Die Pivot-Tabelle wird nun erstellt und ein

zusätzliches Fenster mit den Feldüberschriften (Spaltenüberschriften) angezeigt. Das Fenster ist nur sichtbar, wenn eine der Zellen innerhalb der Pivot-Tabelle aktiv ist. Es wird zudem eine Symbolleiste für Pivot-Tabellen eingeblendet.

Die Pivot-Felder können nun nach Belieben verschoben, hinzugefügt oder aus der Pivot-Tabelle entfernen werden. Verschiedene Auswertungen können so auf jede erdenkliche Weise erfolgen.

Um den entsprechenden VBA-Code zu erhalten, zeichnen Sie die Schritte am besten mit dem Makro-Rekorder auf und passen die Prozedur gegebenenfalls an.

Ein unbereinigter Code könnte wie folgt aussehen:

```
'=====
' Auf CD      Buchdaten\Beispiele\04_Auswertungen\
' Dateiname  04_08_PivotTable.xls
' Tabelle     04_CreatePivot
' Modul      mdl_01_MyPivotTable
'=====

Sub Makro1()
    ActiveWorkbook.PivotCaches.Add _
        (SourceType:=xlDatabase, _
        SourceData:="Tabelle1!R1C1:R25C6") _
        .CreatePivotTable _
            TableDestination:="", _
            TableName:="PivotTable1", _
            DefaultVersion:=xlPivotTableVersion10

    ActiveSheet.PivotTableWizard _
        TableDestination:=ActiveSheet.Cells(3, 1)

    ActiveSheet.Cells(3, 1).Select

    ActiveSheet.PivotTables("PivotTable4").AddFields _
        RowFields:="Artikel", _
        ColumnFields:="Kunde", _
        PageFields:="Datum"

    ActiveSheet.PivotTables("PivotTable4"). _
        PivotFields("Total").Orientation = _
        xlDataField
End Sub
```

Um den Code bereinigen zu können, müssen Sie mit den Objekten einer Pivot-Tabelle vertraut sein.

Objekt	Beschreibung
PivotCaches	Der Zwischenspeicher (Daten-Buffer) der Pivot-Tabellendaten. Unter Zugabe von Add und weiteren Anweisungen wird die Pivot-Tabelle erzeugt. Ein Workbook-Objekt.
PivotTables	Die Pivot-Tabelle selbst. Ein Workbook-Objekt.

Tabelle 28: Pivot-Tabellen-Objekte

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

Objekt	Beschreibung
PivotFields	Die Felder der Pivot-Tabelle. Ein PivotTable-Objekt.
PivotItems	Jedes Pivot-Tabellenfeld enthält eine Ansammlung an Pivot-Items. In unserem Beispiel haben wir im Feld ARTIKEL drei Items: »Gemüse«, »Früchte« und »Non-Food«. Ein PivotField-Objekt.
CreatePivotTable	Das Erzeugen der Pivot-Tabelle unter Angabe des Ziels und des Pivot-Tabellennamens. Ein PivotCaches-Objekt.
PivotTableWizard	Der Pivot-Tabellen-Assistent. Im bereinigten Code wird dieser nicht mehr verwendet. Ein Worksheet-Objekt.

Tabelle 28: Pivot-Tabellen-Objekte (Forts.)

Nachdem Sie die Objekte nun kennen, können wir den Code bereinigen. Eine korrekte Referenzierung erleichtert die Arbeit und präsentiert am Ende einen sauberen und übersichtlichen Code.

Am einfachsten ist es, wenn wir den Daten der Tabelle einen Namen zuweisen (EINFÜGEN | NAMEN | DEFINIEREN). Wir vergeben in unserem Beispiel den Namen DATENBANK. Auf diese Weise können wir im Code direkt auf den benannten Bereich zugreifen, ohne das Tabellenblatt anzusprechen.

Für den Fall, dass die Pivot-Tabelle bereits erstellt wurde, verwenden wir eine Fehlerbehandlung, die darauf hinweist.

```
'=====
' Auf CD      Buchdaten\Beispiele\04_Auswertungen\
' Dateiname  04_08_PivotTable.xls
' Tabelle     04_CreatePivot
' Modul       mdl_02_MyPivotTable
'=====
```

```
Sub MyPivotTable()
    Dim ptCache As PivotCache
    Dim pt As PivotTable

    On Error GoTo ErrorHandler

    Set ptCache = ActiveWorkbook.PivotCaches.Add _
        (SourceType:=xlDatabase, _
        SourceData:="Datenbank")

    Set pt = ptCache.CreatePivotTable _
        (TableDestination:="", _
        TableName:="MyPivotTable")

    With pt
```

```

        .PivotFields("Artikel").Orientation = xlRowField
        .PivotFields("Kunde").Orientation = xlColumnField
        .PivotFields("Datum").Orientation = xlPageField
        .PivotFields("Total").Orientation = xlDataField
    End With

    Set ptCache = Nothing
    Set pt = Nothing

Exit Sub

ErrorHandler:
    MsgBox ("Die Pivot-Tabelle besteht bereits.")
End Sub
```

123 Zwei Pivot-Tabellen auf einem Blatt erzeugen

In diesem Beispiel werden wir zwei unterschiedliche Pivot-Tabellen auf einem Tabellenblatt erzeugen. Die Pivot-Tabellen werden diesmal nicht in einem neuen Tabellenblatt erstellt, sondern auf jenem, in welchem sich auch die Quelldaten befinden.

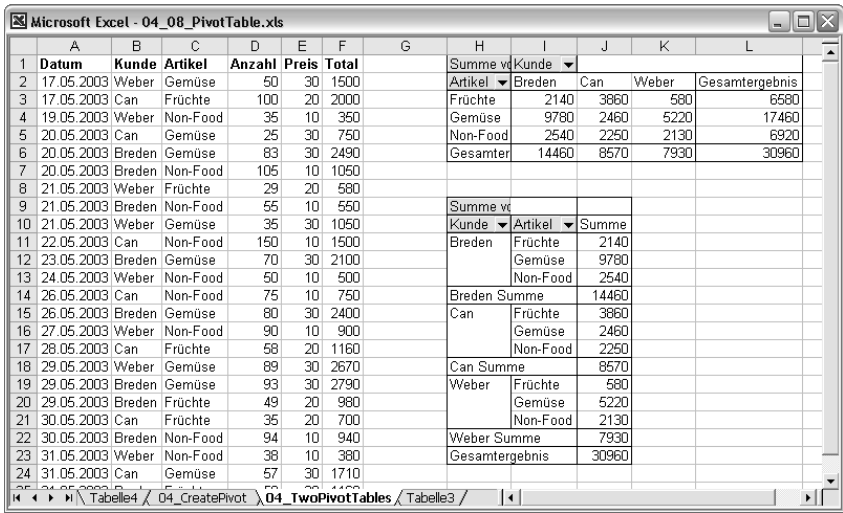


Abbildung 102: Zwei Pivot-Tabellen auf einem Tabellenblatt

Um diesmal auf eine Fehlerbehandlung zu verzichten, werden zu Beginn der Prozedur die im Tabellenblatt enthaltenen Pivot-Tabellen gelöscht (TableRange2.Delete).

Danach wird die erste Pivot-Tabelle erstellt. Beachten Sie die Referenzierung. Diesmal verzichten wir auf eine separate Referenzierung des Pivot-Caches, so wie das im Rezept 120 noch der Fall war. Erstmals verwenden wir einen nichtleeren Wert bei der Anweisung TableDestination. Damit die Pivot-Tabelle in keinem Fall Daten der Datenbank überlagert bzw. überschreiben kann, werden wir die vorhandene Anzahl an benutzten Spalten mit Columns.Count zählen. Zwei Spalten rechts des letzten benutzten Bereiches wird die Pivot-Tabelle erstellt (End(xlToLeft).Offset(0, 2)). Damit entsteht eine leere Spalte zwischen der Datenbank und den Pivot-Tabellen.

Grundlagen
Allgemein

Datum
Zeit

tungen

Steuer
elemente

Befehl
leisten

Objekt

Diagramm

Ereignisse

User-
Form

Web/
Mail

Extern

Gemischtes

Spezial

Das Verfahren für die zweite Pivot-Tabelle ist ähnlich, nur dass wir bei `TableDestination` sicherstellen, dass die zweite Pivot-Tabelle *unter* der ersten Pivot-Tabelle erstellt wird. Es sollen zwei leere Zeilen zwischen den beiden Spalten enthalten sein.

```
TableDestination:=Cells(Rows.Count, 8).End(xlUp).Offset(3, 0)
```

In der zweiten Pivot-Tabelle werden wir zudem die Felder anders anordnen, indem wir die Eigenschaft `Orientation` verwenden (siehe Abbildung 102).

```
'=====
' Auf CD      Buchdaten\Beispiele\04_Auswertungen\
' Dateiname   04_08_PivotTable.xls
' Tabelle     04_CreatePivot
' Modul       mdl_03_TwoPivotTables
'=====
```

```
Sub TwoPivotTables()
    Dim pt As PivotTable

    ' Vorhandene Pivot-Tabellen löschen
    For Each pt In ActiveSheet.PivotTables
        pt.TableRange2.Delete
    Next pt

    ' Erste Pivot-Tabelle aufbauen
    Set pt = ActiveWorkbook.PivotCaches.Add _
        (SourceType:=xlDatabase, _
        SourceData:="Datenbank"). _
        CreatePivotTable _
        (TableDestination:=Cells(1, Columns.Count). _
        End(xlToLeft).Offset(0, 2), _
        TableName:="MyFirstPivotTable")

    With pt
        .PivotFields("Artikel").Orientation = xlRowField
        .PivotFields("Kunde").Orientation = xlColumnField
        .PivotFields("Total").Orientation = xlDataField
    End With

    ' Zweite Pivot-Tabelle aufbauen
    Set pt = ActiveWorkbook.PivotCaches.Add _
        (SourceType:=xlDatabase, _
        SourceData:="Datenbank"). _
        CreatePivotTable _
        (TableDestination:=Cells(Rows.Count, 8). _
        End(xlUp).Offset(3, 0), _
        TableName:="MySecondPivotTable")

    With pt
        .PivotFields("Kunde").Orientation = xlRowField
        .PivotFields("Artikel").Orientation = xlRowField
        .PivotFields("Total").Orientation = xlDataField
    End With
End Sub
```

```
End With

Set pt = Nothing
End Sub
```

124 Eine bestehende Pivot-Tabelle formatieren

Sie können bestehende Pivot-Tabellen auch anders gestalten. Der folgende Code zeigt, wie Sie Datenfelder `DataRange` und den Datenbereich `DataBodyRange` andersfarbig als die Standardeinstellung von Excel und mit anderem Schriftschnitt formatieren können. Jede zugehörige Schaltfläche `LabelRange` kann zudem separat formatiert werden. Weitere Informationen zum Ablauf der Prozedur entnehmen Sie bitte den Kommentaren im Code.

```
'=====
' Auf CD      Buchdaten\Beispiele\04_Auswertungen\
' Dateiname  04_08_PivotTable.xls
' Tabelle     04_DesignPivotTable
' Modul      mdl_04_Design
'=====

Sub DesignPivotTable()
    Dim pt As PivotTables

    Set pt = ActiveSheet.PivotTables

    With pt(1)
        With .PivotFields("Kunde")
            ' Feld "Kunde" in Hellgelb und Fett
            With .DataRange
                .Interior.ColorIndex = 19
                .Font.Bold = True
            End With

            ' Schaltfläche "Kunde" in Gelb und Fett
            With .LabelRange
                .Interior.ColorIndex = 6
                .Font.Bold = True
            End With
        End With

        With .PivotFields("Artikel")
            ' Feld "Artikel" in Hellorange und Fett
            With .DataRange
                .Interior.ColorIndex = 40
                .Font.Bold = True
            End With

            ' Schaltfläche "Artikel" in Orange und Fett
            With .LabelRange
                .Interior.ColorIndex = 45
                .Font.Bold = True
            End With
        End With
    End With
End Sub
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

```
End With

' Datenbereich in Grau und Kursiv
With .DataBodyRange
    .Interior.ColorIndex = 15
    .Font.Italic = True
End With
End With

Set pt = Nothing
End Sub
```

Hinweis

Im selben Modul finden Sie auch den Code, der die Formatierungen zurücksetzt. Wir werden hier auf die Abbildung dieses Codes verzichten.

125 Datenpunkte ausblenden

Einzelne Datenpunkte einer Pivot-Tabelle können ausgeblendet werden. Innerhalb des Pivot-Feldes »Artikel« sind drei verschiedene Datenpunkte `PivotItem` enthalten: FRÜCHTE, GEMÜSE und NON-FOOD (der Letztere ist auf der Abbildung 103 ausgeblendet). Wenn Sie im Feld ARTIKEL auf den DropDown-Pfeil klicken, können Sie die einzelnen Datenpunkte gut erkennen. Wahlweise können einzelne Punkte aktiviert oder deaktiviert werden.

H	I	J	K	L	M	N
	Summe von Total	Kunde ▾				
	Artikel ▾	Breden	Can	Weber	Gesamtergebnis	
	Früchte	2140	3860	580	6580	
	Gemüse	9780	2460	5220	17460	
	Gesamtergebnis	11920	6320	5800	24040	
	☐ Früchte ausblenden			☒ Non-Food ausblenden		
	☐ Gemüse ausblenden			☐ Alles einblenden		

Abbildung 103: Datenpunkte ausblenden

Für unser Beispiel haben wir im Tabellenblatt 04_HIDEFIELDS vier Optionsfelder aus der Symbolleiste STEUERELEMENT-TOOLBOX eingefügt. Um die Optionsfelder ansprechen zu können, muss jedem eine eigene Ereignis-Prozedur hinterlegt werden. Die Ereignis-Prozeduren werden an das Tabellenblatt gebunden, in welchem sich die Optionsfelder befinden. Mehr zum Thema Steuerelemente können Sie in der Kategorie »Steuerelemente« nachlesen.

Es sind somit insgesamt vier Prozeduren für die Optionsfelder erforderlich, nämlich für jedes eine. Pro Optionsfeld kann ein einzelner Datenpunkt ausgeblendet werden. Das letzte Optionsfeld dient dazu, alle Datenpunkte wieder einzublenden. Ein Datenpunkt muss immer vorhanden sein, sonst wird der Debugger gestartet.

```
' =====
' Auf CD      Buchdaten\Beispiele\04_Auswertungen\
' Dateiname  04_08_PivotTable.xls
```



```
' Tabelle      04_HideFields
' Ereignis     Tabelle4(04_HideFields)
'=====

Private Sub OptionButton1_Click()
    With ActiveSheet.PivotTables(1).PivotFields("Artikel")
        If OptionButton1 = True Then
            .PivotItems("Früchte").Visible = False
            .PivotItems("Gemüse").Visible = True
            .PivotItems("Non-Food").Visible = True
        End If
    End With
End Sub

'=====
' Auf CD      Buchdaten\Beispiele\04_Auswertungen\
' Dateiname   04_08_PivotTable.xls
' Tabelle      04_HideFields
' Ereignis     Tabelle4(04_HideFields)
'=====

Private Sub OptionButton2_Click()
    With ActiveSheet.PivotTables(1).PivotFields("Artikel")
        If OptionButton2 = True Then
            .PivotItems("Gemüse").Visible = False
            .PivotItems("Früchte").Visible = True
            .PivotItems("Non-Food").Visible = True
        End If
    End With
End Sub

'=====
' Auf CD      Buchdaten\Beispiele\04_Auswertungen\
' Dateiname   04_08_PivotTable.xls
' Tabelle      04_HideFields
' Ereignis     Tabelle4(04_HideFields)
'=====

Private Sub OptionButton3_Click()
    With ActiveSheet.PivotTables(1).PivotFields("Artikel")
        If OptionButton3 = True Then
            .PivotItems("Non-Food").Visible = False
            .PivotItems("Früchte").Visible = True
            .PivotItems("Gemüse").Visible = True
        End If
    End With
End Sub

'=====
' Auf CD      Buchdaten\Beispiele\04_Auswertungen\
' Dateiname   04_08_PivotTable.xls
' Tabelle      04_HideFields
' Ereignis     Tabelle4(04_HideFields)
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

```
' =====

Private Sub OptionButton4_Click()
    With ActiveSheet.PivotTables(1).PivotFields("Artikel")
        If OptionButton4 = True Then
            .PivotItems("Non-Food").Visible = True
            .PivotItems("Früchte").Visible = True
            .PivotItems("Gemüse").Visible = True
        End If
    End With
End Sub
```

126 Pivot-Tabellen berechnen

Bis jetzt haben wir in allen Rezepten die Summe der Daten berechnen lassen. Sie können jedoch auch andere Berechnungen vornehmen, wie zum Beispiel Anzahl, Mittelwert usw.

Damit schnell und einfach zwischen verschiedenen Berechnungsmöglichkeiten gewechselt werden kann, haben wir in unserem Tabellenblatt in der Zelle I8 eine Gültigkeit (DATEN | GÜLTIGKEIT) in Form einer Liste erstellt. Die Liste umfasst drei verschiedene Berechnungsmöglichkeiten (Summe, Anzahl und Mittelwert). Über das Zellen-DropDown können Sie bequem die gewünschte Berechnung auswählen. Bei jeder Zellenänderung wird das hinterlegte Worksheet_Change-Ereignis der Tabelle ausgelöst. Mehr zum Thema Ereignis-Prozeduren erfahren Sie in der Kategorie »Ereignisse«.

Zu Beginn der Prozedur wird festgelegt, dass nur bei Veränderung in der Zelle I8 die ausgelöste Ereignisprozedur weiter ausgeführt werden soll. Wenn es sich um eine andere Zelle handelt, wird die Prozedur verlassen.

Innerhalb der With-Anweisung wird geprüft, welche Auswahl zur Berechnung des Pivot-Feldes getroffen wurde. Entsprechend der Auswahl der Berechnung wird jeweils eine andere Konstante verwendet. Um die Ereignis-Prozedur bei der Berechnung nicht in einer Endlosschleife enden zu lassen, werden wir die Eigenschaft EnableEvents vor der With-Anweisung deaktivieren und danach wieder aktivieren.

```
' =====
' Auf CD      Buchdaten\Beispiele\04_Auswertungen\
' Dateiname   04_08_PivotTable.xls
' Tabelle     04_Calculate
' Ereignis    Tabelle5(04_Calculate)
' =====

Private Sub Worksheet_Change(ByVal Target As Range)
    If Target.Address <> "$I$8" Then Exit Sub

    Application.EnableEvents = False
    With Me.Range("I1")
        If Range("I8").Value = "Summe" Then
            .PivotField.Function = xlSum
        ElseIf Range("I8").Value = "Anzahl" Then
            .PivotField.Function = xlCount
        ElseIf Range("I8").Value = "Mittelwert" Then
```

```
        .PivotField.Function = xlAverage
    End If
End With
Application.EnableEvents = True
End Sub
```

Die Konstanten, die Sie in der nachfolgenden Tabelle finden, beziehen sich auf die möglichen Berechnungsmethoden in Pivot-Tabellen.

Konstante	Index	Beschreibung
xlAverage	-4106	Mittelwert
xlCount	-4112	Anzahl
xlCountNums	-4113	Anzahl Zahlen
xlMax	-4136	Maximum
xlMin	-4139	Minimum
xlProduct	-4149	Produkt
xlStDev	-4155	Standardabweichung (Stichprobe)
xlStDevP	-4156	Standardabweichung (Grundgesamtheit)
xlSum	-4157	Summe
xlUnknown	1000	Unbekannt
xlVar	-4164	Varianz (Stichprobe)
xlVarP	-4165	Varianz (Grundgesamtheit)

Tabelle 29: Berechnungsmöglichkeiten in einer Pivot-Tabelle

127 Sortieren nach Farben

In Excel haben Sie standardmäßig lediglich die Möglichkeit, Zellen und Bereiche nach Buchstaben oder Werten zu sortieren. Oftmals werden jedoch auch Farben als Hintergrund oder Schriftfarbe verwendet und es könnte sinnvoll sein, nach dem Farbindex oder nach den RGB-Farben zu sortieren.

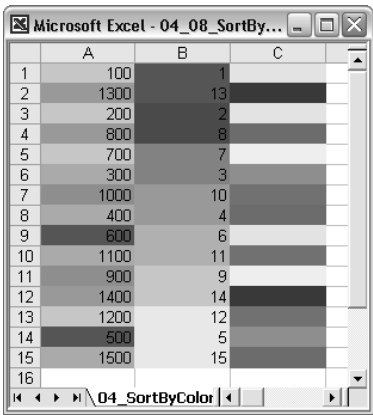


Abbildung 104: Sortieren nach Farben

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

Die folgende Prozedur ist so aufgebaut, dass die Hintergrundfarben eines Bereichs nach dem RGB-Farbwert sortiert werden können. Es muss lediglich eine Zelle in der zu sortierenden Spalte selektiert werden. Die Prozedur kann dann gestartet werden. Sie können zu Beginn der Prozedur wählen, ob auf- oder absteigend sortiert werden soll. Voraussetzung für ein Sortieren nach Farben ist, dass Farben vorhanden sind und die Zellen nicht leer sind. Eine entsprechende Prüfung wird im Code vorgenommen. Beim Nichtvorhandensein von Werten oder Zahlen wird eine entsprechende Meldung ausgegeben.

Auch ein Sortieren nach dem Farbindex ist möglich. Beachten Sie dazu den Kasten »Tipp« am Ende des Rezeptes.

Im ersten Block der Prozedur wird über einen Eingabedialog `InputBox` ermittelt, ob auf- oder absteigend sortiert werden soll. Bei Eingabe des Buchstabens »a« wird aufsteigend, bei »u« absteigend sortiert. Bei falscher Eingabe durch den Benutzer wird die Prozedur verlassen. Je nach Wert der `InputBox` wird die Konstante `xlAscending` oder `xlDescending` an die Variable `strSortOrder` übergeben. Am Ende der Prozedur wird diese Variable für den Sortiervorgang verwendet.

Im zweiten Code-Block, das heißt vor der Schleife, wird der Index der aktiven Spalte an die Variable `intCol` übergeben. Wir werden diese Variable später vermehrt in unserer Prozedur verwenden. Danach wird rechts der aktiven Spalte eine neue Spalte temporär eingefügt. In diese Spalte werden später die ermittelten Farbwerte der überprüften Zellen geschrieben, nach denen sortiert werden soll. Die Spalte wird ausgeblendet und nach dem Sortiervorgang gelöscht. Zugegeben, das Ausblenden könnte man sich ersparen, aber es ist einfach schöner, wenn nicht zu sehen ist, dass eine Zusatzspalte verwendet wird.

In der darauf folgenden `For`-Schleife wird geprüft, wie viele Zeilen im benutzten Bereich liegen. Entsprechend oft wird die Schleife durchlaufen. Innerhalb der Schleife wird in einer `If`-Entscheidung geprüft, ob die Zelle eine Hintergrundfarbe enthält. Wenn dies zutrifft, wird der Farbwert in die Nebenzelle der temporären, neu erstellten Spalte geschrieben. Der Zähler `intColor` wird bei jedem Treffer um den Wert 1 erhöht. Sollten keine Farben gefunden werden, wird nach der Schleife eine entsprechende Meldung ausgegeben.

In einer zweiten `If`-Entscheidung wird ein weiterer Zähler `intValue` verwendet. Er prüft, ob Werte in der Farbspalte enthalten sind. Bei Zutreffen wird der Zähler um den Wert 1 erhöht.

Nach der Schleife wird geprüft, ob die beiden Zähler leer sind. Wenn ja, bedeutet das, dass entweder keine Farben oder keine Werte in der zu sortierenden Spalte enthalten sind. Das würde bedeuten, dass die Spalte nicht sortiert werden kann. Die Hilfsspalte wird dann gelöscht und die Prozedur verlassen. Ansonsten wird die Spalte nach dem Farbwert sortiert. Je nachdem, welche Auswahl zu Beginn der Prozedur getroffen wurde, wird die Spalte auf- oder absteigend sortiert. Die Spalte wird nach dem Sortiervorgang gelöscht.

```
'=====
' Auf CD      Buchdaten\Beispiele\04_Auswertungen\
' Dateiname   04_09_SortByColor.xls
' Tabelle     04_SortByColor
' Modul       mdl_01_SortByColor
'=====
```

```
Sub SortByColor()
    Dim intRow As Integer
    Dim intCol As Integer
    Dim strSortOrder As String
```

```

Dim intColor As Integer
Dim intValue As Integer

' Rückfragen, ob auf- oder absteigend sortiert werden soll
' Bei falscher Eingabe wird die Prozedur abgebrochen
strSortOrder = Application.InputBox _
    ("""a"" für absteigend" & Chr(10) & _
    """"u"" für aufsteigend eingeben", _
    Default:="a")
If strSortOrder = "a" Then
    strSortOrder = xlAscending
ElseIf strSortOrder = "u" Then
    strSortOrder = xlDescending
Else
    MsgBox ("Sie haben nicht ""a"" oder ""u"" eingegeben, " & _
        "die Prozedur wird abgebrochen")
    Exit Sub
End If

' Eine Spalte für den Farbindex einfügen
' Am Ende der Prozedur wird diese Spalte wieder gelöscht
intCol = ActiveCell.Column
Columns(intCol).Insert Shift:=xlToRight
Columns(intCol).Hidden = True

For intRow = 1 To ActiveSheet.UsedRange.Rows.Count
    ' Prüfen, ob Farben in der Spalte enthalten sind
    If Cells(intRow, intCol + 1).Interior.Color <> 0 Then

        ' Den Farbindex in die neu erstellte Spalte schreiben
        Cells(intRow, intCol).Value = _
            Cells(intRow, intCol + 1).Interior.Color
        intColor = intColor + 1

        ' Prüfen, ob Werte in den Spalten enthalten sind
        If Cells(intRow, intCol + 1).Value <> "" Then
            intValue = intValue + 1
        End If
    End If
Next intRow

' Wenn die beiden Zähler leer sind, dann die Prozedur verlassen
If intColor = 0 Or intValue = 0 Then
    MsgBox "Es wurden keine Werte oder keine Farben " & _
        "in dieser Spalte gefunden."
    Columns(intCol).Delete
    Exit Sub
Else
    ' Die neue Spalte nach dem Farbindex sortieren,
    ' Hilfsspalte löschen und A1 selektieren
    Cells(1, intCol).Sort _
        Key1:=Columns(intCol), _
        Order1:=strSortOrder

```

Grund-
lagenAllge-
meinDatum
Zeit

tungen

Steuer
elementeBefehl
leisten

Objekt

Dia-
grammEreig-
nisseUser-
FormWeb/
Mail

Extern

Ge-
misch-
tes

Specia

```

        Columns(intCol).Delete
    End If
End Sub

```

Tipp

Wenn Sie nach dem Farbindex und nicht nach der RGB-Farb-Reihenfolge sortieren möchten, dann verwenden Sie an Stelle von `Interior.Color` das `Interior.ColorIndex`. Es kommt im Code zweimal vor.

In der folgenden Codezeile ersetzen Sie die 0 (Null) durch -4142. Dies entspricht einer Zelle ohne Farbe.

```
If Cells(intRow, intCol + 1).Interior.Color <> 0 Then
```

128 Sortieren nach selbst festgelegter Farbfolge

Diese Prozedur baut auf dem vorangegangenen Rezept auf. Diesmal wollen wir die Sortierreihenfolge jedoch selbst bestimmen. In unserem Tabellenblatt in der Spalte A legen wir fest, in welcher Farbreihenfolge wir sortieren möchten. Wir hinterlegen den Zellen in der Spalte A die verwendeten Farben. In die farbigen Zellen schreiben wir die gewünschte Farbreihenfolge, z.B. 1 bis 4 (siehe Abbildung 105).

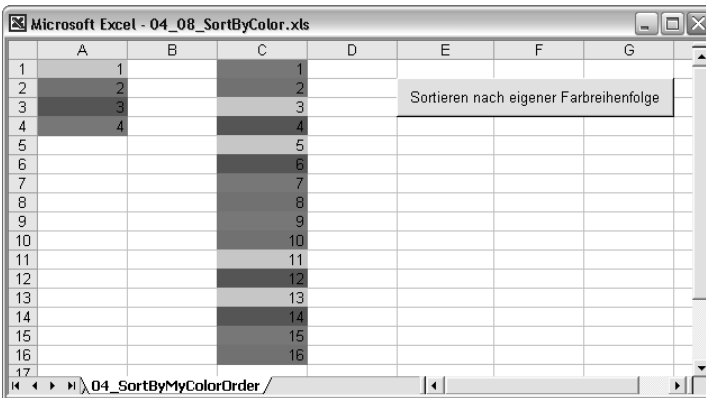


Abbildung 105: Sortieren nach eigener Farbreihenfolge

Im ersten Teil der Prozedur wird abgefragt, ob auf- oder absteigend sortiert werden soll. Danach wird eine neue Spalte eingefügt und ausgeblendet.

Nun beginnt der Kern der Prozedur. In einer `If`-Entscheidung wird ermittelt, ob die ausgewählte Spalte nicht die Spalte A ist, denn nach der Spalte A soll nicht sortiert werden, sondern nach der Spalte C. Die Spalte A enthält die festgelegte Sortierreihenfolge und soll unverändert bleiben.

In der äußeren `For`-Schleife wird geprüft, wie viele benutzte Zeilen die Tabelle enthält. Entsprechend oft wird die Schleife durchlaufen. In der inneren `For`-Schleife wird geprüft, nach wie vielen Farben sortiert werden soll. Es werden die benutzten Zellen in der Spalte A gezählt.

In der ersten `If`-Entscheidung wird sichergestellt, dass nur eingefärbte Zellen eine Nummer für die Sortierung erhalten. In der zweiten `If`-Entscheidung wird jede zu sortierende Zelle mit den Zellen in der Spalte A verglichen. Der entsprechende Wert (Sortierindex) wird in die neu erstellte Spalte geschrieben.

Der Rest der Prozedur verhält sich wie im vorangegangenen Rezept. Es wird je nach Auswahl zu Beginn der Prozedur auf- oder absteigend nach dem Sortierindex sortiert.

Grundlagen

```
'=====
' Auf CD      Buchdaten\Beispiele\04_Auswertungen\
' Dateiname  04_09_SortByColor.xls
' Tabelle    04_SortByMyColorOrder
' Modul      mdl_02_SortByMyColorOrder
'=====

Sub SortByMyColorOrder()
    Dim intRow As Integer
    Dim intCol As Integer
    Dim strSortOrder As String
    Dim intColor As Integer
    Dim intValue As Integer
    Dim i As Integer

    strSortOrder = Application.InputBox _
        ("""a"" für absteigend" & Chr(10) & _
        """"u"" für aufsteigend eingeben", _
        Default:="a")
    If strSortOrder = "a" Then
        strSortOrder = xlAscending
    ElseIf strSortOrder = "u" Then
        strSortOrder = xlDescending
    Else
        MsgBox ("Sie haben nicht ""a"" oder ""u"" eingegeben, " & _
            "die Prozedur wird abgebrochen")
    End If
    Exit Sub

    intCol = ActiveCell.Column
    Columns(intCol).Insert Shift:=xlToRight
    Columns(intCol).Hidden = True

    ' Die zu sortierende Spalte darf nicht die Spalte "A" sein,
    ' denn diese enthält die Sortierreihenfolge
    If Left(ActiveCell.Address(False, False), 1) <> "A" Then
        i = 1
        For intRow = 1 To ActiveSheet.UsedRange.Rows.Count
            ' Prüfen, nach wie vielen Farben sortiert werden soll
            For i = 1 To Cells(Rows.Count, 1).End(xlUp).Row
                If Cells(intRow, intCol + 1).Interior.Color <> 0 Then

                    ' Den Sortierindex in die neue Spalte schreiben
                    ' und mit der eigenen Farbreihenfolge vergleichen
                    If Cells(intRow, intCol + 1).Interior.Color = _
                        Cells(i, 1).Interior.Color Then
                            Cells(intRow, intCol).Value = Cells(i, 1).Value
                        End If
                    End If
                End If
            End For
        End For
    End If
```

Algemein

Datum

Zeit

tungen

Steuer

elementen

Befehl

leisten

Objekt

Diagramm

Ereignisse

User-Form

Web/Mail

Extern

Gemischtes

Spezial

```

        intColor = intColor + 1
        If Cells(intRow, intCol + 1).Value <> "" Then
            intValue = intValue + 1
        End If
    End If
Next i
Next intRow
Else
    MsgBox "Die Spalte " & "A" & " kann enthält die Sortierfolge. " & _
        "Wählen Sie eine andere Spalte aus."
    Columns("A").Delete
    Exit Sub
End If

If intColor = 0 Or intValue = 0 Then
    MsgBox "Es wurden keine Werte oder keine Farben " & _
        "in dieser Spalte gefunden."
    Columns(intCol).Delete
    Exit Sub
Else
    Cells(1, intCol).Sort _
        Key1:=Columns(intCol), _
        Order1:=strSortOrder
    Columns(intCol).Delete
End If
End Sub

```

129 Tabellenblätter alphabetisch sortieren

Wenn in einer Excel-Arbeitsmappe viele Tabellen- und/oder Diagrammblätter vorhanden sind, ist es sehr mühsam, diese von Hand nach einer bestimmten Reihenfolge zu sortieren. In den nachfolgenden Beispielen zeigen wir Ihnen, worauf Sie beim Sortieren achten müssen. Zum einen ist es wichtig, um welche Art der Beschriftung es sich handelt. Buchstaben werden beim Sortiervorgang anders bewertet als Zahlen. Weiterhin gibt es die Möglichkeit, die Registerreiter nach ihrer Beschriftung auf- oder absteigend zu sortieren.

Die Prozedur `TabSortAlphabetic` kann die Arbeitsblattregister sowohl in auf- als auch in absteigender Reihenfolge sortieren. Die jeweilige Anweisung `Up` oder `Down` wird im Prozedurkopf an das Argument `strSortOrder` übergeben. Beim Aufruf der Prozedur ist es zwingend erforderlich, eine dieser beiden Konstanten anzugeben. Folgende Prozedur sortiert die Registerreiter der aktiven Arbeitsmappe in aufsteigender Reihenfolge (siehe Abbildung 106).

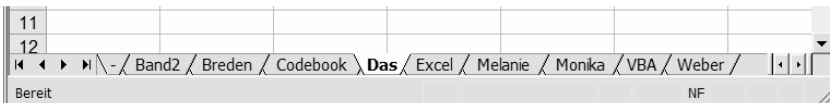


Abbildung 106: Aufsteigende alphabetische Sortierung

```

' =====
' Auf CD      Buchdaten\Beispiele\04_Auswertungen\
' Dateiname   04_10_SortSheetsAlphabetic.xls

```



```
' Tabelle      Das
' Modul        mdl_01_TabSortAlphabetic
' =====

Sub SortSheetsAlphabeticUp()
    TabSortAlphabetic "Up"
End Sub
```

Ebenso können Sie den Aufruf der Prozedur mit der `Call`-Anweisung durchführen. Dabei muss die übergebene Konstante in Klammern gesetzt werden. Diese beiden Prozeduren wurden in der Beispieldatei je einer Schaltfläche zugewiesen. Geben Sie beim Prozeduraufruf die Zeichenkette `Down` an, damit die Registerreiter in absteigender Reihenfolge sortiert werden (siehe Abbildung 107).

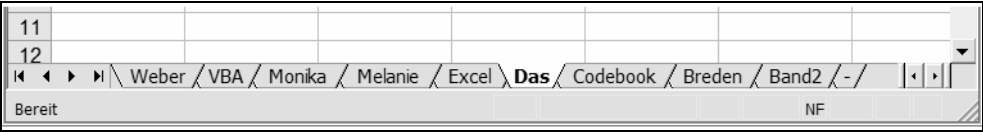


Abbildung 107: Absteigende alphabetische Sortierung

```
' =====
' Auf CD      Buchdaten\Beispiele\04_Auswertungen\
' Dateiname   04_10_SortSheetsAlphabetic.xls
' Tabelle     Das
' Modul       mdl_01_TabSortAlphabetic
' =====

Sub SortSheetsAlphabeticDown()
    Call TabSortAlphabetic("Down")
End Sub
```

Wenn Sie ein anderes Argument als `Up` oder `Down` an den Parameter `strSortOrder` übergeben, kann keine Sortierung durchgeführt werden. In einer `If`-Anweisung wird die Prozedur mit `Exit Sub` verlassen, wenn eine andere Zeichenkette übergeben wurde.

Schauen wir uns zunächst die verwendeten Variablen und deren Deklaration an. Diese weichen in den anderen Sortierprozeduren nur unwesentlich voneinander ab.

Beim späteren Sortiervorgang werden einzelne Blätter innerhalb der Blattregister verschoben. Dabei werden diese zwangsweise aktiviert. Dies führt dazu, dass das zu Beginn der Prozedur aktive Blatt nach der Sortierung womöglich nicht mehr aktiv ist, sondern das zuletzt verschobene Arbeitsblatt. Um im Anschluss an die Sortierung das anfänglich aktive Blatt wieder anzuzeigen, setzen wir vor der Sortierung mit der `Set`-Anweisung einen Verweis auf dieses Blatt. Dazu verwenden wir die Variable `objSh` vom Datentyp `Objekt`. Wenn Sie stattdessen den Datentyp `Worksheet` verwenden, würde Excel eine Fehlermeldung ausgeben, wenn es sich bei dem aktiven Blatt um ein Diagrammblatt handelt.

Der Variablen `intSh` wird über die `Count`-Eigenschaft der `Sheets`-Auflistung die Anzahl der vorhandenen Arbeitsblätter zugewiesen. Hier wird wiederum das `Sheets`- statt des `Worksheets`-Objektes angewandt, damit sowohl Tabellen- als auch Diagrammblätter in die Auflistung einbezogen werden.

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

Die Variable `intFirst` wird in der ersten `For...Next`-Anweisung verwendet, um alle Arbeitsblätter der Datei zu überprüfen. In der zweiten `For...Next`-Anweisung wird durch `intSecond` jedes Blatt der Datei mit dem Blatt `intFirst` der ersten Schleife verglichen.

Da das Verschieben der Arbeitsblätter mit VBA normalerweise am Bildschirm sichtbar ist, schalten wir die Bildschirmaktualisierung zu Beginn der Prozedur über die `ScreenUpdating`-Eigenschaft aus und abschließend mit `True` wieder ein.

In einer `Select Case`-Anweisung werden in den beiden `Case`-Abschnitten die möglichen Sortierkonstanten `Up` und `Down` auf den Ausdruck `True` überprüft. Die Entscheidung der Sortierreihenfolge könnte ebenso in einer `If...Then...Else`-Anweisung erfolgen.

Die `UCase`-Funktion wandelt die Zeichenfolge der jeweiligen Registerbeschriftung in Großbuchstaben um. Dadurch ist ein eindeutiger Vergleich der beiden zu überprüfenden Beschriftungen möglich. Durch den Größer/kleiner-Vergleich der beiden Registerbeschriftungen wird das Blatt `intSecond` vor `intFirst` verschoben oder nicht.

Das Prinzip der Sortierung ist bei allen nachfolgenden Prozeduren zur Tabellensortierung identisch.

```
'=====
' Auf CD      Buchdaten\Beispiele\04_Auswertungen\
' Dateiname   04_10_SortSheetsAlphabetic.xls
' Tabelle     Das
' Modul       mdl_01_TabSortAlphabetic
'=====
```

```
Sub TabSortAlphabetic(strSortOrder As String)
    Dim objSh As Object
    Dim intSh As Integer
    Dim intFirst As Integer
    Dim intSecond As Integer

    Set objSh = ActiveSheet
    intSh = ActiveWorkbook.Sheets.Count

    If Not strSortOrder = "Up" And _
        Not strSortOrder = "Down" Then Exit Sub

    Application.ScreenUpdating = False

    For intFirst = 1 To intSh
        For intSecond = intFirst To intSh

            Select Case True
                Case strSortOrder = "Up"
                    If UCase(Sheets(intSecond).Name) _
                        < UCase(Sheets(intFirst).Name) Then
                        Sheets(intSecond).Move Before:=Sheets(intFirst)
                    End If
                Case strSortOrder = "Down"
                    If UCase(Sheets(intSecond).Name) _
                        > UCase(Sheets(intFirst).Name) Then
                        Sheets(intSecond).Move Before:=Sheets(intFirst)
                    End If
            End Select
        Next intSecond
    Next intFirst

    Application.ScreenUpdating = True
End Sub
```

```
End Select

Next intSecond
Next intFirst

objSh.Activate
Application.ScreenUpdating = True
End Sub
```

Grund-
lagen

Allge-
mein

Datu
Zeit

130 Tabellenblätter nach Codenamen sortieren

Jedes Arbeitsblatt einer Excel-Arbeitsmappe hat zwei Namen, über welche die Blätter angesprochen und identifiziert werden können. Zum einen den Registernamen, welcher in der Excel-Ansicht auf den Registerbeschriftungen abgelesen werden kann. Im VBA-Explorer gibt es zu jedem Registernamen einen Codenamen. Er wird vor dem in Klammern stehenden Registernamen angezeigt (siehe Abbildung 108).

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

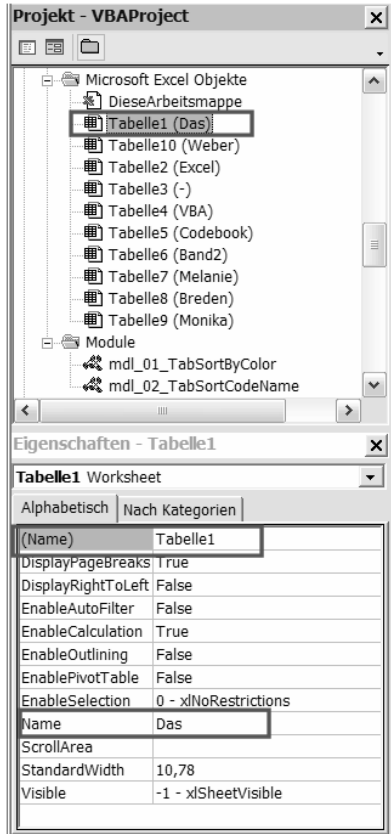


Abbildung 108: Register- und Codenamen der Arbeitsblätter

Mit der `CodeName`-Eigenschaft wird das Arbeitsblatt über seinen VBA-Namen angesprochen. Wenn Sie nun die alphabetische Sortierung der Codenamen aus Rezept 127 anwenden würden,

wäre die Reihenfolge genau die gleiche wie im Projekt-Explorer (siehe Abbildung 108). Das Blatt TABELLE10 würde sich zwischen TABELLE1 und TABELLE2 einreihen (siehe Abbildung 109).

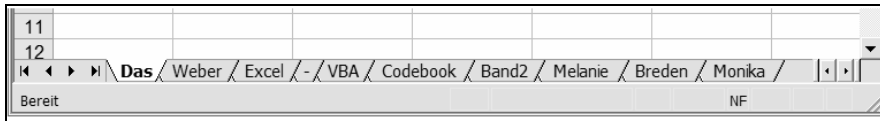


Abbildung 109: Falsche Sortierreihenfolge über den Codenamen der Arbeitsblätter

Die Arbeitsblätter müssen demzufolge nicht über die Zeichenkette *Tabelle1*, sondern nur über die extrahierte Ziffer *1* ausgewertet und sortiert werden. Die `Replace`-Funktion ersetzt in unserer Prozedur die Zeichenfolge *Tabelle* des Strings *Tabelle1* durch einen Leerstring `" "`. Das Resultat ist der gewünschte numerische Wert *1* des Codenamens. Die auf diese Weise extrahierten Codeblattnummern werden in den Variablen `intDummy1` und `intDummy2` gespeichert.

Damit nicht die falsche Zahlenfolge *1,10,2,...* erfolgt, setzen wir die `Val`-Funktion ein. Dadurch werden Zahlen nach ihrem numerischen Wert behandelt und eine folgerichtige Sortierung *1,2,...,9,10* erreicht (siehe Abbildung 110). Zum Abschluss der Prozedur wird das erste Arbeitsblatt aktiviert.

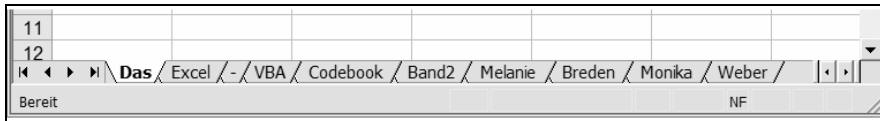


Abbildung 110: Richtige Sortierreihenfolge über den Codenamen der Arbeitsblätter

```
' =====
' Auf CD      Buchdaten\Beispiele\04_Auswertungen\
' Dateiname   04_10_SortSheetsAlphabetic.xls
' Tabelle     Das
' Modul       mdl_02_TabSortCodeName
' =====
```

```
Sub TabSortCodeName()
    Dim objSh As Object
    Dim intSh As Integer
    Dim intFirst As Integer
    Dim intSecond As Integer
    Dim intDummy1 As Integer
    Dim intDummy2 As Integer

    Set objSh = ActiveSheet
    intSh = ActiveWorkbook.Sheets.Count

    Application.ScreenUpdating = False

    For intFirst = 1 To intSh
        For intSecond = intFirst To intSh

            intDummy1 = Replace(Sheets(intFirst).CodeName, _
                               "Tabelle", " ")
```

```
intDummy2 = Replace(Sheets(intSecond).CodeName, _
                    "Tabelle", "")

If Val(intDummy2) < Val(intDummy1) Then
    Sheets(intSecond).Move Before:=Sheets(intFirst)
End If

Next intSecond
Next intFirst

Sheets(1).Activate
Application.ScreenUpdating = True
End Sub
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

131 Tabellenblätter numerisch sortieren

Um Arbeitsblätter einer Excel-Arbeitsmappe nach ihrer numerischen Registerbeschriftung zu sortieren, verwenden wir fast die gleiche Prozedur wie zur alphabetischen Sortierung aus Rezept 4.21. Der einzige Unterschied besteht darin, dass die Registerbeschriftung nicht mittels der UCase-Funktion in Großbuchstaben, sondern mit der Val-Funktion von Textwerten in numerische Werte umgewandelt wird.

```
'=====
' Auf CD      Buchdaten\Beispiele\04_Auswertungen\
' Dateiname  04_11_SortSheetsNumeric.xls
' Tabelle    1
' Modul      mdl_01_TabSortNumeric
'=====

Sub SortSheetsNumericUp()
    TabSortNumeric "Up"
End Sub

Sub SortSheetsNumericDown()
    TabSortNumeric "Down"
End Sub

Sub TabSortNumeric(strSortOrder As String)
    Dim objSh As Object
    Dim intSh As Integer
    Dim intFirst As Integer
    Dim intSecond As Integer

    Set objSh = ActiveSheet
    intSh = ActiveWorkbook.Sheets.Count

    If Not strSortOrder = "Up" And _
       Not strSortOrder = "Down" Then Exit Sub

    Application.ScreenUpdating = False
```

```

For intFirst = 1 To intSh
    For intSecond = intFirst To intSh

        Select Case True
            Case strSortOrder = "Up"
                If Val(Sheets(intSecond).Name) _
                    < Val(Sheets(intFirst).Name) Then
                    Sheets(intSecond).Move Before:=Sheets(intFirst)
                End If
            Case strSortOrder = "Down"
                If Val(Sheets(intSecond).Name) _
                    > Val(Sheets(intFirst).Name) Then
                    Sheets(intSecond).Move Before:=Sheets(intFirst)
                End If
        End Select

    Next intSecond
Next intFirst

objSh.Activate
Application.ScreenUpdating = True
End Sub

```

132 Tabellenblätter nach Farben sortieren

Ab Excel 2002 ist es über das Kontextmenü der Registerreiter möglich, die Registerreiter farblich zu kennzeichnen (siehe Abbildung 111).

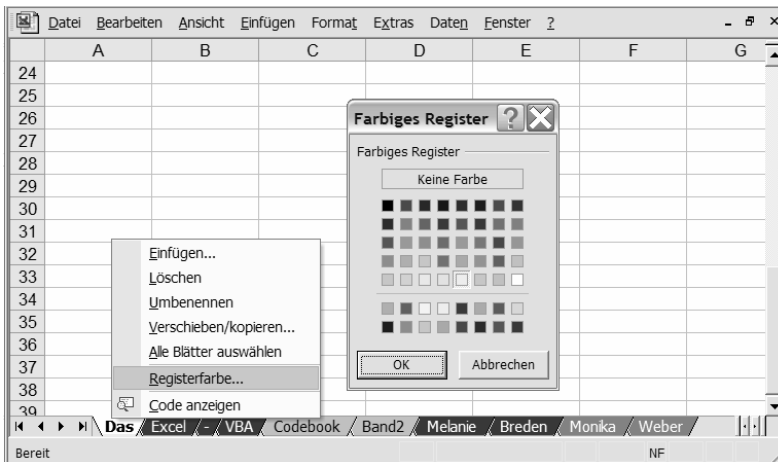


Abbildung 111: Farbliche Kennzeichnung der Registerreiter

Die Farben der Registerreiter werden über die `ColorIndex`-Eigenschaft ausgelesen bzw. zugewiesen. Die Sortierung erfolgt nun entsprechend des numerischen `ColorIndex`-Wertes in auf- bzw. absteigender Reihenfolge. Der Ablauf der Prozedur unterscheidet sich weiterhin nicht von den vorangegangenen Beispielen.

```
'=====
' Auf CD      Buchdaten\Beispiele\04_Auswertungen\
' Dateiname   04_12_SortSheetsByColor.xls
' Tabelle     Das
' Modul       mdl_01_TabSortByColor
'=====

Sub SortSheetsColorUp()
    TabSortColor "Up"
End Sub

Sub SortSheetsColorDown()
    TabSortColor "Down"
End Sub

'=====
' Auf CD      Buchdaten\Beispiele\04_Auswertungen\
' Dateiname   04_12_SortSheetsByColor.xls
' Tabelle     Das
' Modul       mdl_01_TabSortByColor
'=====

Sub TabSortColor(strSortOrder As String)
    Dim objSh As Object
    Dim intSh As Integer
    Dim intFirst As Integer
    Dim intSecond As Integer

    Set objSh = ActiveSheet
    intSh = ActiveWorkbook.Sheets.Count

    If Not strSortOrder = "Up" And _
        Not strSortOrder = "Down" Then Exit Sub

    Application.ScreenUpdating = False

    For intFirst = 1 To intSh
        For intSecond = intFirst To intSh

            Select Case True
                Case strSortOrder = "Up"
                    If Val(Sheets(intSecond).Tab.ColorIndex) _
                        < Val(Sheets(intFirst).Tab.ColorIndex) Then
                        Sheets(intSecond).Move Before:=Sheets(intFirst)
                    End If
                Case strSortOrder = "Down"
                    If Val(Sheets(intSecond).Tab.ColorIndex) _
                        > Val(Sheets(intFirst).Tab.ColorIndex) Then
                        Sheets(intSecond).Move Before:=Sheets(intFirst)
                    End If
            End Select
        End Select
    End For
End Sub
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

```
        Next intSecond
    Next intFirst

    objSh.Activate
    Application.ScreenUpdating = True
End Sub
```

Steuerelemente

Steuerelemente sind Grafikobjekte, die auf einem Excel-Arbeitsblatt oder einem Formular User-Form platziert werden können.

Tipp Die Steuerelemente aus der Werkzeugsammlung der Formulare werden in der Kategorie »UserForm« anhand von einzelnen Rezepten ausführlich beschrieben.

Diese Objekte können aus Textfeldern, Listefeldern, Optionsfeldern, Befehlsschaltflächen und anderen Elementen bestehen. Steuerelemente stellen Benutzern Optionen, Auswahllisten oder Schaltflächen, auf die geklickt werden kann, zur Verfügung. Eine Übersicht aller Steuerelemente finden Sie in Tabelle 30. In der Liste der integrierten Symbolleisten im Menü ANSICHT | SYMBOLLEISTEN befinden sich zwei Leisten mit Steuerelementen:

- Formular-Symbolleiste
- Steuerelement-Toolbox-Symbolleiste

In der heutigen VBA-Programmierung verwendet man meist die ACTIVEX-Steuerelemente aus der STEUERELEMENT-TOOLBOX-Symbolleiste. Sie können ActiveX-Steuerelemente verwenden, wenn Sie verschiedene Ereignisse steuern müssen, die eintreten, wenn das Steuerelement verwendet wird. Beispielsweise können Sie einem Arbeitsblatt ein LISTENFELD-Steuerelement hinzufügen und anschließend ein Makro für andere Vorgänge erstellen, in Abhängigkeit davon, welche Option der Benutzer aus der Liste auswählt.

Steuerelemente der FORMULAR-Symbolleiste sind kompatibel mit früheren Versionen von Excel (ab Excel Version 5.0) und können neben Tabellenblättern auch auf XLM-MAKROBLÄTTERN und DIAGRAMMBLÄTTERN verwendet werden. Elemente der FORMULAR-Symbolleiste verfügen nur über das Click-Ereignis, welche bei Klick darauf das zugewiesene Makro ausführen.

Hinweis In Diagrammblättern ist es nicht möglich, ACTIVEX-Steuerelemente aus der STEUERELEMENT-TOOLBOX-Symbolleiste anzuwenden. Verwenden Sie auf Diagrammblättern die Steuerelemente der FORMULAR-Symbolleiste.

Bei beiden Symbolleisten fügen Sie auf die gleiche Weise neue Steuerelemente in ein Tabellenblatt ein. Klicken Sie mit der linken Maustaste in einer der beiden Leisten auf ein Steuerelement-Symbol und anschließend auf das Tabellenblatt. Excel erstellt daraufhin das Element an der markierten Zellposition in seiner Standardgröße. Wenn Sie einen Doppelklick auf ein Steuerelement in einer Symbolleiste ausführen, können mehrere gleiche Elemente nacheinander auf dem Tabellenblatt eingefügt werden, ohne das Symbol des einzufügenden Steuerelementes jedes Mal drücken zu müssen. Durch nochmaliges Klicken auf das Symbol oder Drücken der [ESC]-Taste wird der Einfügemodus wieder aufgehoben.

Sie können aber auch nach der Auswahl auf der Symbolleiste das gewählte Element auf dem Tabellenblatt »aufziehen«, indem Sie die linke Maustaste gedrückt halten und durch die Mausbewegung einen Rahmen für die gewünschte Größe ziehen. Wenn Sie beim Aufziehen eines Steuer-

Grundlagen
Allgemein
Datum und Zeit
Formeln
Diagramme
Diagrammblättern
Ereignisse
User-Form
Web/Mail
Extern
Gemischtes
Spezial

elementes die **[Alt]**-Taste gedrückt halten, wird das Objekt an den nächstliegenden Zellen ausgerichtet.

Um zu ermitteln, ob ein Steuerelement ein ActiveX-Steuerelement oder ein Steuerelement der FORMULAR-Symbolleiste ist, klicken Sie mit der rechten Maustaste auf das Steuerelement. Wenn kein Kontextmenü angezeigt wird oder das Kontextmenü den Befehl EIGENSCHAFTEN enthält, dann handelt es sich um ein ActiveX-Steuerelement. Wenn das Kontextmenü den Befehl MAKRO ZUWEISEN enthält, handelt es sich um ein Steuerelement der Formulsymbolleiste.

133 Steuerelemente der Formular-Symbolleiste

Sie können ein Steuerelement der Symbolleiste FORMULAR verwenden, wenn Sie alle Makros für ein Formular aufzeichnen möchten, jedoch keinen Makrocode in VBA schreiben oder ändern möchten. Steuerelemente der Symbolleiste FORMULAR lassen sich auch auf Diagrammblättern anwenden. Im Gegensatz zu den ACTIVEX-Steuerelementen der STEUERELEMENT-TOOLBOX verfügen diese nur über ein **Beiklick**-Ereignis. Steuerelemente der Symbolleiste FORMULAR können nicht zur Steuerung von anderen Ereignissen verwendet werden, wie sie bei ActiveX-Elementen zur Verfügung stehen. Darüber hinaus können Steuerelemente der FORMULAR-Symbolleiste nicht verwendet werden, um Webskripts auf Webseiten auszuführen.

Auf der FORMULAR-Symbolleiste befinden sich einige Schaltflächen, welche nicht in den neueren Excel-Versionen zur Verfügung stehen und entsprechend ausgegraut sind (siehe Abbildung 112).



Abbildung 112: Formular-Symbolleiste

Aber es gibt auch einige interessante Funktionen (GRUPPENFELD und RASTER) auf dieser Leiste, welche sich nicht auf der STEUERELEMENT-TOOLBOX-Symbolleiste befinden. Nachfolgend eine Übersicht aller Symbole und Funktionen der FORMULAR-Symbolleiste.

Icon	Name	Typ	Beschreibung
	Beschriftung	Labels	Text, der Informationen zu einem Steuerelement, zum Arbeitsblatt oder zum Formular liefert
	Bearbeitungsfeld		Steht in Microsoft Excel-Arbeitsmappen nicht zur Verfügung. Dieses Steuerelement wird bereitgestellt, um es Ihnen zu ermöglichen, mit Dialogblättern zu arbeiten, die mit Excel Version 5.0 erstellt wurden.
	Gruppenfeld	GroupBoxes	Fasst zusammengehörende Steuerelemente zusammen, wie z.B. Optionsfelder oder Kontrollkästchen
	Schaltfläche	Buttons	Führt ein Makro aus, wenn darauf geklickt wird

Tabelle 30: Symbole der Formular-Symbolleiste

Icon	Name	Typ	Beschreibung
	Kontrollkästchen	CheckBoxes	Aktiviert oder deaktiviert eine Option. Sie können mehrere Kontrollkästchen auf einem Blatt oder in einer Gruppe gleichzeitig aktivieren.
	Optionsfeld	Option-Buttons	Wählt eine Option aus einer Gruppe von Optionen aus, die in einem Gruppenfeld enthalten sind. Verwenden Sie Optionsfelder, um nur eine von verschiedenen Möglichkeiten zuzulassen.
	Listenfeld	ListBoxes	Zeigt eine Liste mit Elementen an
	Kombinationsfeld	DropDowns	Ein Dropdown-Listenfeld. Das im Listenfeld ausgewählte Element wird im Textfeld angezeigt.
	Kombinationsfeld Liste-Text		Steht in Microsoft Excel-Arbeitsmappen nicht zur Verfügung
	Kombinationsfeld Dropdown-Text		Steht in Microsoft Excel-Arbeitsmappen nicht zur Verfügung
	Bildlaufleiste	ScrollBars	Blättert durch einen Wertebereich, wenn Sie auf die Bildlaufpfeile klicken oder wenn Sie das Bildlauffeld ziehen. Sie können durch eine Seite mit Werten blättern, indem Sie auf eine Stelle zwischen dem Bildlauffeld und einem Bildlaufpfeil klicken.
	Drehfeld	Spinners	Erhöht oder verringert einen Wert. Um den Wert zu erhöhen, klicken Sie auf den Aufwärtspfeil. Um den Wert zu verringern, klicken Sie auf den Abwärtspfeil.
	Steuerelement-eigenschaften		Ist auswählbar, wenn das Steuerelement erstellt wird oder markiert ist. Formatieren der Steuerelemente und Einstellung ändern.
	Code bearbeiten		Wechselt in das Codeblatt der Tabelle. Es wird eine Codeschablone mit Click-Ereignis des markierten Steuerelementes erstellt, oder zu der vorhandenen zugewiesenen Prozedur gesprungen.
	Raster		Blendet die Gitternetzlinien des aktiven Tabellenblattes ein bzw. aus
	Dialogfeld ausführen		Steht in Microsoft Excel-Arbeitsmappen nicht zur Verfügung

Tabelle 30: Symbole der Formular-Symbolleiste (Forts.)

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

St
elem
te

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

Sie können an ein Steuerelement der FORMULAR-Symbolleiste ein vorhandenes Makro über das Kontextmenü des Elementes Option MAKRO ZUWEISEN anbinden. Wenn ein Benutzer des Formulars auf das Steuerelement klickt, führt dieses das Makro aus. In vielen Rezepten dieses Buches werden Befehlsschaltflächen der FORMULAR-Symbolleiste verwendet, weil man diesen leicht eine bestehende Prozedur zuweisen kann und sie keine zusätzliche Ereignisprozedur im Tabellenblatt aufweisen müssen.

Nachdem Sie ein neues Steuerelement erstellt haben, wird anschließend automatisch der Dialog MAKRO ZUWEISEN angezeigt (siehe Abbildung 113). Hier können Sie entweder dem Objekt ein bestehendes Makro zuweisen oder über die Schaltfläche AUFZEICHNEN ein neues Makro erstellen.

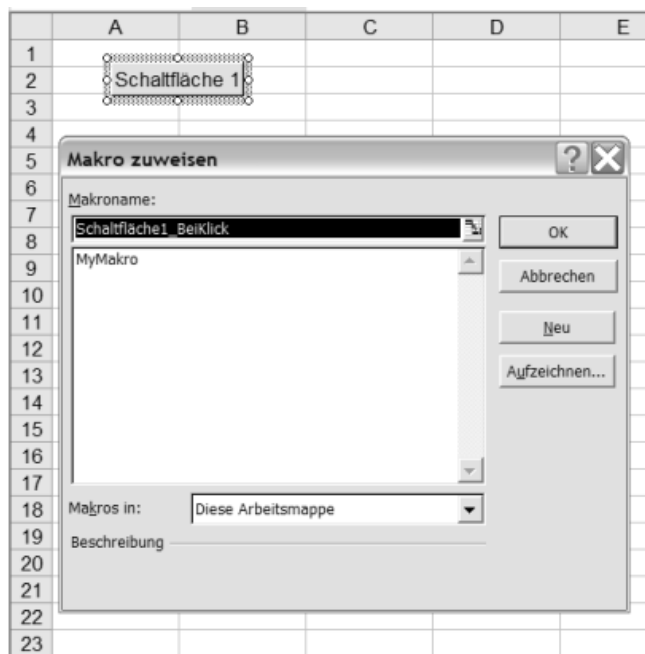


Abbildung 113: Einer Schaltfläche ein Makro zuweisen

Die meisten Steuerelemente besitzen Eigenschaften, die Sie ändern können; Sie können jedoch nur bei Befehlsschaltflächen die Formatierungseigenschaften der Schaltfläche selbst und die Beschriftung ändern. Klicken Sie dazu das Steuerelement mit der rechten Maustaste an und wählen Sie im Kontextmenü den Eintrag STEUERELEMENT FORMATIEREN.

Hinweis

Die ActiveX-Steuerelemente der STEUERELEMENT-TOOLBOX-Symbolleiste – mit ihren programmierbaren Ereignissen und verbesserten Eigenschaften – haben in der heutigen VBA-Programmierung eine höhere Priorität als die FORMULAR-Steuerelemente. Deswegen beschränken wir uns bei den FORMULAR-Controls auf einige kurze Beispiele.

134 Formular-Schaltfläche erstellen

Die folgende Prozedur erstellt auf dem aktiven Tabellenblatt genau über dem Zellenbereich G10:F11 eine Schaltfläche. Um ein Steuerelement an einer festen Position auf ihrem Tabellen-

blatt zu erstellen, übergeben Sie den Left-, Top-, Width- und Height-Eigenschaften feste Pixelwerte. Der Reihe nach legen diese Eigenschaften die Pixelwerte vom oberen und linken Bildschirmrand sowie die Breite und Höhe des Steuerelementes fest. Über die OnAction-Eigenschaft bestimmen Sie die Prozedur MyMakro, die ausgeführt wird, wenn der Anwender auf die Schaltfläche MyButton klickt. Wenn Sie die Prozedur mehrmals ausführen, wird jeweils die gleiche Schaltfläche mit gleich lautender Bezeichnung erstellt. Um andere Steuerelement-Typen der FORMULAR-Symbolleiste mit VBA zu erstellen, verwenden Sie den jeweiligen Typ aus der Übersichtstabelle Tabelle 30.

```
' =====  
' Auf CD      Buchdaten\Beispiele\05_Steuerelemente\  
' Dateiname   05_01_FormularControls.xls  
' Tabelle     Tabelle1  
' Modul       mdl_01_AddFormularButton  
' =====
```

```
Sub AddFormularButton()  
    Dim objButton As Object  
  
    With Range("G10:F11")  
        Set objButton = ActiveSheet.Buttons.Add( _  
                                Left:=.Left, Top:=.Top, _  
                                Width:=.Width, Height:=.Height)  
    End With  
  
    With objButton  
        .Name = "MyButton"  
        .OnAction = "MyMakro"  
        .Characters.Text = "Makro ausführen"  
        .Characters.Font.Name = "Arial"  
        .Characters.Font.Size = 12  
    End With  
End Sub  
  
Sub MyMakro()  
    MsgBox "Hallo " & Application.UserName  
End Sub
```

135 Formular-Schaltfläche löschen

Um ein Steuerelement der FORMULAR-Symbolleiste zu löschen, verwenden Sie den Namen der Schaltfläche. Existiert die Schaltfläche im angegebenen Tabellenblatt nicht, gibt Excel eine Fehlermeldung aus. Dies können Sie mit einer vorangestellten Fehlerroutine verhindern.

```
On Error Resume Next  
ActiveSheet.Shapes("MyButton").Delete
```

In unserer Prozedur werden alle Objekte mittels einer For Each...Next-Anweisung durchlaufen. Wenn ein Objekt den im Code angegebenen Namen aufweist, wird es gelöscht und die Schleife verlassen. Dadurch erübrigt sich die beschriebene Fehlerroutine.

Grundlagen

Allgemein

Datum
Zeit

tungen

St
elem
te

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

```
'=====
' Auf CD      Buchdaten\Beispiele\05_Steuerelemente\
' Dateiname  05_01_FormularControls.xls
' Modul      mdl_01_AddFormularButton
'=====

Sub DeleteFormularButton()
    Dim shButton As Shape

    For Each shButton In ActiveSheet.Shapes
        If shButton.Name = "MyButton" Then
            shButton.Delete
            Exit For
        End If
    Next shButton
End Sub
```

136 Makro an Formular-Schaltfläche zuweisen

Mit VBA können Sie einer bestehenden Schaltfläche leicht eine neue Prozedur zuweisen. In dem folgenden Beispiel wird der ersten Schaltfläche im aktiven Tabellenblatt die Prozedur `MyNewMakro` zugewiesen. Diese wird anstelle der vorher zugewiesenen Prozedur `MyMakro` bei Klick auf die Schaltfläche ausgeführt.

```
'=====
' Auf CD      Buchdaten\Beispiele\05_Steuerelemente\
' Dateiname  05_01_FormularControls.xls
' Modul      mdl_02_NewOnAction
'=====

Sub NewOnAction()
    ActiveSheet.Shapes(1).OnAction = "MyNewMakro"
End Sub

Sub MyNewMakro()
    MsgBox "Hallo Welt"
End Sub
```

137 Steuerelement-Toolbox

Die STEUERELEMENT-TOOLBOX ist eine Symbolleiste, die ACTIVE-X-Steuerelemente zur Verfügung stellt.



Abbildung 114: Steuerelement-Toolbox-Symbolleiste

ActiveX-Steuerelemente sind mit Steuerelementen vergleichbar, die in Programmiersprachen (z.B. Microsoft Visual Basic) verwendet werden. Hierbei handelt es sich um dieselben Steuerelemente, die Sie benutzerdefinierten Formularen im Visual Basic Editor hinzufügen können. Wenn

Sie einem Arbeitsblatt ein ActiveX-Steuerelement hinzufügen, schreiben Sie Makrocode, der sich auf die ID-Nummer des Steuerelementes bezieht, und weisen nicht ein Makro zu, das ausgeführt werden soll, wenn auf das Steuerelement geklickt wird. Wenn ein Benutzer des Formulars das Steuerelement verwendet, wird der von Ihnen angegebene Code ausgeführt, um auftretende Ereignisse zu verarbeiten.

Achtung

Sie können ActiveX-Steuerelemente nicht auf Diagrammblättern benutzen. Verwenden Sie dazu die Steuerelemente aus der FORMULAR-Symbolleiste.

Einige dieser Steuerelemente sind den Steuerelementen auf der FORMULAR-Symbolleiste ähnlich. Andere dagegen, wie beispielsweise Umschaltflächen und Bild-Steuerelemente, sind auf der FORMULAR-Symbolleiste nicht verfügbar. Die STEUERELEMENT-TOOLBOX enthält zusätzlich angepasste ActiveX-Steuerelemente, die von anderen Programmen installiert werden, z.B. den Microsoft Webbrowser, der von Microsoft Internet Explorer installiert wird. Mit diesem Steuerelement können Sie zum Beispiel Webseiten, animierte *.gif-Dateien oder Excel-Tabellenblätter anzeigen.

Nachfolgend eine Übersicht aller Symbole und Funktionen der STEUERELEMENT-TOOLBOX-Symbolleiste.

Icon	Name	Typ / progID	Beschreibung
	Entwurfsmodus		Wechselt zwischen Formularentwurfsmodus, in dem Steuerelemente erstellt und bearbeitet werden können, und dem Anwendermodus.
	Eigenschaften		Zeigt das Eigenschaftsfenster des markierten Steuerelementes im Tabellenblatt an. Ändern der Eigenschaften der Websteuerelemente.
	Code anzeigen		Zeigt das Codeblatt des aktiven Tabellenblattes im Visual Basic-Editor an.
	Kontrollkästchen	Forms.CheckBox.1	Eine Option, die Sie einschalten oder ausschalten können, indem Sie sie aktivieren oder deaktivieren. In einem Blatt können mehrere Kontrollfelder gleichzeitig aktiviert sein.
	Textfeld	Forms.TextBox.1	Ein Feld, in dem Text eingegeben werden kann.
	Befehlsschaltfläche	Forms.CommandButton.1	Eine Schaltfläche, die eine Aktion initialisiert, wenn darauf geklickt wird.

Tabelle 31: Symbole der Steuerelement-Toolbox

Grundlagen

Allgemein

Datum
Zeit

Diagramme

Steuerelemente

Befehlsleiste

Objekt

Diagramm

Ereignisse

User-Form

Web/Mail

Extern

Gemischtes

Spezial

Icon	Name	Typ / progID	Beschreibung
	Optionsfeld	Forms.OptionButton.1	Eine Schaltfläche, die zum Auswählen einer Option – aus einer Gruppe von Optionen gleichen GroupNamens – verwendet wird.
	Listenfeld	Forms.ListBox.1	Ein Feld, das eine Liste mit Elementen enthält.
	Kombinationsfeld	Forms.ComboBox.1	Ein Textfeld mit einem DropDown-Listenfeld. Sie können in dem Feld entweder eine Auswahl in der Liste treffen oder Ihren eigenen Eintrag eingeben.
	Umschaltfläche	Forms.ToggleButton.1	Eine Schaltfläche, die nach dem Klicken darauf niedergedrückt bleibt und nach einem erneuten Klicken darauf wieder herauspringt.
	Drehfeld	Forms.SpinButton.1	Eine Schaltfläche, die einer Zelle oder einem Textfeld zugeordnet werden kann. Um einen Wert zu erhöhen, klicken Sie auf den Aufwärtspfeil; um einen Wert zu verringern, klicken Sie auf den Abwärtspfeil.
	Bildlaufleiste	Forms.ScrollBar.1	Ein Steuerelement, das durch einen Wertebereich blättert, wenn Sie auf die Bildlaufpfeile klicken oder wenn Sie das Bildlauffeld ziehen. Sie können durch eine Seite mit Werten blättern, indem Sie auf eine Stelle zwischen dem Bildlauffeld und einem Bildlaufpfeil klicken.
	Bezeichnung	Forms.Label.1	Text, der einem Arbeitsblatt oder Formular hinzugefügt wurde und Informationen zu einem Steuerelement oder zu dem Arbeitsblatt bzw. Formular enthält.
	Bild	Forms.Image.1	Ein Steuerelement, das ein Bild in einem Formular einbettet.
	Weitere Steuerelemente		Eine Liste weiterer ActiveX-Steuerelemente.

Tabelle 31: Symbole der Steuerelement-Toolbox (Forts.)

ActiveX-Steuerelemente verfügen über Eigenschaften, die Sie im Eigenschaftsfenster des markierten Steuerelementes anpassen können (siehe Abbildung 115). Beispielsweise können Sie die Hintergrundfarbe des Elementes ändern oder die Darstellung des Mauszeigers festlegen, wenn ein Benutzer auf das Steuerelement zeigt.

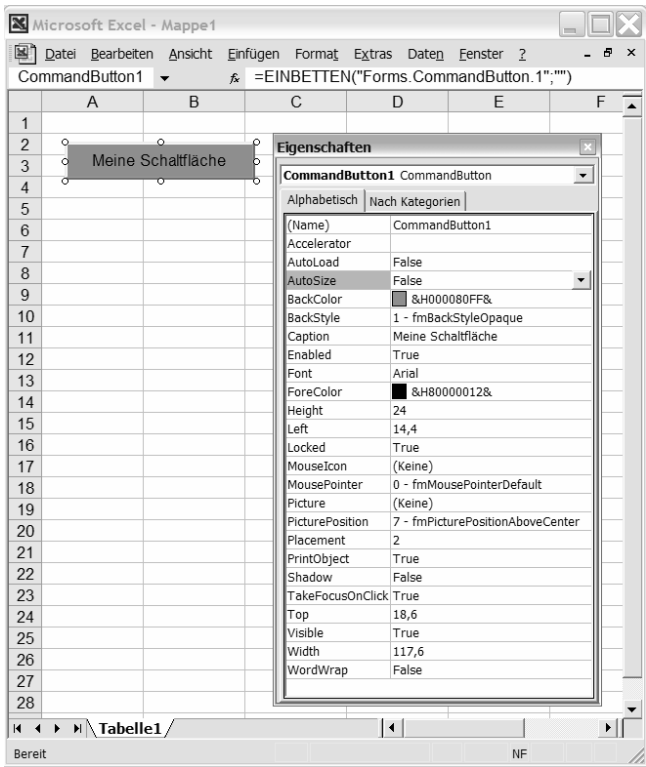


Abbildung 115: Eigenschaftsfenster des aktiven Objektes

Tipp Klicken Sie im Eigenschaftsfenster auf der linken Seite eine Eigenschaft an und drücken Sie dann die Taste **[F1]**. Daraufhin wird die Online-Hilfe zum markierten Thema angezeigt.

Sie können Makros schreiben, die auf Ereignisse reagieren, die mit ActiveX-Steuerelementen verknüpft sind. Im Codeblatt der Tabelle, in welcher die Steuerelemente platziert sind, werden alle eingefügten Steuerelemente im linken DropDown-Feld **OBJEKT** (siehe Abbildung 116) angezeigt.

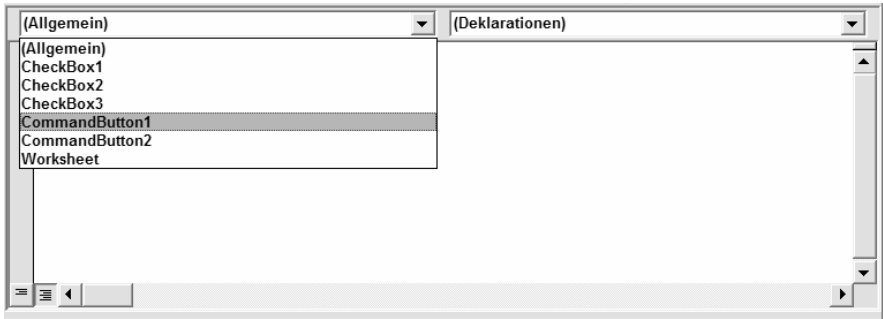


Abbildung 116: Objekte eines Tabellenblattes

Grund-lagen

Allge-mein

Datu Zeit

tungen

St elem te

Befehl leisten

Objekt

Dia-gramm

Ereig-nisse

User-Form

Web/Mail

Extern

Ge-misch tes

Specia

Nachdem Sie ein Objekt aus der Liste ausgewählt haben, stehen im rechten DropDown-Feld PROZEDUREN (siehe Abbildung 117) alle für dieses Objekt verfügbaren Ereignisprozeduren zur Auswahl. Klicken Sie auf einen Eintrag, damit die Codeschablone des Ereignisses mit allen benötigten Parametern im Codeblatt erstellt wird.

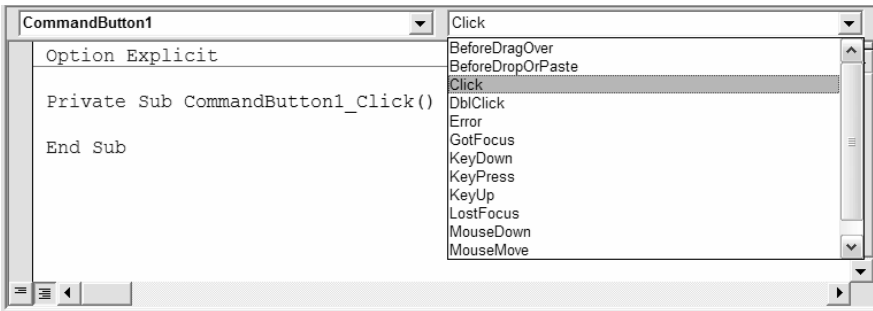


Abbildung 117: Ereignisprozeduren des ausgewählten Objektes

Tipp

Wechseln Sie in der Excel-Umgebung im Tabellenblatt in den Entwurfmodus und führen Sie einen Doppelklick auf ein Steuerelement aus. Daraufhin wird das Codeblatt der Tabelle mit dem Click-Ereignis des Steuerelementes angezeigt. Wenn noch kein Ereignis für dieses Steuerelement vorhanden war, wird die entsprechende Codeschablone erstellt.

Hinweis

Der Unterschied zwischen einem `CommandButton` aus der STEUERELEMENT-TOOLBOX und einer BEFEHLSCHALTFLÄCHE der FORMULAR-Symbolleiste liegt in erster Linie darin, dass Sie dem `CommandButton` Code zuweisen können. Dieser wird bei verschiedenen auftretenden Ereignissen ausgeführt. Befehlsschaltflächen können nur Makros zugewiesen werden, die bei Klick auf die Schaltfläche aufgerufen werden. Zudem können `CommandButtons` aus der STEUERELEMENT-TOOLBOX mit farbiger Hintergrund-Fläche dargestellt werden, was bei den Formular-Befehlsschaltflächen nicht möglich ist.

138 Steuerelemente erstellen

Um auf einem Tabellenblatt ein ActiveX-Steuerelement zu erstellen, müssen Sie lediglich den programmtechnischen Bezeichner des Objekts, welcher über die `ProgId`-Eigenschaft festgelegt ist, an die `Add`-Methode des `OLEObjects`-Auflistungsobjekts übergeben. Folgende einfache Syntax erstellt im aktiven Tabellenblatt an der aktiven Zellposition eine Umschaltfläche. Die Bezeichner `ProgId` der weiteren Steuerelemente können Sie der Tabelle 31 entnehmen.

```
ActiveSheet.OLEObjects.Add "Forms.ToggleButton.1"
```

Viel praktischer ist es, wenn Sie für die programmtechnischen Bezeichner Konstanten am Anfang des Moduls (vor der ersten Prozedur) deklarieren.

```
'=====
' Auf CD      Buchdaten\Beispiele\05_Steuerelemente\
' Dateiname   05_02_ActiveXControls.xls
```

```
' Ereignis    Tabelle1 (AddControls)
'=====

Const CHECKBOX As String = "Forms.CheckBox.1"
Const COMBOBOX As String = "Forms.ComboBox.1"
Const COMMANDBUTTON As String = "Forms.CommandButton.1"
Const IMAGE As String = "Forms.Image.1"
Const LABEL As String = "Forms.Label.1"
Const LISTBOX As String = "Forms.ListBox.1"
Const OPTIONBUTTON As String = "Forms.OptionButton.1"
Const SCROLLBAR As String = "Forms.ScrollBar.1"
Const SPINBUTTON As String = "Forms.SpinButton.1"
Const TEXTBOX As String = "Forms.TextBox.1"
Const TOGGLEBUTTON As String = "Forms.ToggleButton.1"
```

Diese Konstanten können Sie in dem Modul, in dem sie erstellt wurden, in allen Prozeduren in Vertretung der progID verwenden.

```
ActiveSheet.OLEObjects.Add TOGGLEBUTTON
```

Um in allen Modulen des VBA-Projektes auf diese Konstanten zuzugreifen, stellen Sie zusätzlich die Public-Anweisung vor die jeweilige Deklaration.

```
Public Const TOGGLEBUTTON = "Forms.ToggleButton.1"
```

In der folgenden Prozedur wird ein Kontrollkästchen im aktiven Tabellenblatt eingefügt und an die Zellposition der Zelle G3 angepasst. Dies erreichen Sie über die Left-, Top-, Width und Height-Eigenschaften der Zelle G3, welche auf das einzufügende Objekt übertragen wird. Anstelle der progID für Kontrollkästchen Forms.CheckBox.1 verwenden Sie die deklarierte Konstante CHECKBOX.

In dem folgenden With-Block werden verschiedene Eigenschaften der neuen CheckBox wie Name, Beschriftungstext etc. festgelegt.

Wenn Sie versuchen, ein Steuerelement gleichen Namens zu erstellen, gibt Excel eine Fehlermeldung aus. Dies verhindern wir, indem wir über die Count-Eigenschaft die Anzahl der bereits vorhandenen Controls an den Namen des neu eingefügten Steuerelementes anhängen.

```
'=====
' Auf CD      Buchdaten\Beispiele\05_Steuerelemente\
' Dateiname   05_02_ActiveXControls.xls
' Ereignis    Tabelle1 (AddControls)
'=====

Private Sub CommandButton1_Click()
    Const cstrRange As String = "G3"
    Dim obj As OLEObject
    Dim objForm As MSForms.CHECKBOX

    With ActiveSheet
        Set obj = .OLEObjects.Add( _
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

St
elem
te

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

```

        ClassType:=CHECKBOX, _
        Left:=Range(cstrRange).Left, _
        Top:=Range(cstrRange).Top, _
        Width:=Range(cstrRange).Width, _
        Height:=Range(cstrRange).Height)

Set objForm = obj.Object

With objForm
    .Name = "TestCheckBox" & ActiveSheet.OLEObjects.Count
    .Caption = "MyCheckBox"
    .Enabled = True
    .GroupName = "MyGroup"
    .LinkedCell = Range(cstrRange).Offset(0, -1).Address
    .Value = True
End With
End With
End Sub

```

139 Steuerelemente löschen

Um ein bestimmtes Steuerelement Ihrer Tabelle zu löschen, verwenden Sie einfach den Namen des zu löschenden Objektes. Für den Fall, dass das genannte Element nicht existiert, haben wir vor dem Löschvorgang die Fehlerroutine `On Error Resume Next` gesetzt. Es ist ebenso möglich, alle Steuerelemente in einer `For Each...Next`-Anweisung zu durchlaufen und dabei die Namen der Elemente zu überprüfen. Eine Beispielprozedur dazu finden Sie in Rezept 140. Dadurch würde sich unsere vorangestellte Fehlerroutine erübrigen. Der Schaltfläche haben wir den Namen `CmdCheckBox1Delete` zugewiesen. Wenn Sie diese Schaltfläche anklicken, wird das hinterlegte gleichnamige Click-Ereignis aufgerufen.

```

' =====
' Auf CD      Buchdaten\Beispiele\05_Steuerelemente\
' Dateiname   05_02_ActiveXControls.xls
' Ereignis     Tabelle2 (DeleteControls)
' =====

Private Sub CmdCheckBox1Delete_Click()
    On Error Resume Next
    ActiveSheet.OLEObjects("CheckBox1").Delete
End Sub

```

140 Alle Steuerelemente löschen

Mit einer einzigen Zeile ist es möglich, *alle* Grafikobjekte, zu welchen auch die ActiveX-Steuerelemente zählen, zu löschen:

```

ActiveSheet.DrawingObjects.Delete

```

Wenn Sie allerdings diesen Befehl über eine Schaltfläche auf dem Tabellenblatt ausführen, wird die aufrufende Schaltfläche ebenfalls gelöscht. Die nachfolgende Prozedur löscht nur die Objekte, welche nicht vom Typ `CommandButton` sind.

```
'=====
' Auf CD      Buchdaten\Beispiele\05_Steuerelemente\
' Dateiname   05_02_ActiveXControls.xls
' Ereignis     Tabelle2 (DeleteControls)
'=====

Private Sub CmdDeleteAllControls_Click()
    Dim obj As OLEObject

    For Each obj In ActiveSheet.OLEObjects
        If obj.progID <> "Forms.CommandButton.1" Then
            obj.Delete
        End If
    Next obj
End Sub
```

Grundlagen

Allgemein

Datum
Zeit

tungen

St
elem
te

Befehl
leisten

141 Steuerelemente eines Typs löschen

Genauso, wie Sie beim Löschen von Steuerelementen einen Typ ausschließen, können Sie auch nur die Elemente eines bestimmten Typs löschen. Folgende Prozedur löscht im aktiven Tabellenblatt alle Steuerelemente vom Typ `CheckBox`.

```
'=====
' Auf CD      Buchdaten\Beispiele\05_Steuerelemente\
' Dateiname   05_02_ActiveXControls.xls
' Ereignis     Tabelle2 (DeleteControls)
'=====

Private Sub CmdDeleteCheckBoxes_Click()
    Dim obj As OLEObject

    For Each obj In ActiveSheet.OLEObjects
        If obj.progID = "Forms.CheckBox.1" Then
            obj.Delete
        End If
    Next obj
End Sub
```

Objekt

Diagramm

Ereignisse

User-Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

142 Fokus der Steuerelemente verwalten

In der Microsoft Windows-Umgebung kann zu einem Zeitpunkt nur ein Objekt den Fokus besitzen. Zum Beispiel:

- ▶ ein einziges Fenster
- ▶ ein einziges Formular
- ▶ ein einziges Steuerelement
- ▶ eine Zelle/Zellenbereich

Nur das Objekt, welches den Fokus besitzt, kann Mausklicks oder Tastatureingaben empfangen. Das aktive Objekt wird normalerweise durch eine hervorgehobene Überschrift oder Titelleiste angezeigt. Der Fokus kann vom Benutzer oder von der Anwendung gesetzt werden.

Wenn Sie auf ein Steuerelement klicken, erhält dieses standardmäßig den Fokus, bis Sie mit der Maus ein anderes Objekt oder eine Zelle anklicken. Dies erkennen Sie auch daran, dass es nicht möglich ist, sich mit der Tabulator- oder den Pfeiltasten auf dem Tabellenblatt von Zelle zu Zelle zu bewegen, wenn ein Objekt markiert ist. Auch die sonst übliche Adressanzeige im Namensfeld bleibt leer.

Solange ein Steuerelement den Fokus hat, sind einige Menübefehle in Excel deaktiviert und können nicht angewählt werden. Ab Excel 2000 ist es aber möglich, mit VBA-Code dennoch Befehle auszuführen, welche über die Excel-Oberfläche nicht ausgeführt werden können.

Der `CommandButton` der `STEUERELEMENT-TOOLBOX` besitzt als einziges Element die `TakeFocusOnClick`-Eigenschaft. Setzen Sie diese Eigenschaft auf `False`, damit nach dem Ausführen des hinterlegten Codes das vorher aktive Objekt wieder den Fokus erhält. Dies kann entweder eine Zelle oder ein anderes Steuerelement sein.

Um bei den anderen Steuerelementen den Fokus nach Beendigung der aufgerufenen Ereignis-Prozedur an die Tabelle zurückzugeben, selektieren Sie die vorher aktive Zelle bzw. Markierung. Die `Activate`- und `Select`-Methode sind dabei gleichwertig. Am Beispiel eines Kontrollkästchens verwenden Sie folgenden Code:

```
' =====
' Auf CD      Buchdaten\Beispiele\05_Steuerelemente\
' Dateiname   05_02_ActiveXControls.xls
' Ereignis    Tabelle3 (TakeFocusOnClick)
' =====

Private Sub CheckBox2_Click()
    Selection.Activate
End Sub
```

Achtung

Unter Excel 97 erhalten Sie einen Laufzeitfehler, wenn Sie versuchen, auf deaktivierte Befehle zuzugreifen, wenn eine Befehlsschaltfläche den Fokus hat. Setzen Sie die `TakeFocusOnClick`-Eigenschaft des `CommandButtons` auf `False`, damit der Fehler abgefangen und der Befehl ausgeführt wird.

143 Wert in Kontrollkästchen (CheckBox) ändern

Ein Kontrollkästchen `CheckBox` dient dazu, eine angegebene Option als aktiviert bzw. deaktiviert zu kennzeichnen. Bei jedem Klick auf das Kontrollkästchen wird der Status zwischen aktiviert und deaktiviert gewechselt; indem das Kontrollkästchen einen Haken bzw. keinen Haken aufweist. Die `LinkedCell`-Eigenschaft legt die verknüpfte Zelle des Steuerelementes fest. Der aktuelle Zustand der `CheckBox` wird in der angegebenen verknüpften Zelle als boolescher Wert in Form von `Wahr` oder `Falsch` angezeigt. Umgekehrt können Sie in der verknüpften Zelle den Wert ändern, um die Anzeige in der `CheckBox` zu ändern. Geben Sie in der verknüpften Zelle für eine aktivierte `CheckBox` den Wert `Wahr` bzw. die Zahl 1 ein. `False` bzw. 0 bewirkt einen deaktivierten Status des Kontrollkästchens.

Tipp

Den Wert eines Kontrollkästchens können Sie auch über die Leertaste Ihrer Tastatur ändern. Jedes Drücken der Leertaste bewirkt einen Wechsel zwischen `Wahr` und `Falsch`. Dies wird optisch durch einen Haken für ein aktiviertes bzw. kein Haken für ein deaktiviertes Steuerelement angezeigt.

Wenn Sie den Wert der verknüpften Zelle entfernen, wird das Kontrollkästchen in einen neutralen Zustand NULL versetzt und es erscheint abgedunkelt mit hellgrauem Hintergrund (siehe Abbildung 118). Diesen Zustand können Sie auch über das Anklicken des Kontrollkästchens erzielen. Dazu muss die TripleState-Eigenschaft auf True gesetzt werden. Jeder Klick auf die CheckBox wechselt dann zwischen Wahr, Falsch und Null. In der verknüpften Zelle wird der Null-Wert als Fehlermeldung #NV angezeigt.

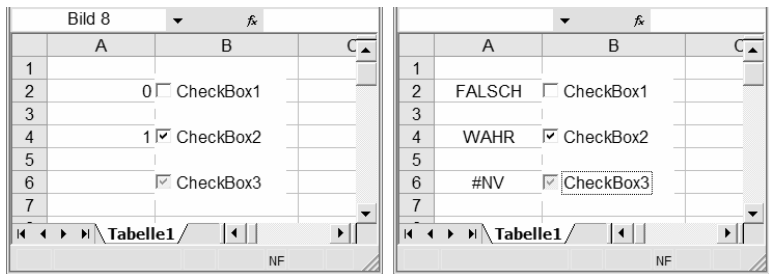


Abbildung 118: Status von Kontrollkästchen ändern

Neben dem Kontrollkästchen weisen die Umschaltfläche und das Optionsfeld ebenfalls die TripleState-Eigenschaft auf. Das Handling der Umschaltfläche ToggleButton ist identisch mit dem der CheckBox.

Bei dem OptionsButton kann ein Null-Zustand nur über VBA oder das Entfernen des Wertes der verknüpften Zelle, aber nicht über das Anklicken der Optionsfelder erzeugt werden. Zu den Optionsfeldern finden Sie in diesem Kapitel einige Rezepte.

Hinweis

Standardmäßig ist die TripleState-Eigenschaft auf False eingestellt. Über VBA bzw. Entfernen des Wertes der verknüpften Zelle können Sie aber dennoch einen NULL-Zustand des Kontrollkästchens erzielen. Um einen NULL-Zustand per Klick auf das Steuerelement zu bewirken, muss allerdings die TripleState-Eigenschaft auf True gesetzt werden.

Nachfolgende Prozeduren ändern den Aktivierungsstatus der CheckBox1.

```
'=====
' Auf CD      Buchdaten\Beispiele\05_Steuerelemente\
' Dateiname  05_02_ActiveXControls.xls
' Ereignis   Tabelle4 (CheckBox)
'=====

Private Sub CmdCheckBoxActivate_Click()
    ActiveSheet.CheckBox1.Value = True
End Sub

Private Sub CmdCheckBoxDeactivate_Click()
    ActiveSheet.CheckBox1.Value = False
End Sub

Private Sub CmdCheckBoxNULL_Click()
    ActiveSheet.CheckBox1.Value = Null
End Sub
```

Grund-lagen

Allge-mein

Datu-Zeit

tungen

St-elem-te

Befehl-leisten

Objekt

Dia-gramm

Ereig-nisse

User-Form

Web/Mail

Extern

Ge-misch-tes

Specia

144 Kontrollkästchen (CheckBox)-Wert auslesen

Die Abfrage nach dem Wert eines Kontrollkästchens kann drei verschiedene Ergebnisse liefern. Zwei der Werte (Wahr und Falsch) sind vom Datentyp `Boolean`. Wenn die `CheckBox` einen Zwischenstand aufweist, also mit sichtbarem Häkchen abgedunkelt dargestellt wird, liefert die `Value`-Eigenschaft den Wert `Null` vom Datentyp `Variant`. `Null` darf nicht mit dem Wert `0` verwechselt werden. Es ist nicht möglich, ein Kontrollkästchen direkt auf einen `Null`-Wert in der Form `If CheckBox1.Value = Null` zu überprüfen. Der Wert `Null` ist nur mit der `IsNull`-Funktion vom Datentyp `Boolean` nachweisbar.

```
If IsNull(CheckBox1.Value) Then
```

Neben unserer nachfolgend vorgestellten Prozedur `CmdReadValue_Click` können Sie lediglich noch über den `Case Else`-Abschnitt der `Select Case`-Anweisung einen `Null`-Zustand genau ermitteln.

```
Select Case ActiveSheet.CheckBox1.Value
    Case True
        MsgBox "Wahr"
    Case False
        MsgBox "Falsch"
    Case Else
        MsgBox "Null"
End Select
```

Klicken Sie in der Beispieldatei auf die Schaltfläche `CHECKBOX-WERT AUSLESEN`, um das hinterlegte `Click`-Ereignis auszulösen.

```
' =====
' Auf CD      Buchdaten\Beispiele\05_Steuerelemente\
' Dateiname   05_02_ActiveXControls.xls
' Ereignis     Tabelle4 (CheckBox)
' =====
```

```
Private Sub CmdReadValue_Click()
    Dim varValue As Variant

    varValue = ActiveSheet.CheckBox1.Value

    If varValue = True Then
        MsgBox "Kontrollkästchen ist aktiviert"
    ElseIf varValue = False Then
        MsgBox "Kontrollkästchen ist deaktiviert"
    ElseIf IsNull(varValue) Then
        MsgBox "Kontrollkästchen ist im Null-Zustand"
    End If
```

```
End Sub
```

145 Wahrheitswert eines Kontrollkästchens übertragen

Wenn Sie sicher sind, dass ein Kontrollkästchen nur den Wert `True` oder `False` für aktiviert/deaktiviert haben kann, reicht es, wenn Sie den booleschen Wert des Kontrollkästchens nutzen, um Einstellungen, welche ebenfalls einen booleschen Wert erwarten, zu ändern. Um zum Beispiel die Gitternetzlinien des Tabellenblattes ein- oder auszublenden, verwenden Sie das `Change`-Ereignis einer `CheckBox`. Jedes Mal, wenn der Anwender den Status des Kontrollkästchens durch Anklicken ändert, wird die Anzeige der Gitternetzlinien über die folgende Ereignisprozedur gewechselt.

Achtung

Wenn das Kontrollkästchen einen Zwischenzustand `Null` aufweist, liefert folgende Prozedur einen Fehler. Gitternetzlinien können nur ein- oder ausgeblendet werden, weisen aber keinen Zustand `Null` auf. Wenn Sie die `TriState`-Eigenschaft auf `False` gesetzt haben und keine verknüpfte Zelle festlegen oder diese für den Anwender nicht zugänglich machen, kann aber kein Fehler passieren.

```
'=====
' Auf CD      Buchdaten\Beispiele\05_Steuerelemente\
' Dateiname   05_02_ActiveXControls.xls
' Ereignis     Tabelle4 (CheckBox)
'=====

Private Sub CheckBox2_Change()
    ActiveWindow.DisplayGridlines = CheckBox2.Value
End Sub
```

146 Steuerelement-Eigenschaften ändern

Die Eigenschaften eines Steuerelementes ändern Sie, indem Sie in der VBA-Codezeile nach dem Objekt eine Eigenschaft, gefolgt von einem Wert angeben. Verwenden Sie folgende Prozedur, um den Beschriftungstext `Caption` eines Kontrollkästchens zu ändern. Sie können auf diese Art eine beliebige Eigenschaft oder Methode des genannten Objektes angeben und einstellen.

```
'=====
' Auf CD      Buchdaten\Beispiele\05_Steuerelemente\
' Dateiname   05_02_ActiveXControls.xls
' Ereignis     Tabelle4 (CheckBox)
'=====

Private Sub CmdCheckBoxCaption_Click()
    ActiveSheet.CheckBox1.Caption = "My CheckBox"
End Sub
```

Wenn Sie nicht alle Eigenschaften, welche für ein Steuerelement zur Verfügung stehen, auswendig im Kopf haben, lassen Sie sich alle verwendbaren Eigenschaften in der `INTELLISENSE`-Liste anzeigen.

Damit Excel erkennen kann, um welches Objekt es sich handelt, bedarf es allerdings vorab einer genauen Zuweisung des Objektes. Da es sich bei dem zu bearbeitenden Steuerelement um eine `CheckBox` der `STEUERELEMENT-TOOLBOX` handelt, deklarieren Sie die Variable `objForm` vom Datentyp `MSForms.CHECKBOX`. Innerhalb einer `With`-Anweisung können alle Eigenschaften und Methoden des angegebenen Objektes `objForm` untereinander angegeben werden. Geben Sie innerhalb des `With`-

Grundlagen
Allgemein
Datum und Zeit
Formeln
Diagramme
Steuerelemente
Befehle
Objekte
Diagramme
Ereignisse
User-Formen
Web/Mail
Extern
Gemischtes
Spezial

Blocks einen Punkt (.) ein, damit automatisch die INTELLISENSE mit allen für dieses Objekt verfügbaren Eigenschaften angezeigt wird. Ohne Verwendung der With-Anweisung wird die Intellisense-Liste im Anschluss an den Variablennamen, gefolgt von einem Punkt (.) angezeigt (objForm.).

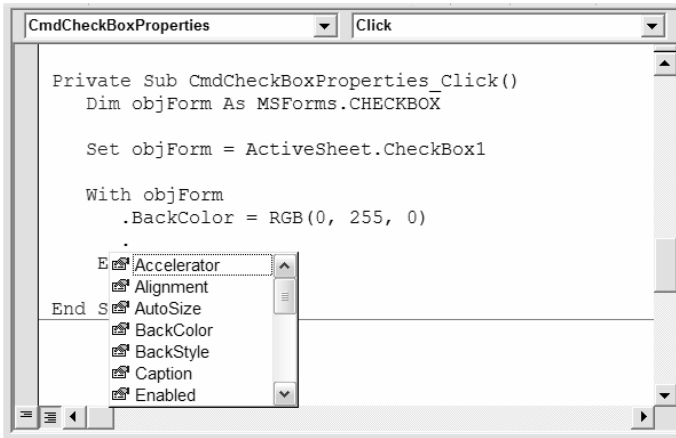


Abbildung 119: CheckBox-Eigenschaften in Intellisense-Liste anzeigen

Folgende Prozedur ändert im aktiven Tabellenblatt die Hintergrundfarbe der CheckBox1.

```
'=====
' Auf CD      Buchdaten\Beispiele\05_Steuerelemente\
' Dateiname   05_02_ActiveXControls.xls
' Ereignis    Tabelle4 (CheckBox)
'=====

Private Sub CmdCheckBoxProperties_Click()
    Dim objForm As MSForms.CHECKBOX

    Set objForm = ActiveSheet.CheckBox1

    With objForm
        .BackColor = RGB(0, 255, 0)
        ' .      weitere Eigenschaft
    End With
End Sub
```

147 Mit Umschaltfläche Steuerelement steuern

Die Umschaltfläche `ToggleButton` verhält sich ähnlich wie eine `CheckBox`. Anstelle eines Häkchens wird diese Schaltfläche als gedrückt bzw. nicht gedrückt dargestellt. Wenn die Umschaltfläche gedrückt dargestellt wird, liefert deren `Visible`-Eigenschaft den Wert `True`, nicht gedrückt gibt `False` zurück. Wenn die `TriState`-Eigenschaft auf `True` eingestellt ist, ergibt der Zwischenstand der Umschaltfläche den Wert `Null`.

Um den Aktivierungswechsel beim Anklicken auf die Umschaltfläche abzufangen, können Sie das `Change`-Ereignis des `ToggleButtons` verwenden. Nachfolgende Prozedur ermittelt über eine `If...Then...Else`-Anweisung den booleschen Wert der Umschaltfläche und blendet ein Kontroll-

kästchen entsprechend diesem Wert ein oder aus. Die `ToggleButton`-Beschriftung `Caption` wird ebenfalls in Abhängigkeit der `Value`-Eigenschaft geändert.

Hinweis Die `If...Then...Else`-Anweisung fragt nur nach dem Wert `True`. Das heißt, nur wenn die Umschaltfläche gedrückt ist, wird die `CheckBox` eingeblendet. Es wird kein Zwischenstand berücksichtigt. Wenn die `TriState`-Eigenschaft auf `True` eingestellt ist, bewirkt ein Wert `Null` wie beim Wert `False` das Ausblenden des angegebenen Steuerelementes.

```
'=====
' Auf CD      Buchdaten\Beispiele\05_Steuerelemente\
' Dateiname  05_02_ActiveXControls.xls
' Ereignis    Tabelle4 (CheckBox)
'=====

Private Sub TglCheckBoxVisible_Click()
    If TglCheckBoxVisible.Value = True Then
        ActiveSheet.CheckBox1.Visible = True
        TglCheckBoxVisible.Caption = "CheckBox ausblenden"
    Else
        ActiveSheet.CheckBox1.Visible = False
        TglCheckBoxVisible.Caption = "CheckBox einblenden"
    End If
End Sub
```

Ein Klick auf eine Umschaltfläche löst zuerst das `Change`-Ereignis des Steuerelementes aus. Anschließend wird aber auch ein hinterlegtes `Click`-Ereignis aufgerufen. Es ist also mit beiden Ereignissen möglich, auf einen Wechsel des Aktivierungsstatus zu reagieren.

Hinweis Wenn die `TriState`-Eigenschaft der Umschaltfläche auf `True` eingestellt ist, wird beim Anklicken des `ToggleButtons` nur das `Change`-Ereignis, jedoch nicht das `Click`-Ereignis ausgelöst.

In folgender Prozedur wird der boolesche Wert der `Enabled`-Eigenschaft, also das mögliche Anwählen der `CheckBox1`, über den booleschen Wert der Umschaltfläche gesteuert. Der Beschriftungstext wird über die `IIf`-Funktion dirigiert, welche sich ebenfalls nach der `Value`-Eigenschaft der Umschaltfläche richtet.

```
'=====
' Auf CD      Buchdaten\Beispiele\05_Steuerelemente\
' Dateiname  05_02_ActiveXControls.xls
' Ereignis    Tabelle4 (CheckBox)
'=====

Private Sub TglCheckBoxEnabled_Click()
    ActiveSheet.CheckBox1.Enabled = TglCheckBoxEnabled.Value

    With TglCheckBoxEnabled
        .Caption = "CheckBox " & _
```

Grundlagen

Allgemein

Datum
Zeit

Formen

Steuerelemente

Bedienung

Objekt

Diagramm

Ereignisse

User-Form

Web/Mail

Extern

Gemischtes

Spezial

```

        IIf(.Value, "deaktivieren", "aktivieren")
    End With
End Sub

```

148 Gruppen von Steuerelementen erstellen

Das aus der FORMULAR-Symbolleiste bekannte GRUPPENFELD ist in der STEUERELEMENT-TOOLBOX nicht verfügbar. Um eine Anzahl an ActiveX-Steuerelementen optisch zusammenzufassen, verwenden Sie das Gruppenfeld der FORMULAR-Symbolleiste. Oft ist es erwünscht, Kontrollkästchen oder Optionsfelder lediglich als *optische* Gruppe darzustellen. Ziehen Sie dazu einfach um mehrere Steuerelemente einen Rahmen der Formular-Symbolleiste.

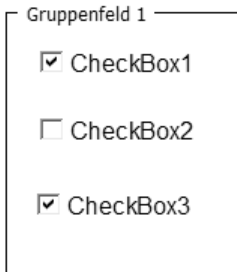


Abbildung 120: Optische Gruppe von ActiveX-Steuerelementen mit Rahmen aus Formular-Symbolleiste umgeben

Hinweis

Beachten Sie, dass ActiveX-Steuerelemente in einem Gruppenfeld der FORMULAR-Symbolleiste nicht als eigenständige funktionelle Gruppe erkannt werden. Gruppen von Steuerfeldern müssen über die `GROUPNAME`-Eigenschaft der Steuerelemente entsprechend definiert werden.

Als *funktionelle* Gruppe bezeichnet man Steuerelemente, die inhaltlich und logisch miteinander verbunden sind. Steuerelemente, die logisch miteinander verbunden sind, beeinflussen sich gegenseitig. Wenn Sie auf einem Tabellenblatt mehrere Optionsfelder aufziehen und der Wert eines Optionsfeldes auf `True` gesetzt wird, werden die Werte aller anderen Optionsfelder auf `False` gesetzt.

Oftmals sollen einzelne Gruppen von Optionsfeldern auf einem Tabellenblatt sich aber nicht gegenseitig beeinflussen und nur innerhalb der Gruppe logisch miteinander verbunden sein. Um eine Gruppe von Optionsfeldern von anderen Steuerelementen getrennt zu verwalten, müssen Sie allen Elementen der Gruppe über die `GroupName`-Eigenschaft den gleichen Gruppennamen zuweisen, damit diese als eigenständige Gruppe erkannt werden.

Tipp

Um mehrere Steuerelemente als eine Gruppe zu definieren, können Sie die Elemente mit gedrückter `[Strg]`-Taste nacheinander markieren, und im Eigenschaftsfenster in der Zeile `GroupName` einen Namen, zum Beispiel `GRUPPE1`, für diese Gruppe vergeben (siehe Abbildung 121).

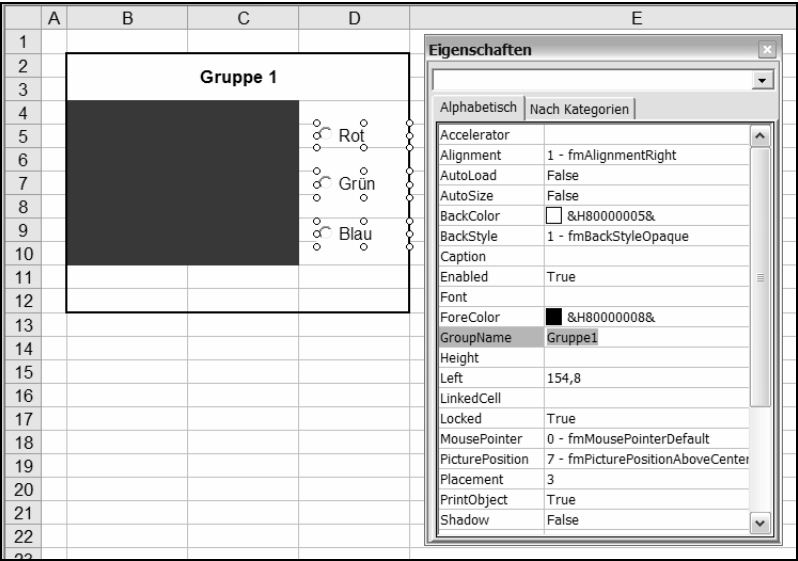


Abbildung 121: Optionsfelder als funktionelle Gruppe definieren

Im Tabellenblatt `OPTIONBUTTON` der Beispieldatei sollen drei Bereiche aus mehreren Zellen unabhängig voneinander über Optionsfelder eingefärbt werden. Die einzufärbenden Bereiche werden im Codeblatt der Tabelle vor der ersten Prozedur über deklarierte Konstanten definiert. Dadurch können die Zellenbereiche in den verwendeten Prozeduren direkt über die Variablen angesprochen werden. Änderungen an der Tabellenstruktur müssen dann nur noch in der Konstanten-Deklaration und nicht in den einzelnen Prozeduren geändert werden. Die Konstanten wurden ohne vorangestellte `Public`-Anweisung deklariert. Dadurch stehen sie nur in dem Modul, in dem sie deklariert wurden, zur Verfügung.

Wenn Sie einen `OptionButton` aus einer der drei Gruppen anklicken, wird das hinterlegte `Click`-Ereignis aufgerufen und der angegebene Zellenbereich entsprechend dem genannten Farbindex eingefärbt.

```
'=====
' Auf CD      Buchdaten\Beispiele\05_Steuerelemente\
' Dateiname  05_02_ActiveXControls.xls
' Ereignis   Tabelle5 (OptionButton)
'=====

Const strColorRange1 As String = "B4:C10"
Const strColorRange2 As String = "F4:G10"
Const strColorRange3 As String = "J4:K10"

' Gruppe 1
Private Sub OptBtnRed_Click()
    Range(strColorRange1).Interior.ColorIndex = 3
End Sub

Private Sub OptBtnGreen_Click()
    Range(strColorRange1).Interior.ColorIndex = 10
```

Grund-lagen
Allge-mein
Datu-Zeit
tungen
St-elem-te
Befehl-leisten
Objekt
Dia-gramm
Ereig-nisse
User-Form
Web/Mail
Extern
Ge-misch-tes
Specia

```

End Sub

Private Sub OptBtnBlue_Click()
    Range(strColorRange1).Interior.ColorIndex = 5
End Sub

' Gruppe 2
Private Sub OptBtnYellow_Click()
    Range(strColorRange2).Interior.ColorIndex = 6
End Sub

Private Sub OptBtnGrey_Click()
    Range(strColorRange2).Interior.ColorIndex = 15
End Sub

Private Sub OptBtnOrange_Click()
    Range(strColorRange2).Interior.ColorIndex = 45
End Sub

' Gruppe 3
Private Sub OptBtnTurkey_Click()
    Range(strColorRange3).Interior.ColorIndex = 8
End Sub

Private Sub OptBtnOliv_Click()
    Range(strColorRange3).Interior.ColorIndex = 12
End Sub

Private Sub OptBtnPink_Click()
    Range(strColorRange3).Interior.ColorIndex = 7
End Sub

```

149 Optionsfelder auswerten

Innerhalb einer Gruppe von Optionsfeldern kann immer nur ein Element aktiviert sein. In der VBA-Programmierung gibt die `Value`-Eigenschaft den Wert `True` für das aktive Element einer Gruppe zurück. Alle anderen Elemente der Gruppe haben entweder den Wert `False`, für nicht aktiviert, oder `Null` für einen Zwischenstand. Um zu ermitteln, welches Element der Gruppe aktiv ist, müssen alle `OptionButton` dieser Gruppe überprüft werden. Sobald eines dieser Elemente den Wert `True` aufweist, wird die Beschriftung über die `Caption`-Eigenschaft als Ergebnis der Auswertung ausgelesen.

Zur Auswertung einer Optionsfelder- oder Kontrollkästchen-Gruppe stellen wir Ihnen in diesem Beispiel die benutzerdefinierte Funktion `ResultControls` mit folgenden zwei Parametern vor:

```
ResultControls(strWks, strGroup)
```

Diese Funktion ist in einem Standardmodul hinterlegt und erwartet als erstes Argument den Namen des Tabellenblattes, in dem sich die zu überprüfende Gruppe befindet. Als zweites Argument muss der Gruppenname übergeben werden. Über das Schlüsselwort `Me` wird das Tabellenblatt angesprochen, in dessen Codemodul sich die aufgerufene Ereignis-Prozedur befindet.

Der Aufruf der Funktion erfolgt über eine Befehlsschaltfläche.

```
'=====
' Auf CD      Buchdaten\Beispiele\05_Steuerelemente\
' Dateiname  05_02_ActiveXControls.xls
' Ereignis    Tabelle5 (OptionButton)
'=====

Private Sub CmdBtnResultGr4_Click()
    MsgBox ResultControls(Me.Name, "Gruppe4")
End Sub
```

Um eine Gruppe von Optionsfeldern zu überprüfen, müssen in einer For Each...Next-Anweisung alle Steuerelemente des angegebenen Tabellenblattes durchlaufen werden. Über die progID-Eigenschaft wird der programmtechnische Bezeichner zurückgegeben. Wenn das überprüfte Element der Schleife vom Bezeichner Forms.OptionButton.1 ist, den angegebenen Gruppennamen strGroup und den Wert True aufweist, wird der Beschriftungstext Caption als Ergebnis der Funktion festgelegt und die Funktion verlassen.

```
'=====
' Auf CD      Buchdaten\Beispiele\05_Steuerelemente\
' Dateiname  05_02_ActiveXControls.xls
' Modul      mdl_01_FunctionResultControls
'=====

Function ResultControls(strWks As String, _
                        strGroup As String) As String

Dim objCtrl As OLEObject

For Each objCtrl In Sheets(strWks).OLEObjects
    If objCtrl.progID = "Forms.OptionButton.1" Then
        If objCtrl.Object.GroupName = strGroup Then
            ' Wenn Elementwert auf True, Beschriftungstext als
            ' Funktionsergebnis festlegen
            If objCtrl.Object.Value Then
                ResultControls = objCtrl.Object.Caption
                Exit Function
            End If
        End If
    End If
Next objCtrl
End Function
```

150 Optionsfelder und Kontrollkästchen zurücksetzen

Um Kontrollkästchen zurückzusetzen, klicken Sie diese im Tabellenblatt an, so dass der Haken entfernt ist. Bei Optionsfeldern ist diese *manuelle* Verfahrensweise leider nicht mit der Maustaste möglich, da über die Excel-Anwendung immer ein Optionsfeld einer Gruppe von Optionsfeldern den Wert Wahr aufweist.

Grundlagen
Allgemein
Datum und Zeit
Formeln
Diagramme
Steuerelemente
Befehlsleisten
Objekte
Diagramme
Ereignisse
User-Formen
Web/Mail
Extern
Gemischtes
Spezial

Sie können im Tabellenblatt lediglich die Inhalte der verknüpften Zellen von Optionsfeldern ändern, um diese zurückzusetzen. Um Optionsfeldern eine verknüpfte Zelle zuzuweisen, verwenden Sie deren `LinkedCell`-Eigenschaft. In der angegebenen Zelle wird der Wert des verknüpften Elementes angezeigt. Sie können die `LinkedCell`-Eigenschaft entweder im Eigenschaften-Fenster des Steuerelementes oder über VBA-Code festlegen. Schreiben Sie in die verknüpften Zellen den Wert `Falsch` oder `0`, um das verknüpfte Optionsfeld zurückzusetzen, also zu deaktivieren.

Im Eigenschaften-Fenster selbst können Sie den Wert eines Optionsfeldes ändern, indem Sie der `Value`-Eigenschaft den Wert `True` oder `False` zuweisen. Geben Sie keine `Value`-Eigenschaft an, oder löschen Sie einen vorhandenen Wert, wird das Optionsfeld in den `Null`-Zustand versetzt.

In diesem Beispiel stellen wir Ihnen die Prozedur `ResetControls` vor. Sie ist in der Lage, sowohl Kontrollkästchen als auch Optionsfelder einer Gruppe zurückzusetzen. Ebenso ist es möglich, allen Elementen der angegebenen Gruppe den Zustand `Null` zuzuweisen. Alle Kontrollkästchen einer Gruppe können zudem auf `True` gesetzt werden.

Der Aufruf der Prozedur `ResetControls` erfolgt über eine Befehlsschaltfläche, wobei die erforderlichen Argumente beim Aufruf übergeben werden.

```
ResetControls(strWks, strForm, strGroup, [strValue = "False"])
```

Parameter	Bedeutung
strWks As String	Name des Tabellenblattes, auf dem sich die Elemente befinden.
strForm As String	Objektangabe: ► <code>OptionButton</code> ► <code>CheckBox</code>
strGroup As String	Gruppenname
[strValue As String = "False"])	Optionaler Wert, kann <code>True</code> , <code>False</code> oder <code>Null</code> sein. Standardwert ist <code>False</code> . Eine Gruppe von Optionsfeldern kann nicht auf <code>True</code> gesetzt werden.

Tabelle 32: Parameter der `ResetControls`-Prozedur

```
' =====  
' Auf CD      Buchdaten\Beispiele\05_Steuerelemente\  
' Dateiname   05_02_ActiveXControls.xls  
' Tabelle     OptionButton  
' Modul       mdl_02_ResetControls  
' =====  
  
Sub ResetControls(strWks As String, _  
                  strForm As String, _  
                  strGroup As String, _  
                  Optional strValue As String = "False")  
  
    Dim objCtrl As OLEObject
```


<pre>' Umwandeln der Objektangabe in progID strForm = IIf(strForm = "OptionButton", _ "Forms.OptionButton.1", _ "Forms.CheckBox.1") ' alle Optionsfelder einer Gruppe können nicht den ' Wert True erhalten If strForm = "Forms.OptionButton.1" And _ strValue = "True" Then Exit Sub For Each objCtrl In Sheets(strWks).OLEObjects If objCtrl.progID = strForm Then If objCtrl.Object.GroupName = strGroup Then objCtrl.Object.Value = strValue End If End If Next objCtrl End Sub</pre>	Grund-lagen Allge-mein Datu-Zeit tungen St- elem- te Befehl-leisten Objekt

Wenn Sie im Tabellenblatt **OPTIONBUTTON** auf die Schaltfläche **ZURÜCKSETZEN** der **GRUPPE4** klicken, wird das hinterlegte Click-Ereignis ausgelöst. Über die aufgerufene Prozedur `ResetControls` werden alle Optionsfelder der **GRUPPE4** zurückgesetzt. Dem optionalen Parameter `strValue` wird der Wert `False` zugewiesen. Dies ist der Standardwert von `strValue` und muss nicht unbedingt angegeben werden.

<pre>'===== ' Auf CD Buchdaten\Beispiele\05_Steuerelemente\ ' Dateiname 05_02_ActiveXControls.xls ' Ereignis Tabelle5 (OptionButton) '===== Private Sub CmdBtnResetGr4_Click() ' ResetControls(strWks, strForm, strGroup, [strValue = "False"]) ResetControls Me.Name, "OptionButton", "Gruppe4", "False" End Sub</pre>	User-Form Web/Mail Extern Ge-misch-tes

In **GRUPPE5** werden wiederum alle `OptionButton` dieser Gruppe zurückgesetzt. Der folgende Aufruf verdeutlicht, dass bei fehlender Angabe für den `strValue`-Parameter diesem in der Prozedur automatisch der Wert `False` zugewiesen wird.

<pre>'===== ' Auf CD Buchdaten\Beispiele\05_Steuerelemente\ ' Dateiname 05_02_ActiveXControls.xls ' Ereignis Tabelle5 (OptionButton) '===== Private Sub CmdBtnResetGr5_Click() ResetControls Me.Name, _</pre>	Specia

```
        "OptionButton", _  
        "Gruppe5"  
  
End Sub
```

Hinweis Obwohl Optionsfelder die `TriState`-Eigenschaft aufweisen, ist es manuell über die Excel-Anwendung nur über die verknüpfte Zelle möglich, einen `Null`-Zustand zu erreichen. Wenn Sie über VBA einen Zwischenstand zuweisen, ist es unerheblich, ob die Eigenschaft auf `False` oder `True` gesetzt wird. In beiden Fällen wird der Befehl ausgeführt und die Elemente zurück bzw. auf `Null` gesetzt.

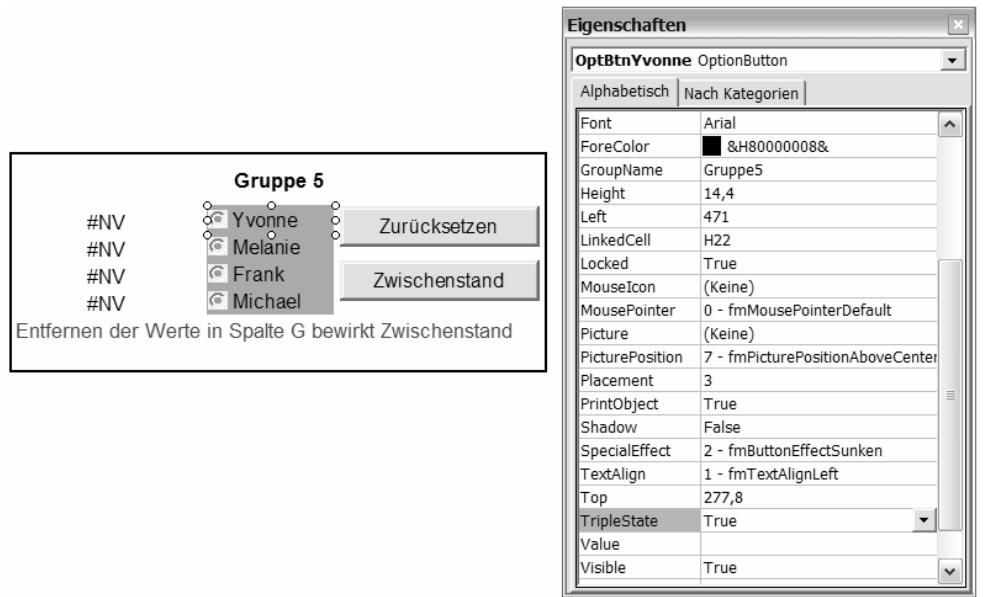


Abbildung 122: Alle Optionsfelder einer Gruppe auf Null setzen

Die folgende Prozedur weist allen Optionsfeldern der GRUPPE5 des aktiven Tabellenblattes einen `Null`-Wert zu (siehe Abbildung 122).

```
' =====  
' Auf CD      Buchdaten\Beispiele\05_Steuerelemente\  
' Dateiname   05_02_ActiveXControls.xls  
' Ereignis    Tabelle5 (OptionButton)  
' =====  
  
Private Sub CmdBtnNULLGr5_Click()  
    ResetControls Me.Name, _  
        "OptionButton", _  
        "Gruppe5", _  
        "NULL"  
  
End Sub
```

151 Arbeitsblätter über Listenfeld auswählen

Für dieses Beispiel haben wir im Tabellenblatt ein Listenfeld aus der STEUERELEMENT-TOOLBOX aufgezogen (siehe Abbildung 123). Die Größe des Listenfeldes `ListBox` haben wir so bemessen, dass immer drei Einträge (Namen von vorhandenen Arbeitsblättern der Excel-Arbeitsmappe) sichtbar sind. Weitere Tabellen- und Diagrammblätter der Arbeitsmappe werden durch Klicken auf die Pfeile der `ListBox` angezeigt.

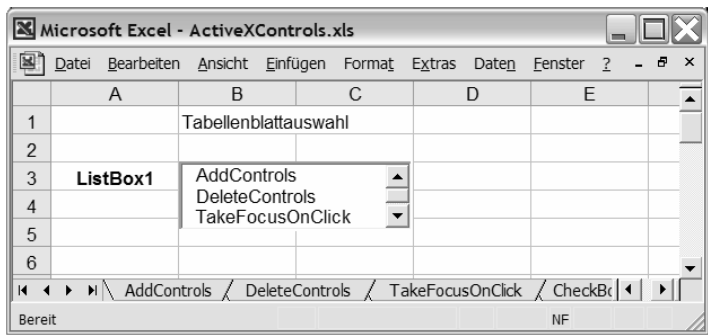


Abbildung 123: Tabellenblattauswahl über Listenfeld

Beim Öffnen der Datei wird das Listenfeld `ListBox1` im `Workbook_Open`-Ereignis gefüllt.

```
'=====
' Auf CD      Buchdaten\Beispiele\05_Steuerelemente\
' Dateiname   05_02_ActiveXControls.xls
' Ereignis    DieseArbeitsmappe
'=====

Private Sub Workbook_Open()
    Dim sh As Object

    For Each sh In ActiveWorkbook.Sheets
        Sheets("ListBox").ListBox1.AddItem sh.Name
    Next sh
End Sub
```

Immer wenn das Listenfeld ausgewählt wird, also den Fokus erhält, wird das `GotFocus`-Ereignis der `ListBox1` ausgelöst. Dieses Ereignis nutzen wir, um zunächst über die `Clear`-Methode bereits vorhandene Einträge aus vorherigen Auflistungen zu entfernen. In der anschließenden `For Each...Next`-Anweisung werden dem Listenfeld über die `AddItem`-Methode alle Blätter der aktiven Datei hinzugefügt. Durch die Verwendung des `Sheets`-Auflistungsobjektes werden sowohl Tabellen- als auch Diagrammblätter der aktiven Arbeitsmappe aufgeführt. Dadurch wird sichergestellt, dass immer alle aktuellen Arbeitsblätter in dem Listenfeld angezeigt werden.

```
'=====
' Auf CD      Buchdaten\Beispiele\05_Steuerelemente\
' Dateiname   05_02_ActiveXControls.xls
' Ereignis    Tabelle6 (ListBox)
```

Grundlagen
Allgemein
Datum und Zeit
Formeln
Diagramme
Steuerelemente
Befehlsleisten
Objekt
Diagramm
Ereignisse
User-Form
Web/Mail
Extern
Gemischtes
Spezial

```
' =====
Private Sub ListBox1_GotFocus()
    Dim sh As Object

    ListBox1.Clear

    For Each sh In ActiveWorkbook.Sheets
        ListBox1.AddItem sh.Name
    Next sh
End Sub
```

Sobald Sie einen Eintrag im Listenfeld anklicken, wird über das hinterlegte `Click`-Ereignis das ausgewählte Arbeitsblatt angezeigt.

```
' =====
' Auf CD      Buchdaten\Beispiele\05_Steuerelemente\
' Dateiname   05_02_ActiveXControls.xls
' Ereignis     Tabelle6 (ListBox)
' =====

Private Sub ListBox1_Click()
    Sheets(ListBox1.Text).Activate
End Sub
```

152 Listenfeld durch Mausbewegung füllen

Wenn Sie einem Listen- oder Kombinationsfeld Einträge über eine Schleife zuordnen, ist das Steuerelement beim Öffnen der Excel-Datei zunächst leer. Das Befüllen können Sie, wie im vorherigen Beispiel, im `Workbook_Open`-Ereignis ausführen lassen. Eine weitere Möglichkeit bietet das `MouseMove`-Ereignis. Dieses wird ausgelöst, sobald der Maus-Cursor das Steuerelement »berührt«.

In diesem Beispiel wird das Steuerelement `ListBox2` mit den sieben Wochentagen gefüllt (siehe Abbildung 124). Dazu verwenden wir die `WeekdayName`-Funktion. Bei jedem Durchlauf der `For Each...Next`-Anweisung wird durch den Schleifenzähler `intI` der Wert des Wochentages hochgezählt und über die `AddItem`-Methode der `ListBox2` hinzugefügt. Die `ListStyle`-Eigenschaft ist auf `fmListStyleOption` eingestellt. Dadurch werden die Listeneinträge als Optionsfelder dargestellt.

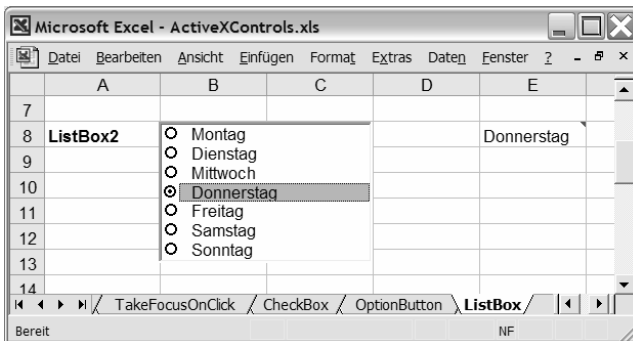


Abbildung 124: Listenfeld als Gruppe von Optionsfeldern anzeigen

Bei mehrmaligem Kontakt mit dem Steuerelement würden die Wochentage durch das ausgelöste `MouseMove`-Ereignis wiederholt in die `ListBox2` geschrieben. Um das zu verhindern, können Sie vor dem Befüllen der `ListBox2` alle vorhandenen Einträge mit der `Clear`-Methode löschen. Da sich aber die Werte der `ListBox2` nicht ändern, wird mittels der `ListCount`-Eigenschaft überprüft, ob das Listenfeld bereits gefüllt ist und die Prozedur in diesem Fall vorzeitig verlassen. Der durch Mausklick ausgewählte Wert wird in die angegebene Zelle der `LinkedCell`-Eigenschaft geschrieben.

```
'=====
' Auf CD      Buchdaten\Beispiele\05_Steuerelemente\
' Dateiname  05_02_ActiveXControls.xls
' Ereignis    Tabelle6 (ListBox)
'=====

Private Sub ListBox2_MouseMove(ByVal Button As Integer, _
                               ByVal Shift As Integer, _
                               ByVal X As Single, ByVal Y As Single)

    Dim intI As Integer

    If ListBox2.ListCount > 0 Then Exit Sub

    For intI = 1 To 7
        ListBox2.AddItem WeekdayName(intI)
    Next intI
End Sub
```

153 Monate in Listenfeld auflisten

Ein Listenfeld lässt sich nicht wie ein Kombinationsfeld als `DropDown`-Feld anzeigen. Wenn Sie ein Listenfeld auf die Höhe eines Eintrages beschränken, ist immer nur ein Eintrag sichtbar. Durch Klick auf die Pfeile am Steuerelement wird der vorherige bzw. nachfolgende Wert der Liste angezeigt (siehe Abbildung 125). Ein Klick auf die Auswahl schreibt den Wert in die verknüpfte Zelle `LinkedCell`.

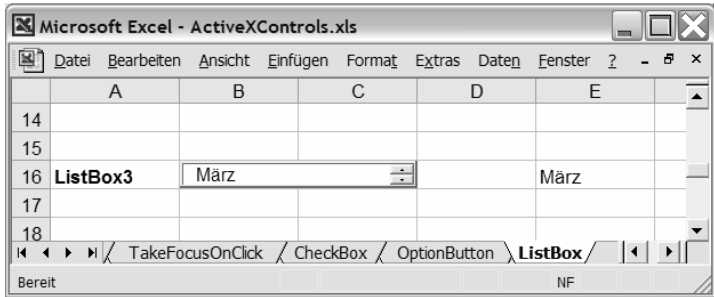


Abbildung 125: Listenfeld mit Monatsauswahl

Die Darstellungsart des Listenfeldes können Sie über die `SpecialEffect`-Eigenschaft festlegen. In unserem Beispiel haben wir die Konstante `fmSpecialEffectBump` ausgewählt. Dadurch ist das Objekt am unteren und rechten Rand hervorgehoben und erscheint am oberen und linken Rand flach. Diese Einstellung ist für Kontrollkästchen und Optionsfelder nicht gültig.

Grund-lagen

Allge-mein

Datu-Zeit

tungen

St-ele-men-te

Befehl-leisten

Objekt

Dia-gramm

Ereig-nisse

User-Form

Web/Mail

Extern

Ge-misch-tes

Specia

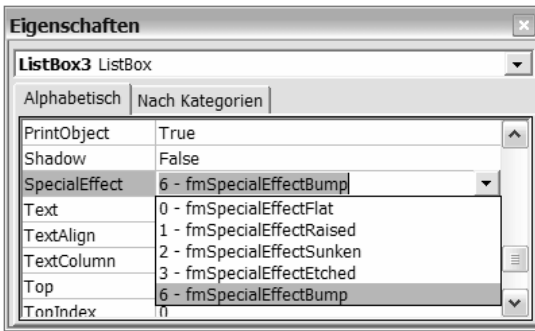


Abbildung 126: Darstellung eines Objekts festlegen

Experimentieren Sie ein wenig mit den verschiedenen Einstellungen der `SpecialEffect`-Eigenschaft um die von Ihnen bevorzugte Darstellung der Steuerelemente zu finden.

Die `ListBox3` wird wieder im `Workbook_Open`-Ereignis der Beispieldatei gefüllt. Dabei wird die `MonthName`-Funktion angewandt, welche eine Zeichenfolge zurückgibt, die den festgelegten Monat angibt. Die Syntax lautet: `MonthName(Monat[, abkürzen])`.

Tip

Um die Monatsnamen in abgekürzter Schreibweise darzustellen, geben Sie in der `MonthName`-Funktion für den zweiten Parameter `abkürzen` den Wert `True` an:

`ListBox3.AddItem MonthName(intI, True)` ' liefert bei 3 den Wert `Mrz`

```
'=====
' Auf CD      Buchdaten\Beispiele\05_Steuerelemente\
' Dateiname  05_02_ActiveXControls.xls
' Ereignis   DieseArbeitsmappe
'=====
```

```
Private Sub Workbook_Open()
    Dim intI As Integer

    For intI = 1 To 12
        Sheets("ListBox").ListBox3.AddItem MonthName(intI)
    Next intI

End Sub
```

154 Listenfeldeinträge bei Auswahl ändern

In diesem Beispiel fügen wir der `ListBox4` im `Workbook_Open`-Ereignis den Eintrag Jahreszeiten als einziges Element hinzu.

```
'=====
' Auf CD      Buchdaten\Beispiele\05_Steuerelemente\
' Dateiname  05_02_ActiveXControls.xls
' Ereignis   DieseArbeitsmappe
'=====
```

```
Private Sub Workbook_Open()  
    Sheets("ListBox").ListBox4.AddItem "Jahreszeiten"  
End Sub
```

Wenn Sie mit der Maus das Listenfeld `ListBox4` anklicken, sollen anstelle der Überschrift *Jahreszeiten* die vier Jahreszeiten zur Auswahl angezeigt werden (siehe Abbildung 127).

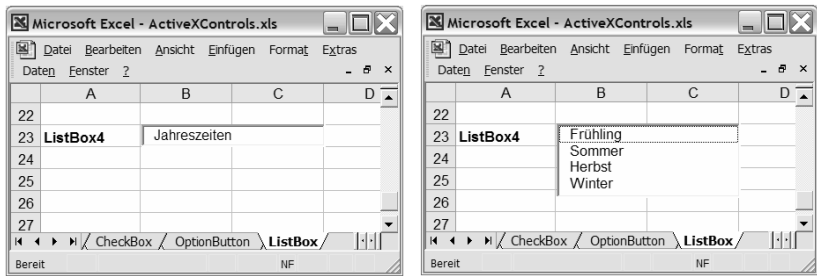


Abbildung 127: *GotFocus*-Ereignis ändert Listenfeld-Einträge und -Größe

Sobald das Listenfeld `ListBox4` den Fokus erhält, wird das hinterlegte `GotFocus`-Ereignis ausgelöst. An die Variant-Variable `Season` wird ein Array mit den einzufügenden Einträgen übergeben. Im folgenden `With`-Block wird zunächst der Eintrag »Jahreszeiten« durch die `RemoveItem`-Methode gelöscht. Der Wert `0` repräsentiert dabei den ersten Eintrag im Listenfeld.

Über die `List`-Eigenschaft wird unser komplettes Datenfeld `Season` mit seinen Werten in das Steuerelement kopiert. Dadurch ersparen Sie sich die bereits bekannte Verfahrensweise über eine `For Each...Next`-Anweisung in Kombination mit der `Add`-Methode.

Jetzt muss nur noch die Höhe `Height` des Listenfeldes angepasst werden, so dass alle Einträge vollständig angezeigt werden. Davon ausgehend, dass ein Listenfeldeintrag eine Höhe von 12,5 Punkt benötigt, wird diese Höhe mit der Anzahl der `Array`-Werte multipliziert. Diese wird über die `UBound`-Funktion ermittelt, welche die Obergrenze einer Datenfeld-Dimension zurückgibt. Da ein Datenfeld mit dem Wert `0` beginnt, wird zu dem ermittelten Wert die Zahl `1` addiert.

Hinweis

Damit der Index der Untergrenze eines Datenfeldes nicht bei 0, sondern bei 1 beginnt, setzen Sie die Option `Base`-Anweisung ein. Diese muss zu Beginn des Moduls vor allen Prozeduren und Deklarationen von Datenfeldern stehen, die Dimensionen enthalten. Folgende Anweisung besagt, dass die Untergrenze aller Datenfelder im verwendeten Modul mit dem Index 1 beginnt:

```
Option Base 1
```

Wenn Sie mit der Standardeinstellung 0 arbeiten wollen, ersetzen Sie die Zahl 1 durch 0, oder lassen Sie die Anweisung ganz einfach weg.

```
' =====  
' Auf CD      Buchdaten\Beispiele\05_Steuerelemente\  
' Dateiname  05_02_ActiveXControls.xls  
' Ereignis   Tabelle6 (ListBox)  
' =====
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

St
elem
te

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

```

Private Sub ListBox4_GotFocus()
    Dim intI    As Integer
    Dim Season As Variant

    Season = Array("Frühling", "Sommer", "Herbst", "Winter")

    With ListBox4
        .RemoveItem (0)
        .List = Season
        ' Größe des Listenfeldes anpassen
        .Height = 12.5 * (UBound(Season) + 1)
    End With
End Sub

```

Durch einen Doppelklick auf einen Eintrag in der `ListBox4` wird das hinterlegte `ListBox4_DblClick`-Ereignis ausgelöst. Der ausgewählte Wert wird in die Zelle E23 geschrieben. Anschließend wird die aktive Zelle im Tabellenblatt aktiviert. Das ist erforderlich, damit das Listenfeld `ListBox4` den Fokus wieder an die aktive Zelle des Tabellenblattes übergibt. Dabei wird zugleich das `ListBox4_LostFocus`-Ereignis ausgelöst.

```

' =====
' Auf CD      Buchdaten\Beispiele\05_Steuerelemente\
' Dateiname   05_02_ActiveXControls.xls
' Ereignis     Tabelle6 (ListBox)
' =====

Private Sub ListBox4_DblClick(ByVal Cancel As MSForms.ReturnBoolean)
    Range("E23") = ListBox4.Text
    ' löst LostFocus-Ereignis aus
    ActiveCell.Activate
End Sub

```

Nachdem das Listenfeld verlassen wurde, soll der Ausgangszustand der `ListBox4` mit der Themenüberschrift »Jahreszeiten« wieder hergestellt werden. Dazu wird die Höhe des Listenfeldes angepasst, alle vorhandenen Einträge mit der `Clear`-Methode gelöscht und die gewünschte Zeichenkette wieder mit der `AddItem`-Methode dem Listenfeld hinzugefügt.

```

' =====
' Auf CD      Buchdaten\Beispiele\05_Steuerelemente\
' Dateiname   05_02_ActiveXControls.xls
' Ereignis     Tabelle6 (ListBox)
' =====

Private Sub ListBox4_LostFocus()
    With ListBox4
        .Height = 15
        .Clear
        .AddItem "Jahreszeiten"
    End With
End Sub

```

155 Mehrpaltiges Listenfeld füllen

Über die `ListFillRange`-Eigenschaft eines Listenfeldes oder Kombinationsfeldes können Sie einen Zellenbereich auf einem Tabellenblatt einem Steuerelement zuweisen. Die Werte im angegebenen Bezug werden im genannten Objekt angezeigt.

```
Worksheets("Tabelle1").ListBox1.ListFillRange = "Tabelle1!A1:A12"
```

Mit dieser Methode können Sie keine einzelnen Einträge im Steuerelement löschen. Um die Liste zu entfernen, weisen Sie der `ListFillRange`-Eigenschaft einen Leerstring zu.

```
Worksheets("Tabelle1").ListBox1.ListFillRange = ""
```

Um einen Bereich aus mehreren Spalten in einem Listenfeld anzuzeigen, reicht es nicht aus, diesen Bereich nur der `ListFillRange`-Eigenschaft zuzuweisen. Zusätzlich muss die Anzahl der Spalten an die `ColumnsCount`-Eigenschaft übergeben und die jeweilige Spaltenbreite in `ColumnWidths` hinterlegt werden (siehe Abbildung 128). Spaltenbreiten werden in Punkten gemessen. Sie können die Angaben aber auch in Zentimeter-Einheiten (zum Beispiel 2,5 cm) im Eigenschaftsfenster angeben. Sobald Sie die Zeile verlassen, rechnet Excel die verwendete Einheit in Punkte um. Die `ColumnWidths`-Eigenschaft kann für einzelne oder alle Spalten leer sein. Wenn Sie gar keine Spaltenbreite bestimmen wollen, lassen Sie die Zeile der `ColumnWidths`-Eigenschaft leer. Wenn für einzelne Spalten keine bestimmte Größe festgelegt werden soll, schreiben Sie ein Semikolon (;) ohne voranstehenden Wert an die Position der betreffenden Spalte.

Hinweis

Wenn `ColumnWidths` leer ist, wird die Spaltenbreite berechnet, indem die Breite des Steuerelements gleichmäßig auf alle Spalten der Liste aufgeteilt wird. Überschreitet die Summe der angegebenen Spaltenbreiten die Gesamtbreite des Steuerelements, wird die Liste innerhalb des Steuerelements linksbündig angeordnet und eine oder mehrere der ganz rechts befindlichen Spalten werden nicht angezeigt. Der Benutzer kann dann mit Hilfe der horizontalen Bildlaufleiste durch die Liste blättern und die ganz rechts befindlichen Spalten anzeigen lassen.

Wenn die `ColumnHeads`-Eigenschaft auf `True` gesetzt ist, wird die Zeile oberhalb der `ListFillRange`-Angabe als Überschriftenzeile angenommen. Wenn `ListFillRange` bei Zeile 1 beginnt, werden die Spaltenbezeichnungen der Excel-Tabelle selbst (SPALTE H, SPALTE I usw.) als Spaltenüberschriften verwendet.

Es ist nicht möglich, über die `ListFillRange`-Eigenschaft einen Bereich aus nicht zusammenhängenden Zellen in einem Listenfeld anzuzeigen. Dazu müssen die einzelnen Bereiche dem Listenfeld über eine `For Each...Next`-Anweisung hinzugefügt werden.

Die `ListBox6` soll in diesem Beispiel mit den Spalten H und L der Tabelle aus Abbildung 128 gefüllt werden.

Die erste Spalte eines Listenfeldes wird einfach mit der `Add`-Methode gefüllt. Weitere Spalten werden über die `List`-Eigenschaft angesprochen. Dabei wird zuerst der Zeilen- und dann der Spaltenindex angegeben. Beide Index-Werte beginnen bei 0.

Grundlagen

Allgemein

Datum und Zeit

Formeln

Steuerelemente

Befehlsliste

Objekt

Diagramm

Ereignisse

User-Form

Web/Mail

Extern

Gemischtes

Spezial

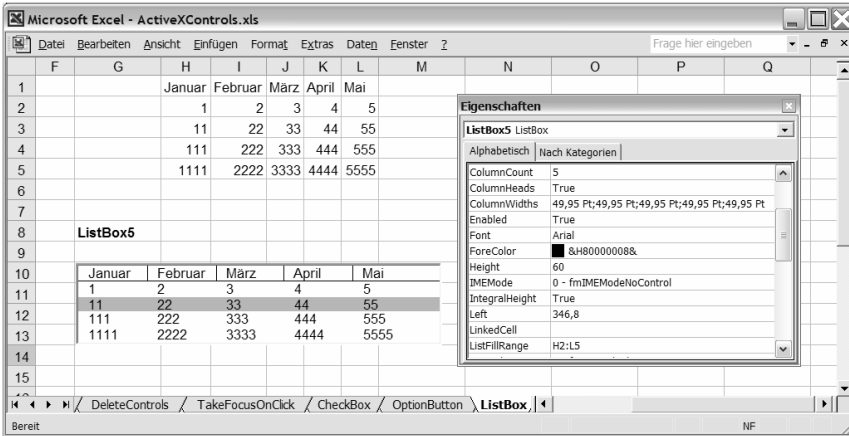


Abbildung 128: Mehrspaltiges Listenfeld mit Spaltenüberschriften

```
List(0, 1) ' Listenfeld(Zeile 1, Spalte 2)
```

Achtung

Bevor in einem Listenfeld einem Feld einer Spalte ein Wert hinzugefügt werden kann, müssen die vorangegangenen Spaltenfelder derselben Zeilenhöhe gefüllt sein.

Bei jedem Schleifendurchlauf wird die zu füllende Listenzeile um eine Zeile hochgezählt und mit dem fortlaufenden Zellenwert aus den angegebenen Bereichen rngA und rngB gefüllt. Die ListBox6 wird durch einen Klick auf eine Befehlsschaltfläche gefüllt.

```
' =====
' Auf CD      Buchdaten\Beispiele\05_Steuerelemente\
' Dateiname   05_02_ActiveXControls.xls
' Ereignis    Tabelle6 (ListBox)
' =====
```

```
Private Sub CommandButton1_Click()
    Dim rngA As Range
    Dim rngB As Range
    Dim intI As Integer

    Set rngA = Me.Range("H1:H5")
    Set rngB = Me.Range("L1:L5")

    With ListBox6
        .Clear
        .ColumnCount = 2

        For intI = 0 To rngA.Cells.Count - 1
            ' erste Spalte füllen
            .AddItem rngA.Cells(intI + 1).Value
            ' zweite Spalte füllen
```

```
        .List(intI, 1) = rngB.Cells(intI + 1).Value
    Next intI

End With
End Sub
```

156 Kombinationsfeld nach Auswahl füllen

Die Handhabung von Kombinationsfeldern `ComboBox` ist ähnlich der von Listenfeldern. Kombinationsfelder lassen sich nicht als aufgeklappte Listen anzeigen, sie funktionieren wie ein Drop-Down-Feld. Wenn Sie auf den Pfeil eines Kombinationsfeldes klicken, klappt die Liste mit den vorhandenen Einträgen auf. Durch Klick auf einen Eintrag wird dieser in das Sichtfenster des Steuerelementes kopiert und die Liste wieder zugeklappt.

In diesem Beispiel wird die `ComboBox1` per Klick auf eine Befehlsschaltfläche `CommandButton1` mit den Werten `Deutsch` und `Englisch` gefüllt. Die `ListIndex`-Eigenschaft erhält den Wert `0`. Dadurch wird der erste Wert der Liste *Deutsch* im Steuerelement angezeigt.

```
' =====
' Auf CD      Buchdaten\Beispiele\05_Steuerelemente\
' Dateiname   05_02_ActiveXControls.xls
' Ereignis    Tabelle7 (ComboBoxBox)
' =====

Private Sub CommandButton1_Click()
    With ComboBox1
        .Clear
        .AddItem "Deutsch"
        .AddItem "Englisch"
        ' Anzeige bei Start des Programms: "Deutsch"
        .ListIndex = 0
    End With
End Sub
```

Wenn der Anwender auf einen Eintrag im Kombinationsfeld klickt, wird das hinterlegte `ComboBox1_Change`-Ereignis ausgelöst. Zu Beginn der Prozedur werden zwei Array-Datenfelder an die Variablen `varGerman` und `varEnglish` übergeben. Über eine `Select Case`-Anweisung wird der ausgewählte Wert überprüft und der `List`-Eigenschaft das dazugehörige Datenfeld übergeben. Die `ComboBox2` weist nach einem Pfeilklick eine Auflistung mit einer deutschen bzw. englischen `CountDown`-Liste auf.

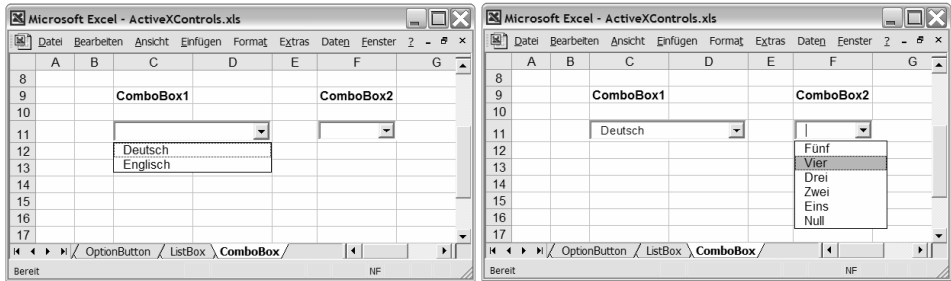


Abbildung 129: Kombinationsfeld nach Auswahl füllen

Grundlagen

Allgemein

Datum
Zeit

Formeln

Steuerelemente

Befehlsleisten

Objekt

Diagramm

Ereignisse

User-Form

Web/
Mail

Extern

Gemischtes

Spezial

Das `ComboBox`-Steuerelement verfügt nicht über die `TakeFocusOnClick`-Eigenschaft. Dadurch behält es standardmäßig weiterhin den Fokus, nachdem eine Auswahl getroffen wurde. Der Fokus wird an das Tabellenblatt zurückgegeben, indem der aktive Zellbereich aktiviert wird.

```
'=====
' Auf CD      Buchdaten\Beispiele\05_Steuerelemente\
' Dateiname   05_02_ActiveXControls.xls
' Ereignis    Tabelle7 (ComboBoxBox)
'=====

Private Sub ComboBox1_Change()
    Dim varGerman As Variant
    Dim varEnglish As Variant

    varGerman = _
        Array("Fünf", "Vier", "Drei", "Zwei", "Eins", "Null")

    varEnglish = _
        Array("Five", "Four", "Three", "Two", "One", "Zero")

    With ComboBox2

        Select Case ComboBox1.Value
            Case "Deutsch"
                .List = varGerman
            Case "Englisch"
                .List = varEnglish
            Case Else
                .Clear
        End Select

        ' bei Start ersten Wert der Liste anzeigen
        If .ListCount > 0 Then .ListIndex = 0

    End With

    ' Fokus an Tabellenblatt zurückgeben
    Selection.Activate
End Sub
```

Wenn der Anwender einen Eintrag aus der `ComboBox2` wählt, wird deren `DropDown`-Liste zugeklappt und die Auswahl erscheint im Steuerelement. Im ausgelösten `ComboBox2_Change`-Ereignis wird der Fokus anschließend an das Tabellenblatt zurückgegeben.

```
'=====
' Auf CD      Buchdaten\Beispiele\05_Steuerelemente\
' Dateiname   05_02_ActiveXControls.xls
' Ereignis    Tabelle7 (ComboBoxBox)
'=====

Private Sub ComboBox2_Change()
    ' Fokus an Tabellenblatt zurückgeben
```

```
Selection.Activate
End Sub
```

157 Passworteingabe durch Sternchen ersetzen

Sie können ein Textfeld der STEUERELEMENT-TOOLBOX verwenden, um wichtige Informationen, z.B. Kennwörter oder Sicherheitscodes, während und nach der Eingabe zu schützen, indem die eingegebenen Zeichen durch so genannte »Platzhalterzeichen« ersetzt werden. Verwenden Sie dazu die PasswordChar-Eigenschaft des TextBox-Objektes.

Geben Sie auf Ihrer Tastatur ein beliebiges einzelnes Zeichen für die PasswordChar-Eigenschaft ein, damit dieses Zeichen anstelle des Zeichens angezeigt wird, das der Benutzer tatsächlich eingibt.

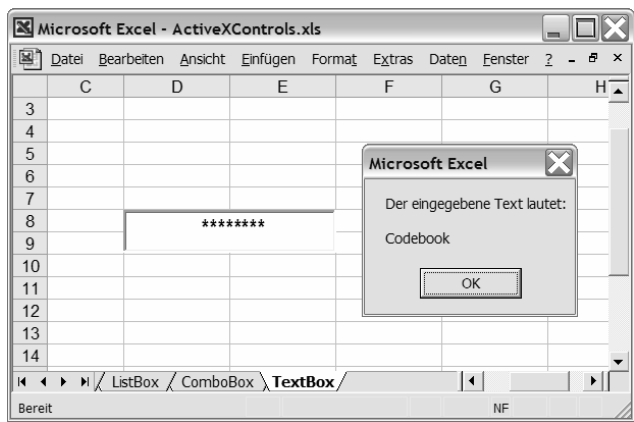


Abbildung 130: Text mit Platzhalterzeichen anzeigen

Über eine Befehlsschaltfläche im Tabellenblatt wird ein Meldungsfenster MsgBox mit dem enthaltenen Text des Textfeldes angezeigt.

```
'=====
' Auf CD      Buchdaten\Beispiele\05_Steuerelemente\
' Dateiname  05_02_ActiveXControls.xls
' Ereignis   Tabelle8 (TextBox)
'=====

Private Sub CommandButton1_Click()
    MsgBox "Der eingegebene Text lautet:" & _
        vbCrLf & vbCrLf & _
        TextBox1.Text
End Sub
```

Mit einer If...Then...Else-Anweisung ist es auf diese Weise möglich, ein vorgegebenes Passwort zu überprüfen, um zum Beispiel ein bestimmtes Tabellenblatt anzuzeigen oder sonstige Befehle auszuführen. Wenn das eingegebene Passwort nicht mit dem im Code hinterlegtem Passwort der Variablen strPassword übereinstimmt, wird die TextBox1 aktiviert und der enthaltene Text durch einen Leerstring ersetzt.

Grundlagen

Allgemein

Datum
Zeit

ungen

St
elem
te

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

```
'=====
' Auf CD      Buchdaten\Beispiele\05_Steuerelemente\
' Dateiname   05_02_ActiveXControls.xls
' Ereignis    Tabelle8 (TextBox)
'=====
```

```
Private Sub CommandButton2_Click()
    Dim strPassword As String

    strPassword = "Codebook"

    If TextBox1.Text = strPassword Then
        MsgBox "Das Paßwort '" & TextBox1.Text & "' ist richtig!"
    Else
        MsgBox "Das Paßwort '" & TextBox1.Text & "' ist falsch!"

        With TextBox1
            .Activate
            .Value = ""
        End With
    End If
End Sub
```

Wenn Sie kein Zeichen für die `PasswordChar`-Eigenschaft angeben, zeigt das Steuerelement das Zeichen an, das der Benutzer eingibt. Wenn Sie einen Leerschritt eingeben, wird der eingegebene Text nicht angezeigt, ist aber dennoch im Textfeld vorhanden.

In unserer Beispieldatei wird bei einem Klick auf die Schaltfläche PLATZHALTERZEICHEN ein Eingabefenster `InputBox` angezeigt. Geben Sie ein einzelnes Zeichen oder einen Leerschritt ein, um den Text der `TextBox1` entsprechend anzuzeigen. Wenn Sie die Eingabezeile leer lassen, wird der Text in den verwendeten Zeichen dargestellt.

```
'=====
' Auf CD      Buchdaten\Beispiele\05_Steuerelemente\
' Dateiname   05_02_ActiveXControls.xls
' Ereignis    Tabelle8 (TextBox)
'=====
```

```
Private Sub CommandButton3_Click()
    Dim strChar As String

    strChar = InputBox( _
        "Wählen Sie ein einzelnes Zeichen auf Ihrer Tastatur aus.", _
        "Platzhalterzeichen", "*")

    If Len(strChar) > 1 Then Exit Sub

    With TextBox1
        .PasswordChar = strChar
    End With
End Sub
```

158 Textfeld-Werte umwandeln

Wie aus dem Namen »Textfeld« schon herauszuhören ist, wird in diesem Steuerelement Text verwaltet. Die im Eigenschaften-Fenster in `LinkedCell` angegebene Zelle zeigt alle Werte (Text-, Zahlen-, Datums- oder Währungswerte) des verknüpften Textfeldes linksbündig im Textformat an. Den Inhalt eines Textfeldes ermitteln Sie entweder über die `Text`- oder die `Value`-Eigenschaft. Beide Eigenschaften sind gleichwertig. Wenn Sie den Wert eines Textfeldes in eine Zelle schreiben oder innerhalb des Codes weiterverwenden bzw. einer Variablen übergeben, wird er weiterhin als Text vom Datentyp `String` verarbeitet.

```
' =====
' Auf CD      Buchdaten\Beispiele\05_Steuerelemente\
' Dateiname  05_02_ActiveXControls.xls
' Ereignis    Tabelle8 (TextBox)
' =====

Private Sub CommandButton4_Click()
    ' Text zurückgeben
    Range("H16").Value = TextBox2.Value
End Sub
```

Um die Werte aus einer `TextBox` typengerecht auszugeben, wenden Sie die gewünschten Typ-Umwandlungsfunktionen an.

Nachfolgende Tabelle informiert Sie über die gängigsten Umwandlungsfunktionen in Bezug auf Textfelder:

Funktion	Rückgabe typ	Bereich des Arguments <i>Ausdruck</i>
CCur	Currency	Ein Datentyp mit einem Wertebereich von -922.337.203.685.477,5808 bis 922.337.203.685.477,5807. Verwenden Sie diesen Datentyp für Berechnungen von Währungen und für Berechnungen mit festgelegten Nachkommastellen, bei denen es besonders auf Genauigkeit ankommt.
CDate	Date	Ein Datentyp, der Datums- und Zeitangaben als reelle Zahlen speichert. Datumsvariablen werden als 64-Bit-Zahlen (8 Bytes) gespeichert. Der Wert links vom Dezimalzeichen steht für ein Datum, der Wert rechts vom Dezimalzeichen für eine Uhrzeit.
CDbl	Double	Ein Datentyp, der Fließkommazahlen mit doppelter Genauigkeit als 64-Bit-Zahlen im Wertebereich von -1.79769313486231E308 bis -4,94065645841247E-324 für negative Werte und von 4,94065645841247E-324 bis 1,79769313486232E308 für positive Werte enthält.
CInt	Integer	Ein Datentyp, der ganzzahlige Variablen enthält und diese als ganze Zahlen mit 2 Byte im Wertebereich von -32.768 bis 32.767 speichert. Nachkommastellen werden gerundet. Der Datentyp Integer wird auch verwendet, um Aufzählungswerte darzustellen.
Cvar	Variant	Numerische Werte im Bereich des Typs Double. Nichtnumerische Werte im Bereich des Typs String.

Tabelle 33: Typ-Umwandlungsfunktionen

Grund-lagen

Allge-mein

Datu-Zeit

tungen

St-elem-te

Befehl-leisten

Objekt

Dia-gramm

Ereig-nisse

User-Form

Web/Mail

Extern

Ge-misch-tes

Specia

Bevor Sie einen `TextBox`-Wert als `Zahl` weiterverwenden bzw. in eine Zelle schreiben, wandeln Sie ihn über die `Cdbl`-Funktion in einen Wert vom Datentyp `Double` um.

Hinweis

Wenn Sie versuchen, Text mit der `Cdbl`-Funktion umzuwandeln, wird ein Fehler verursacht. Um dies zu verhindern, wird vorab über die `IsNumeric`-Funktion in einer `If...Then...Else`-Anweisung überprüft, ob es sich bei dem Wert um eine Zahl handelt.

```
'=====
' Auf CD      Buchdaten\Beispiele\05_Steuerelemente\
' Dateiname   05_02_ActiveXControls.xls
' Ereignis    Tabelle8 (TextBox)
'=====

Private Sub CommandButton5_Click()
    ' Zahl zurückgeben
    With Range("H19")
        If IsNumeric(TextBox2.Text) Then
            .Value = Cdbl(TextBox2.Value)
        Else
            .Value = TextBox2.Value
        End If
    End With
End Sub
```

Im folgenden Beispiel wird bei einem Klick auf die Befehlsschaltfläche `Datum` in Zelle schreiben ein eingegebenes Datum von der `TextBox2` in die Zelle H22 im Datumsformat geschrieben.

```
'=====
' Auf CD      Buchdaten\Beispiele\05_Steuerelemente\
' Dateiname   05_02_ActiveXControls.xls
' Ereignis    Tabelle8 (TextBox)
'=====

Private Sub CommandButton6_Click()
    ' Datum zurückgeben
    With Range("H22")
        If IsDate(TextBox2.Text) Then
            .Value = CDate(TextBox2.Value)
        Else
            .Value = TextBox2.Value
        End If
    End With
End Sub
```

159 Zeilenumbruch in Textfeld (TextBox) per Code einfügen

Immer wieder taucht die Frage auf, wie man in einer Textbox einen Zeilenwechsel per Code realisieren kann. Normalerweise genügt es ja, die `EINGABE`- oder `ENTER`-Taste zu drücken, und schon hat man einen Zeilenwechsel. Mit einem Textfeld funktioniert dies standardmäßig nicht. Nur wenn Sie die `MultiLine`-Eigenschaft auf `True` setzen, wird der eingegebene Text automatisch bei Erreichen des Textfeldrandes umbrochen und in der nächsten Zeile fortgeführt. Einen manuellen

Zeilenumbruch erzeugen Sie bei aktivierter MultiLine-Eigenschaft mit der Tastenkombination `Strg` + `↵`.

Um wie gewohnt mit der Eingabe- oder Entertaste einen Zeilenumbruch hervorzurufen, setzen Sie das `KeyDown`-Ereignis des `TextBox`-Objektes ein. Bei jedem Tastendruck innerhalb des angegebenen Steuerelementes wird es ausgelöst und überprüft das eingegebene Zeichen bzw. die Taste.

```
'=====
' Auf CD      Buchdaten\Beispiele\05_Steuerelemente\
' Dateiname  05_02_ActiveXControls.xls
' Ereignis    Tabelle8 (TextBox)
'=====

Private Sub TextBox3_KeyDown( _
    ByVal KeyCode As MSForms.ReturnInteger, _
    ByVal Shift As Integer)

    If KeyCode = vbKeyReturn Then
        TextBox3.Text = TextBox3.Text & vbCrLf
    End If

End Sub
```

160 Mit Drehfeld (SpinButton) Monatswerte verändern

Mit einem Drehfeld `SpinButton` kann ein Zahlenwert innerhalb eines vorgegebenen Bereichs eingestellt werden. Über die Eigenschaften `Min` und `Max` werden die Unter- und Obergrenze eines Drehfeld-Steuerelements vorgegeben. Durch das Klicken auf ein Drehfeld-Steuerelement wird die `Value`-Eigenschaft des Steuerelements verändert.

In diesem Beispiel wurde die `Min`- und `Max`-Eigenschaft des `SpinButton1` entsprechend der Anzahl Monate auf 1 und 12 festgesetzt. Jeder Wert des Drehfeldes stellt dadurch einen Monatswert dar. Durch das Klicken auf die Pfeile des Drehfeldes wird der Zähler um eine Zahl nach oben oder unten verändert. Die Einstellungen am Drehfeld wirken sich in diesem Beispiel sowohl auf den Wert in der `TextBox1` als auch auf die `ComboBox1` aus. Umgekehrt verändert sich der Wert des Drehfeldes, wenn der `TextBox`- oder der Kombinationfeld-Wert geändert wird. Alle drei Steuerelemente leben sozusagen in Abhängigkeit voneinander (siehe Abbildung 131).

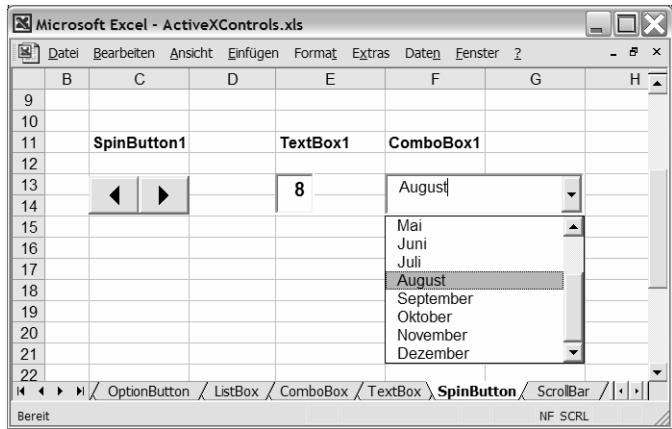


Abbildung 131: Voneinander abhängige Steuerelemente

Grundlagen

Allgemein

Datum
Zeit

Diagramme

Steuerelemente

Befehle
leisten

Objekt

Diagramm

Ereignisse

User-Form

Web/
Mail

Extern

Gemischtes

Spezial

Um das Kombinationsfeld zu Beginn mit den 12 Monatsnamen zu füllen, setzen wir das Worksheet_Activate-Ereignis ein. Wenn das Tabellenblatt SPINBUTTON aktiviert wird, wird die ComboBox1 in einer For...Next-Anweisung über die MonthName-Funktion mit den Monatsnamen gefüllt. Damit die ComboBox1 nicht bei jedem Aktivieren des Tabellenblattes erneut gefüllt wird, überprüfen wir eventuell vorhandene Einträge vorab mit der ListCount-Eigenschaft. Die Prozedur wird beendet, wenn bereits Einträge vorhanden sind.

```
'=====
' Auf CD      Buchdaten\Beispiele\05_Steuerelemente\
' Dateiname  05_02_ActiveXControls.xls
' Ereignis    Tabelle9 (SpinButton)
'=====

Private Sub Worksheet_Activate()
    Dim intI As Integer

    If ComboBox1.ListCount > 0 Then Exit Sub

    For intI = 1 To 12
        ComboBox1.AddItem MonthName(intI)
    Next intI
End Sub
```

Durch das Klicken auf einen der beiden Pfeile des Drehfeldes wird das hinterlegte Change-Ereignis ausgelöst. In der Prozedur wird der neue Zahlenwert des SpinButton1 an die TextBox1 weitergegeben. Die MonthName-Funktion wandelt den Zahlenwert des SpinButton1 in den entsprechenden Monatsnamen um und wird dann in der ComboBox1 als Monatsnamen angezeigt.

```
'=====
' Auf CD      Buchdaten\Beispiele\05_Steuerelemente\
' Dateiname  05_02_ActiveXControls.xls
' Ereignis    Tabelle9 (SpinButton)
'=====

Private Sub SpinButton1_Change()
    TextBox1.Value = SpinButton1.Value
    ' Monatszahl in Monatsnamen umwandeln
    ComboBox1.Value = MonthName(SpinButton1.Value)
End Sub
```

Wenn Sie durch eine Eingabe im Textfeld den Wert der TextBox1 verändern, wird das hinterlegte Change-Ereignis der TextBox1 ausgelöst. Bevor die Werte des SpinButton- und ComboBox-Steuerelementes nach dessen Wert eingestellt werden können, muss überprüft werden, ob es sich bei der Eingabe um einen gültigen Zahlenwert im Bereich von 1 bis 12 für die Monate handelt.

```
'=====
' Auf CD      Buchdaten\Beispiele\05_Steuerelemente\
' Dateiname  05_02_ActiveXControls.xls
' Ereignis    Tabelle9 (SpinButton)
'=====
```

```
Private Sub TextBox1_Change()  
  
    With TextBox1  
        If Not IsNumeric(.Value) Or .Value < 1 Or .Value > 12 Then  
            ' TextBox-Eingabe durch Wertabgleich aus SpinButton  
            ' rückgängig machen  
            .Value = SpinButton1.Value  
        Else  
            SpinButton1.Value = .Value  
            ' Monatszahl in Monatsnamen umwandeln  
            ComboBox1.Value = MonthName(.Value)  
        End If  
    End With  
End Sub
```

Damit auch das Dreh- und Textfeld entsprechend der Auswahl im Kombinationsfeld eingestellt wird, verwenden wir das `Change`-Ereignis der `ComboBox1`.

Um den ausgeschriebenen Monatsnamen aus dem Kombinationsfeld in eine Zahl umzuwandeln, wenden wir die `Month`-Funktion an. Sie gibt einen Wert vom Typ `Integer` zurück, der den Monat als ganze Zahl im Bereich von 1 bis 12 angibt. Diese ermittelte Monatszahl wird an die `Value`-Eigenschaften des `SpinButton1` und `TextBox1`-Objektes übergeben.

```
'=====
```

' Auf CD	Buchdaten\Beispiele\05_Steuerelemente\
' Dateiname	05_02_ActiveXControls.xls
' Ereignis	Tabelle9 (SpinButton)

```
'=====
```

```
Private Sub ComboBox1_Change()  
    ' Monatsname in Monatszahl umwandeln  
    SpinButton1.Value = Month("1." & ComboBox1.Value)  
    TextBox1.Value = Month("1." & ComboBox1.Value)  
End Sub
```

161 Mit Bildlaufleiste (ScrollBar) zu Datum wechseln

Ebenso wie das Drehfeld vermindert oder erhöht die Bildlaufleiste den Wert einer Zahl. Ein Bildlaufleisten-Steuerelement `ScrollBar` ist ein eigenständiges Steuerelement. Es sieht genauso aus wie die Bildlaufleiste in bestimmten Objekten, z.B. in einem Listefeld, oder wie der `DropDown`-Bereich eines Kombinationsfeldes. Im Gegensatz zu diesen Elementen ist das eigenständige Bildlaufleisten-Steuerelement jedoch kein integraler Bestandteil irgendeines anderen Steuerelements. Sie können ein horizontales oder vertikales Bildlaufleisten-Steuerelement erstellen, indem Sie die Ziehpunkte zur Größenänderung in horizontaler oder vertikaler Richtung ziehen.

In diesem Beispiel haben wir in Spalte A alle Datumswerte des aktuellen Jahres untereinander dargestellt. Die Zelle A10 weist durch die Formel `=DATUM(JAHR(HEUTE());1;1)` den 01. Januar des heutigen Jahres auf. Darunter liegende Zellen zählen zu ihrer Vorgängerzelle jeweils einen Tag hinzu (`=A10+1`). Die Bildlaufleiste soll nun die Datumswerte des aktuellen Jahres repräsentieren. Bei Veränderung ihres Wertes soll die Zelle mit dem entsprechenden Datum markiert und an oberster

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

St
elem
te

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

Stelle der Datumsreihe angezeigt werden (siehe Abbildung 132). Der Beschriftungstext des Bezeichnungsfeldes Label1 weist ebenfalls den Wert der ScrollBar1 im Datumsformat auf. Die TextBox1 dient dazu, ein Datum einzutragen, zu dem über den CommandButton2 gewechselt wird.

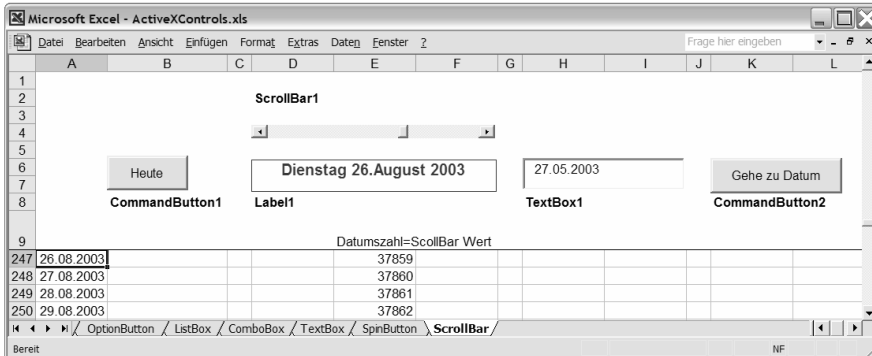


Abbildung 132: Bildlaufleiste mit Datumswerten des derzeitigen Jahres

Da Excel Datumswerte intern als fortlaufende Zahlen verwaltet, weisen wir der Min-Eigenschaft des ScrollBar1-Steuerelementes den 01. Januar zu. Dazu wird der Datumswert mit der CLng-Funktion in eine Zahl vom Datentyp Long umgewandelt.

```
ScrollBar1.Min = CLng(DateSerial(Year(Date), 1, 1))
```

Diese Berechnung liefert im Jahr 2003 die Zahl 37622. Der maximale Wert der Bildlaufleiste wird auf die gleiche Weise berechnet. Die ermittelte Zahl 37986, welche den 31.12.2003 darstellt, wird an die Max-Eigenschaft zugewiesen. Diese Einstellungen werden beim Aktivieren des Tabellenblattes SCROLLBAR der Beispieldatei im Worksheet_Activate-Ereignis vorgenommen. Zusätzlich wird der aktuelle Tag an die Value-Eigenschaft zugewiesen.

```
' =====
' Auf CD      Buchdaten\Beispiele\05_Steuerelemente\
' Dateiname   05_02_ActiveXControls.xls
' Ereignis    Tabelle10 (ScrollBar)
' =====
```

```
Private Sub Worksheet_Activate()
    With ScrollBar1
        .Min = CLng(DateSerial(Year(Date), 1, 1))
        .Max = CLng(DateSerial(Year(Date), 12, 31))
        .Value = CLng(Date)
    End With
End Sub
```

Wenn Sie den Wert der Bildlaufleiste durch Klicken auf einen der äußeren Pfeile oder Verschieben des Bildlauffeldes verändern, wird das hinterlegte Change-Ereignis ausgelöst. Das Bezeichnungsfeld Label1 erhält über die Caption-Eigenschaft den neuen Wert der ScrollBar1 im Datumsformat.

Über die Goto-Methode wird die Datumszelle des eingestellten Wertes markiert. Gleichzeitig werden die nötigen Bildläufe durchgeführt, damit die Zelle als obere linke Zelle des Fensters bzw. der

eingestellten Fenster-Fixierung erscheint. Die Berechnung der Datumszeile wird im ersten Parameter der Cells-Eigenschaft durchgeführt. Im zweiten Parameter, welcher die Spalte angibt, wird der Spaltenbuchstabe A angegeben. Sie können stattdessen auch die entsprechende Spaltenzahl 1 für die Spalte A angeben.

```
'=====
' Auf CD      Buchdaten\Beispiele\05_Steuerelemente\
' Dateiname   05_02_ActiveXControls.xls
' Ereignis    Tabelle10 (ScrollBar)
'=====

Private Sub ScrollBar1_Change()
    Label1.Caption = Format(ScrollBar1.Value, "DDDD DD.MMMM YYYY")

    Application.Goto _
        Cells(ScrollBar1.Value - ScrollBar1.Min + 10, "A"), True
End Sub
```

Über den CommandButton1 mit der Beschriftung HEUTE wird der Wert der ScrollBar1 auf den aktuellen Tag eingestellt. Diese Veränderung löst wiederum das Change-Ereignis aus, wodurch das eingestellte Datum der Systemsteuerung markiert wird.

```
'=====
' Auf CD      Buchdaten\Beispiele\05_Steuerelemente\
' Dateiname   05_02_ActiveXControls.xls
' Ereignis    Tabelle10 (ScrollBar)
'=====

Private Sub CommandButton1_Click()
    ScrollBar1.Value = CLng(Date)
End Sub
```

Geben Sie im Tabellenblatt SCROLLBAR der Beispieldatei in der TEXTBOX1 ein Datum des aktuellen Jahres ein. Die Eingabe eines Datums kann, wie in einer Zelle, auf verschiedene Arten erfolgen. Nachfolgende Schreibweisen repräsentieren den 01.01. des aktuellen Jahres:

- ▶ 01.01.2003
- ▶ 1.1.3
- ▶ 1-1-3
- ▶ 1-1
- ▶ 1/1

Klicken Sie auf die Schaltfläche GEHE ZU DATUM, um das hinterlegte Click-Ereignis aufzurufen und den Wert des ScrollBar1-Steuerelementes auf dieses Datum einzustellen. Um sicherzustellen, dass es sich bei der Datumseingabe um ein gültiges Datum im derzeitigen Jahr handelt, müssen einige Abfragen erfolgen. Excel gibt eine Fehlermeldung aus, wenn Sie versuchen, Text in ein Datum oder eine Zahl umzuwandeln, oder wenn ein Wert außerhalb der Min- und Max-Eigenschaft an das ScrollBar-Element zugewiesen werden soll.

Bevor das Datum der TextBox1 an die Value-Eigenschaft der ScrollBar1 übergeben werden kann, wird es mit der CDate-Funktion vom Text- in das Datums-Format umgewandelt. Die CLng-Funk-

Grundlagen

Allgemein

Datum und Zeit

Funktionen

Steuerelemente

Befehlsleisten

Objekte

Diagramme

Ereignisse

User-Formen

Web/Mail

Extern

Gemischtes

Spezial

tion wandelt dieses Datum in eine Zahl um und weist diese als neuen Wert an die ScrollBar1. Dieser Vorgang löst wiederum das ScrollBar1_Change-Ereignis aus und das Datum des Textfeldes wird im Bezeichnungsfeld Label1 angezeigt und die entsprechende Datumzelle markiert.

```
'=====
' Auf CD      Buchdaten\Beispiele\05_Steuerelemente\
' Dateiname   05_02_ActiveXControls.xls
' Ereignis    Tabelle10 (ScrollBar)
'=====

Private Sub CommandButton2_Click()
    If Not IsDate(TextBox1.Value) Then
        MsgBox "Eingabe ist kein gültiger Datumswert!"
        Exit Sub
    End If

    If CLng(CDate(TextBox1.Value)) < ScrollBar1.Min Or _
        CLng(CDate(TextBox1.Value)) > ScrollBar1.Max Then

        MsgBox "Das eingegebene Datum liegt nicht im Jahr " & _
            & Year(Date)
    Else
        ScrollBar1.Value = CLng(CDate(TextBox1.Value))
    End If
End Sub
```

162 Steuerelemente mit Tastatur aktivieren

In einem Formular UserForm können Sie die Aktivierreihenfolge der Steuerelemente vorgeben. Mit der -Taste wird das nächste, mit  +  das vorherige Steuerelement aktiviert. Auf einem Tabellenblatt mit ActiveX-Elementen der STEUERELEMENT-TOOLBOX ist diese Aktivierung mittels Tastaturanschlägen standardmäßig nicht vorgesehen. Um von einem zum anderen Element zu wechseln, müssen Sie die Elemente mit der linken Maustaste anklicken.

Wenn Sie eine beliebige Taste auf Ihrer Tastatur drücken, wird zunächst das KeyDown-, danach das KeyPress- und abschließend das KeyUp-Ereignis des aktiven Steuerelementes ausgelöst. Da das KeyPress-Ereignis nur auf alphanumerische Tasten sowie RETURN, ESC und STRG-Kombinationen reagiert, ist es für eine allgemein gültige Tastaturverwaltung nicht ausreichend. Die KeyDown- und KeyUp-Ereignisse sind gleichwertig und überwachen jegliche Tastenanschläge.

Beim Aufruf des KeyDown-Ereignisses werden die gedrückten Tasten an die Parameter KeyCode und Shift der Prozedur übergeben. Excel stellt die benötigten Parameter in der Codeschablone selbstständig zur Verfügung. Am Beispiel einer ComboBox lautet die Aufrufzeile wie folgt:

```
Private Sub ComboBox1_KeyDown( _
    ByVal KeyCode As MSForms.ReturnInteger, _
    ByVal Shift As Integer)
```

Die gedrückten Tasten werden in Form einer KeyCode-Konstante an den Parameter KeyCode übergeben. Nachfolgende Übersicht zeigt die Konstanten, welche die einzelnen Tasten auf Ihrer Tastatur darstellen.

Konstante	Beschreibung	Konstante	Beschreibung
vbKeyBack	RÜCKTASTE	vbKeyLeft	NACH-LINKS-TASTE
vbKeyTab	TAB-TASTE	vbKeyUp	NACH-OBEN-TASTE
vbKeyClear	ENTF-TASTE	vbKeyRight	NACH-RECHTS-TASTE
vbKeyReturn	EINGABETASTE	vbKeyDown	NACH-UNTEN-TASTE
vbKeyShift	UMSCHALTTASTE	vbKeySelect	AUSWAHLTASTE
vbKeyControl	STRG-TASTE	vbKeyPrint	DRUCK
vbKeyMenu	MENÜTASTE	vbKeyExecute	AUSFÜHREN-TASTE
vbKeyPause	PAUSE-TASTE	vbKeySnapshot	SNAPSHOT-TASTE
vbKeyCapital	FESTSTELLTASTE	vbKeyInsert	EINFG-TASTE
vbKeyEscape	ESC-TASTE	vbKeyDelete	LÖSCHEN-TASTE
vbKeySpace	LEERTASTE	vbKeyHelp	HILFETASTE
vbKeyPageUp	BILD-AUF-TASTE	vbKeyNumlock	NUM-TASTE
vbKeyPageDown	BILD-AB-TASTE	VbKeyA – VbKeyZ	A-TASTE – Z-Taste
vbKeyEnd	ENDE-TASTE	vbKey0 – vbKey9	0-TASTE – 9-Taste
vbKeyHome	POS1-TASTE	vbKeyF1 – vbKeyF12	F1-Taste – F12-Taste

Tabelle 34: Tasten-Code-Konstanten

Die folgenden Konstanten repräsentieren die Zahlen auf der Zehnertastatur:

Konstante	Beschreibung	Konstante	Beschreibung
vbKeyNumpad0	0-TASTE	vbKeyNumpad8	8-TASTE
vbKeyNumpad1	1-TASTE	vbKeyNumpad9	9-TASTE
vbKeyNumpad2	2-TASTE	vbKeyMultiply	MULTIPLIKATIONSZEICHEN-TASTE (*)
VbKeyNumpad3	3-TASTE	vbKeyAdd	PLUSZEICHEN-TASTE (+)
VbKeyNumpad4	4-TASTE	vbKeyReturn	EINGABETASTE
VbKeyNumpad5	5-TASTE	vbKeySubtract	MINUSZEICHEN-TASTE (–)
vbKeyNumpad6	6-TASTE	vbKeyDecimal	DEZIMALPUNKT-TASTE
vbKeyNumpad7	7-TASTE	vbKeyDivide	DIVISIONSZEICHEN-TASTE

Tabelle 35: Typ-Tasten-Code-Konstanten auf der Zehnertastatur

Der Parameter Shift gibt eine Zahl zurück, welche die Zusatztasten SHIFT, STRG und ALT repräsentieren.

Rückgabewert	Zusatztaste
0	Keine Zusatztaste
1	
2	 oder 
4	

Tabelle 36: Rückgabewerte des Shift-Parameters

Grund-lagen

Allge-mein

Datu Zeit

tungen

St elem te

Befehl leisten

Objekt

Dia-gramm

Ereig-nisse

User-Form

Web/ Mail

Extern

Ge-misch tes

Specia

In der `KeyDown`-Ereignisprozedur werden die Variablen `KeyCode` und `Shift` über eine `If...Then...Else`- oder `Select Case`-Anweisung auf ihren Wert überprüft. Entsprechend dieser Auswertung wird ein anderes Steuerelement aktiviert oder ein anderer Befehl oder Makro ausgeführt.

Folgende Anweisung aktiviert die `TextBox1`, wenn die `TABULATOR`-Taste gedrückt wurde.

```
If KeyCode = vbKeyTab Then TextBox1.Activate
```

Sie müssen für jedes Steuerelement auf einem Tabellenblatt ein eigenständiges `KeyDown`-Ereignis definieren, um einen kontinuierlichen Wechsel zwischen den Steuerelementen zu ermöglichen.

Im nachfolgenden Rezept verdeutlichen wir Ihnen, dass es möglich ist, eine Menüführung nur mit der Tastatur auszuführen (siehe Abbildung 133).

163 Tabellenblatt-Zugriffe mit Steuerelementen verwalten

Die Beispieldatei zu diesem Rezept enthält ein Tabellenblatt `START` mit verschiedenen Steuerelementen (siehe Abbildung 133). Weiter stehen mehrere Tabellenblätter für verschiedene Benutzer zur Verfügung. Beim Öffnen der Datei oder Aktivierung des Start-Blattes ist nur das Start-Blatt sichtbar. Über ein Kombinationsfeld wird eine Liste der vorhandenen, aber ausgeblendeten Blätter angezeigt. Wenn Sie das Benutzerkonto `ADMIN` auswählen und das richtige Passwort eingeben, werden alle Blätter der Beispieldatei angezeigt. Die anderen Benutzer erhalten jeweils nur das Tabellenblatt entsprechend ihrem Namen.

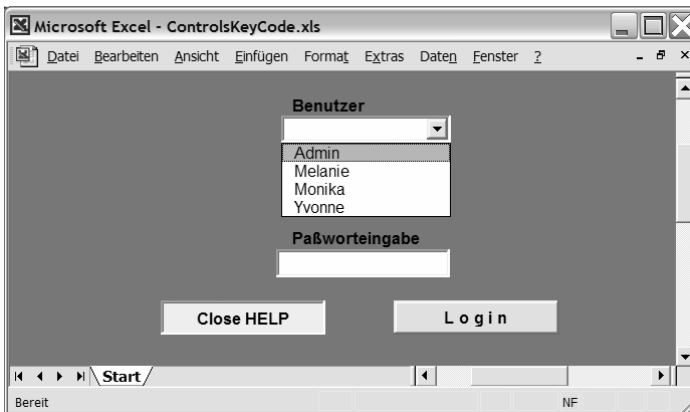


Abbildung 133: Formular-Maske auf einem Tabellenblatt

Diese Liste der verfügbaren Benutzerkonten und Passwörter ist im Tabellenblatt `ADMIN` hinterlegt. Der Bereich `A2:A5` wurde der `ListFillRange`-Eigenschaft der `ComboBox1` zugewiesen.

Wenn die Datei geöffnet wird, setzen wir im `Workbook_Open`-Ereignis die `EnableSelection`-Eigenschaft auf `xlNoSelection`. Da das Tabellenblatt `START` geschützt und alle Zellen im Tabellenblatt gesperrt sind, kann dadurch keine Zelle markiert werden. Es ist also nur möglich, die vorhandenen Steuerelemente zu bedienen. Den Fokus übergeben wir an die `ComboBox1`, indem dieses Element aktiviert wird.

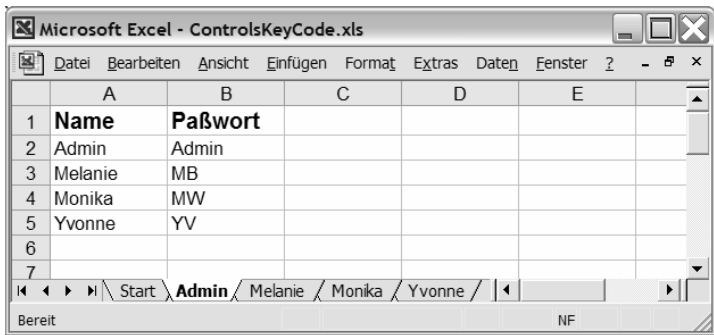


Abbildung 134: Benutzerkonten mit Passwort vermerken

```
'=====
' Auf CD      Buchdaten\Beispiele\05_Steuerelemente\
' Dateiname  05_03_XControlsKeyCode.xls
' Ereignis   DieseArbeitsmappe
'=====

Private Sub Workbook_Open()
    With Worksheets("Start")
        .EnableSelection = xlNoSelection
        .ComboBox1.Activate
    End With
End Sub
```

Nachdem Sie im Kombinationsfeld `ComboBox1` einen Benutzer ausgewählt haben, können Sie mit der TABULATOR-Taste zum nächsten Steuerelement `TextBox1` wechseln. Dies wird durch die hinterlegte `ComboBox1_KeyDown`-Ereignisprozedur ermöglicht.

In der äußeren `If...Then...Else`-Anweisung wird die gedrückte Taste überprüft. Entspricht diese der angegebenen Konstante `vbKeyTab`, steht fest, dass die `[↹]`-Taste gedrückt wurde. Jetzt muss noch ermittelt werden, ob der Benutzer zusätzlich die `[⇧]`-Taste betätigt hat, da mit dieser Kombination ein Rückwärtsspringen zu Steuerelementen gängig ist. Die Variable `Shift` weist den Wert 1 auf, wenn die `[⇧]`-Taste gedrückt wurde. Demzufolge wird das vorherige Steuerelement `CommandButton1` aktiviert. Andernfalls wird die nachfolgende `TextBox1` aktiviert.

```
'=====
' Auf CD      Buchdaten\Beispiele\05_Steuerelemente\
' Dateiname  05_03_XControlsKeyCode.xls
' Ereignis   Tabelle1 (Start)
'=====

Private Sub ComboBox1_KeyDown( _
    ByVal KeyCode As MSForms.ReturnInteger, _
    ByVal Shift As Integer)

    If KeyCode = vbKeyTab Then
        If Shift = 1 Then
            ' Rückwärts springen
```

- Grundlagen
- Allgemein
- Datum und Zeit
- Formeln
- Diagramme
- Ereignisse
- User-Formen
- Web/Mail
- Extern
- Gemischtes
- Spezial

```

        CommandButton1.Activate
    Else
        ' Vorwärts springen
        TextBox1.Activate
    End If
End If
End Sub

```

Die gleiche Vorgehensweise gilt für das Textfeld. Mit  bzw.   wechseln Sie zum nachfolgenden bzw. vorherigen Steuerelement.

```

' =====
' Auf CD      Buchdaten\Beispiele\05_Steuerelemente\
' Dateiname   05_03_XControlsKeyCode.xls
' Ereignis     Tabelle1 (Start)
' =====

Private Sub TextBox1_KeyDown( _
    ByVal KeyCode As MSForms.ReturnInteger, _
    ByVal Shift As Integer)

    If KeyCode = vbKeyTab Then
        If Shift = 1 Then
            ' Rückwärts springen
            ComboBox1.Activate
        Else
            ' Vorwärts springen
            ToggleButton1.Activate
        End If
    End If
End Sub

```

Wenn das Textfeld aktiviert wird, also den Fokus erhält, soll ein eventuell noch vorhandener Text entfernt werden. Diesen Vorgang nehmen wir im `TextBox1_GotFocus`-Ereignis vor.

```

' =====
' Auf CD      Buchdaten\Beispiele\05_Steuerelemente\
' Dateiname   05_03_XControlsKeyCode.xls
' Ereignis     Tabelle1 (Start)
' =====

Private Sub TextBox1_GotFocus()
    TextBox1.Text = ""
End Sub

```

Bei der verwendeten Umschaltfläche `HELP` und Befehlsschaltfläche `LOGIN` wird neben dem `KeyDown`-Ereignis auch das `Click`-Ereignis eingesetzt. Dieses wird normalerweise ausgeführt, wenn der Benutzer mit der Maustaste darauf klickt.

Ein Klick auf die Umschaltfläche `HELP` zeigt ein Bezeichnungsfeld mit Informationen der verwendeten Passwörter an bzw. blendet es aus (siehe Abbildung 135).

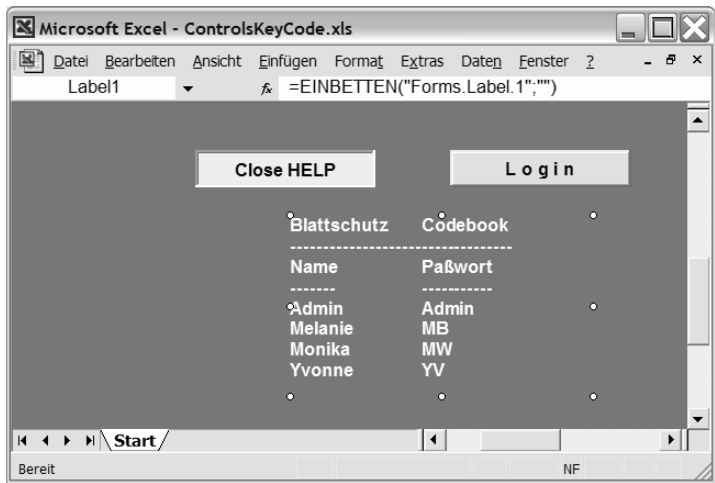


Abbildung 135: Bezeichnungsfeld über Umschaltfläche anzeigen

Nachdem die `Visible`-Eigenschaft des `Label1` nach dem Wert des `ToggleButton1` eingestellt wurde, wird der Beschriftungstext der Umschaltfläche über die `IIf`-Funktion festgelegt.

```
'=====
' Auf CD      Buchdaten\Beispiele\05_Steuerelemente\
' Dateiname   05_03_XControlsKeyCode.xls
' Ereignis    Tabelle1 (Start)
'=====

Private Sub ToggleButton1_Click()
    Label1.Visible = ToggleButton1.Value

    ToggleButton1.Caption = _
        IIf(ToggleButton1.Value, "Close HELP", "H E L P")
End Sub
```

In realen Projekten ist es natürlich nicht ratsam, die geheimen Passwörter gleich mitzuliefern. Sie können dem Anwender aber andere wichtige Informationen über das Bezeichnungsfeld mitteilen.

Die Gestaltung und Formatierung des Textes eines Bezeichnungsfeldes gestaltet sich über die Eingabezeile im Eigenschaften-Fenster sehr schwierig. Umfangreicher Text muss in eine Zeile geschrieben werden. Es können keine manuellen Zeilenumbrüche oder Ausrichtungen innerhalb des Textes vorgenommen werden. In unserem Beispiel haben wir deshalb den Beschriftungstext über die Prozedur `LabelText` an das Bezeichnungsfeld zugewiesen. Durch die `vbTab`-Konstante wird ein Tabulatorzeichen erzeugt, welches die linksbündige Ausrichtung in der rechten Hälfte des Labels ermöglicht. Mit `vbCrLf` wird eine Kombination aus Wagenrücklauf und Zeilenvorschub bewirkt. Das Ergebnis zeigt Ihnen die Abbildung 135.

```
'=====
' Auf CD      Buchdaten\Beispiele\05_Steuerelemente\
' Dateiname   05_03_XControlsKeyCode.xls
' Modul       mdl_01_LabelText
```

- Grundlagen
- Allgemein
- Datum und Zeit
- Formularelemente
- Steuerelemente
- Beifügen und löschen
- Objekt
- Diagramm
- Ereignisse
- User-Form
- Web/Mail
- Extern
- Gemischtes
- Spezial

```
' =====

Sub LabelText()
    Worksheets("Start").Label1.Caption = _
        "Blattschutz" & vbTab & "Codebook" & vbCrLf & _
        "-----" & vbCrLf & _
        "Name" & vbTab & vbTab & "Paßwort" & vbCrLf & _
        "-----" & vbTab & vbTab & "-----" & vbCrLf & _
        "Admin" & vbTab & vbTab & "Admin" & vbCrLf & _
        "Melanie" & vbTab & vbTab & "MB" & vbCrLf & _
        "Monika" & vbTab & vbTab & "MW" & vbCrLf & _
        "Yvonne" & vbTab & vbTab & "YV"
End Sub
```

Wenn Sie bei aktivierter Umschaltfläche die Tabulatortaste drücken, soll – wie bei den bereits genannten Steuerelementen – das nachfolgende bzw. vorherige Element aktiviert werden. Diese Prüfung wird in der folgenden `ToggleButton1_KeyDown`-Prozedur in einer `Select Case`-Anweisung vorgenommen.

Ein Druck auf die -Taste bei aktivierter Umschaltfläche soll den Wert des `ToggleButton1` umkehren und dadurch das Bezeichnungsfeld ein- oder ausblenden. Die Eingabetaste hat also die gleiche Auswirkung auf das Steuerelement wie ein Mausklick. Durch den `Not`-Operator erreichen wir eine logische Umkehrung der `Value`-Eigenschaft des `ToggleButton1`. Durch die Umkehrung, also Änderung des `ToggleButton1`-Wertes, wird das `Change`-Ereignis dieses Elementes ausgelöst. Damit es bei der weiteren Verarbeitung von Code nicht zu Fehlern kommt, wird die Variable `KEYCODE` auf 0 zurückgesetzt.

```
' =====
' Auf CD      Buchdaten\Beispiele\05_Steuerelemente\
' Dateiname   05_03_XControlsKeyCode.xls
' Ereignis    Tabelle1 (Start)
' =====

Private Sub ToggleButton1_KeyDown( _
    ByVal KeyCode As MSForms.ReturnInteger, _
    ByVal Shift As Integer)

    Select Case KeyCode
        Case vbKeyTab
            If Shift = 1 Then
                ' Rückwärts springen
                TextBox1.Activate
            Else
                ' Vorwärts springen
                CommandButton1.Activate
            End If
        Case vbKeyReturn
            ' ToggleButton-Wert umkehren
            ToggleButton1.Value = Not ToggleButton1.Value
            KeyCode = 0
    End Select
End Sub
```

Bei einem Klick auf die Befehlsschaltfläche LOGIN wird zunächst geprüft, ob ein Benutzerkonto ausgewählt und ein Passwort in das Textfeld eingegeben wurde. Anschließend wird das hinterlegte Passwort des ausgewählten Benutzers im Tabellenblatt ADMIN ausgelesen und an die Variable strPassword übergeben. Je nach ausgewähltem Benutzerkonto werden entweder alle Tabellenblätter der Beispieldatei oder nur jenes des ausgewählten Benutzers angezeigt und aktiviert.

```
' =====  
' Auf CD      Buchdaten\Beispiele\05_Steuerelemente\  
' Dateiname   05_03_XControlsKeyCode.xls  
' Ereignis    Tabelle1 (Start)  
' =====
```

```
Private Sub CommandButton1_Click()  
    Dim strPassword As String  
    Dim wks As Worksheet  
  
    ' ComboBox1 überprüfen  
    If ComboBox1.Value = "" Then  
        MsgBox "Wählen Sie ein Benutzerkonto.", _  
            vbOKOnly + vbInformation  
        ComboBox1.Activate  
        Exit Sub  
    End If  
  
    ' TextBox1 überprüfen  
    If TextBox1.Text = "" Then  
        MsgBox "Geben Sie ein Paßwort ein.", _  
            vbOKOnly + vbInformation  
        TextBox1.Activate  
        Exit Sub  
    End If  
  
    ' Passwort aus Sheet "Admin" auslesen  
    strPassword = Worksheets("Admin").Columns("A") _  
        .Find(ComboBox1.Value, , xlValues) _  
        .Offset(0, 1).Value  
  
    ' Bei falscher Passwortheingabe verlasse Prozedur  
    If TextBox1.Text <> strPassword Then  
        MsgBox "Das Paßwort ist falsch!", vbCritical  
  
        With TextBox1  
            .Text = ""  
            .Activate  
        End With  
  
        Exit Sub  
    End If  
  
    Application.ScreenUpdating = False  
    If ComboBox1.Value = "Admin" Then  
        ' Alle Blätter einblenden  
        For Each wks In ThisWorkbook.Worksheets
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

St
elem
te

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

```

        wks.Visible = xlSheetVisible
    Next wks
Else
    ' Alle Blätter bis auf Start-Blatt ausblenden
    For Each wks In ThisWorkbook.Worksheets
        If wks.Name <> "Start" Then _
            wks.Visible = xlSheetVeryHidden
    Next wks
End If

    ' Tabellenblatt des Anmelders anzeigen
With Worksheets(ComboBox1.Value)
    .Visible = xlSheetVisible
    .Activate
End With

Application.ScreenUpdating = True
End Sub

```

Das Click-Ereignis des CommandButton1 können Sie auch auslösen, wenn Sie bei aktiviertem Element die Tasten  oder  drücken. Wenn in der Select Case-Anweisung die Konstante vbKeyReturn für die Eingabe-Taste ermittelt wird, bewirkt die Aufrufzeile des CommandButton1_Click-Ereignisses den Start desselben. Vergessen Sie nicht die Variable KeyCode auf 0 zurückzusetzen. Bei der weiteren Verarbeitung von Code kann es unter Umständen sonst zu Fehlermeldungen kommen.

```

' =====
' Auf CD      Buchdaten\Beispiele\05_Steuerelemente\
' Dateiname   05_03_XControlsKeyCode.xls
' Ereignis     Tabelle1 (Start)
' =====

Private Sub CommandButton1_KeyDown( _
    ByVal KeyCode As MSForms.ReturnInteger, _
    ByVal Shift As Integer)

    Select Case KeyCode
        Case vbKeyTab
            If Shift = 1 Then
                ' Rückwärts springen
                ToggleButton1.Activate
            Else
                ' Vorwärts springen
                ComboBox1.Activate
            End If
        Case vbKeyReturn
            CommandButton1_Click
            KeyCode = 0
    End Select
End Sub

```

Um den Ausgangszustand, also nur das Start-Blatt anzuzeigen, verwenden wir die Prozedur HideSheets. Nachdem die Benutzerblätter ausgeblendet wurden, wird das Tabellenblatt START geschützt und die Steuerelemente auf ihren Ausgangszustand zurückgesetzt. Abschließend wird die Datei gespeichert, damit sie beim nächsten Öffnen in diesem Zustand angezeigt wird.

```
' =====
' Auf CD      Buchdaten\Beispiele\05_Steuerelemente\
' Dateiname  05_03_XControlsKeyCode.xls
' Modul      mdl_02_HideSheets
' =====

Sub HideSheets()
    Dim wks As Worksheet

    Application.ScreenUpdating = False

    ' Alle Blätter bis auf Start-Blatt ausblenden
    For Each wks In ThisWorkbook.Worksheets
        If wks.Name <> "Start" Then
            wks.Visible = xlSheetVeryHidden
        End If
    Next wks

    Application.ScreenUpdating = True

    With Worksheets("Start")
        .Protect "Codebook"
        .ComboBox1.Value = ""
        .TextBox1.Text = ""
        .ToggleButton1.Value = False
    End With

    ThisWorkbook.Save
End Sub
```

Die Prozedur HideSheets wird beim Schließen der Datei und Aktivierung des Tabellenblattes START von den entsprechenden Ereignissen Workbook_BeforeClose und Worksheet_Activate aufgerufen.

```
' =====
' Auf CD      Buchdaten\Beispiele\05_Steuerelemente\
' Dateiname  05_03_XControlsKeyCode.xls
' Ereignis   DieseArbeitsmappe
' =====

Private Sub Workbook_BeforeClose(Cancel As Boolean)
    HideSheets
End Sub

' =====
' Auf CD      Buchdaten\Beispiele\05_Steuerelemente\
' Dateiname  05_03_XControlsKeyCode.xls
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

St
elem
te

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

```
' Ereignis   Tabelle1 (Start)
' =====

Private Sub Worksheet_Activate()
    HideSheets
End Sub
```


Befehlsleisten

Microsoft Excel enthält dutzende von integrierten Befehlsleisten und kann um beliebig viele benutzerdefinierte erweitert werden. Es gibt drei Typen von Befehlsleisten:

- ▶ Menüleisten
- ▶ Symbolleisten
- ▶ Kontextmenüs

Alle drei Befehlstypen können zusätzliche benutzerdefinierte Befehlsleisten und jeweils beliebig viele Steuerelemente enthalten.

Es gibt in Excel zwei integrierte Menüleisten, wobei immer nur eine Menüleiste angezeigt werden kann. Die ARBEITSBLATT-MENÜLEISTE wird angezeigt, wenn ein Tabellenblatt aktiviert ist. Bei einem aktivierten Diagramm oder Diagrammblatt wird stattdessen die DIAGRAMMMENÜLEISTE mit diagrammspezifischen Funktionen zur Verfügung gestellt. Neben einer Menüleiste kann keine weitere Symbolleiste platziert werden.

Im Menü ANSICHT | SYMBOLLEISTEN Registerkarte SYMBOLLEISTEN finden Sie eine Auflistung der Menüleisten, der gängigsten integrierten und allen benutzerdefinierten Symbolleisten.

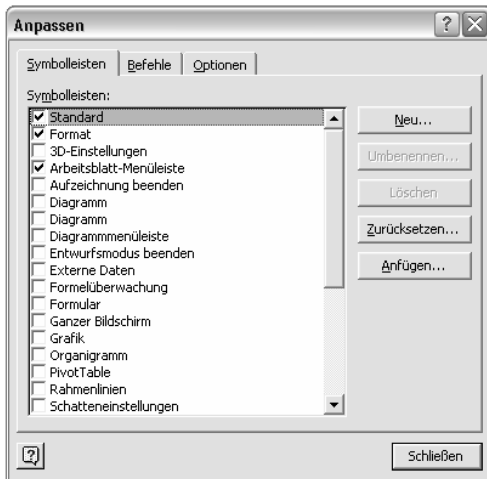


Abbildung 136: Anpassen von Symbolleisten und Befehlen

Kontextmenüs sind bewegliche Befehlsleisten, welche auf der Arbeitsoberfläche von Excel, im Tabellenblatt oder bei markierten Objekten aufgerufen werden können. Kontextmenüs werden mit der rechten Maustaste bzw. mit der Kontext-Taste der Tastatur angezeigt. Letztere befindet sich auf der rechten Seite der Schreibtastatur, neben der Leertaste, zwischen der Windows- und Strg-Taste. Kontextmenüs enthalten jeweils die wichtigsten Befehle zu dem angeklickten Objekt und können dieselben Steuerelementtypen wie Befehlsleisten enthalten. Auch das Verhalten der Steuerelemente in Kontextmenüs entspricht dem der Befehlsleisten.

164 Befehlsleistendatei finden

Wenn Sie die integrierten Symbolleisten anpassen oder eine benutzerdefinierte Symbolleiste hinzufügen, werden Ihre Änderungen und die jeweilige Anordnung der Symbolleisten jeweils beim Beenden von Excel in der Symbolleistendatei *.xlb gespeichert. Je nach Excel-Version ist der Dateiname sowie der Speicherpfad auf Ihrer Festplatte unterschiedlich.

Excel-Version	Dateiname	Position
Excel 2003	Excel11.xlb	C:\Dokumente und Einstellungen\Benutzername\Anwendungsdaten\Microsoft\Excel\
Excel 2002	Excel10.xlb	C:\Dokumente und Einstellungen\Benutzername\Anwendungsdaten\Microsoft\Excel\
Excel 2000	Excel.xlb	C:\Dokumente und Einstellungen\Benutzername\Anwendungsdaten\Microsoft\Excel
Excel 97	username8.xlb	Windows-Ordner, wenn keine andere Position von der Administration festgelegt wurde

Tabelle 37: Name und Pfad der Befehlsleistendatei

Hinweis

Wenn sich in dem angegebenen Ordner zu Ihrer aktuellen Excel-Version keine *.xlb-Datei befindet, liegt das daran, dass noch keine Änderungen an den Symbolleisten vorgenommen wurden.

Unter Windows XP werden über die Windows-Suche START | SUCHEN die *.xlb-Dateien sowie der Ordner *Anwendungsdaten* nicht gefunden. Suchen Sie im Explorer manuell den Ordner über die Pfadangaben aus Tabelle 37, oder verwenden Sie die Prozedur aus diesem Beispiel.

Die folgende Prozedur ermittelt über die `Version`-Eigenschaft Ihre verwendete Excel-Version. Verwenden Sie eine Version vor Excel 2000, erscheint ein Meldungsfenster und die Prozedur wird beendet. Mittels der `Environ`-Funktion liefert die Umgebungsvariable `APPDATA` den Speicherpfad Ihrer *.xlb-Datei. Zusätzlich wird in der Zelle C5 ein Hyperlink zu dem entsprechenden Ordner hinterlegt. Klicken Sie im Anschluss an die Prozedur auf den Hyperlink, öffnet sich der Explorer und zeigt den Excel-Ordner mit den vorhandenen *.xlb-Dateien an.

Tipp

Ausführliche Informationen zu der `Environ`-Funktion finden Sie in der Kategorie »Allgemein«.

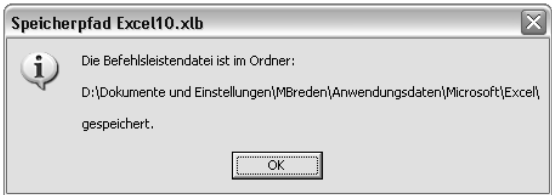


Abbildung 137: Meldungsfenster mit Speicherpfad der *.xlb-Dateien

<pre>'===== ' Auf CD Buchdaten\Beispiele\06_Befehlsleisten\ ' Dateiname 06_01_xlbFile.xls ' Modul mdl_01_xlbPath '=====</pre>		Grund- lagen
<pre>Sub xlbPath() Dim strFile As String Dim strPath As String Select Case Val(Application.Version) Case 9 strFile = "Excel" Case 10 strFile = "Excel10" Case 11 strFile = "Excel11" Case Else MsgBox "Die Excel-Version konnte nicht identifiziert" & _ "werden." & vbCrLf & _ "Die Prozedur wird beendet.", _ vbExclamation Exit Sub End Select strPath = Environ("APPDATA") & "\Microsoft\Excel" MsgBox "Die Befehlsleistendatei ist im Ordner:" & _ vbCrLf & vbCrLf & _ strPath & _ vbCrLf & vbCrLf & _ "gespeichert.", _ vbInformation, _ "Speicherpfad " & strFile & ".xlb" Range("C5").Hyperlinks.Add _ Anchor:=Range("C5"), _ Address:=strPath, _ TextToDisplay:=strFile & ".xlb" End Sub</pre>		Allge- mein
		Datu Zeit
		tungen
		Steuer elemen
		Befehl leisten
		Objekt
		Dia- gramm
		Ereig- nisse
		User- Form
		Web/ Mail
		Extern
		Ge- misch tes
		Specia

165 Befehlsleistendatei sichern

Wenn Sie von einer früheren Excel-Version auf eine neuere Excel-Version aufrüsten und die frühere Version von Excel nicht entfernen, bleiben die Symbolleistendateien der früheren Version zusätzlich im Excel-Ordner erhalten. Ersetzen Sie hingegen eine Excel-Version gegen eine andere, wird die ursprüngliche Symbolleistendatei mit der gelöschten Version entfernt.

Unter Windows und ganz allgemein am PC ist die gewohnte Arbeitsumgebung ein entscheidender Faktor. Daher sollten Sie dafür sorgen, dass Sie Ihre Einstellungen und die Symbolleisten, die Sie in Excel selbst definiert haben, regelmäßig sichern. Um eine gesicherte Befehlsleistendatei wieder herzustellen, öffnen Sie die Datei im Explorer mit einem Doppelklick oder über das Menü DATEI | ÖFFNEN.

Hinweis

Beim Beenden von Excel wird die originale *.*xl*b-Datei ohne Rückfrage mit den aktuellen Einstellungen überschrieben. Bedenken Sie dies bei einem Wechsel der *.*xl*b-Datei und sichern Sie eventuell vor dem Öffnen einer anderen Befehlsleistendatei die aktuellen Einstellungen. Sie können jedoch auch verhindern, dass die originale *Excel.xlb* geändert wird, wenn Sie das Schreibschutz-Attribut setzen.

Diese unscheinbaren *.*xl*b-Dateien ermöglichen es Ihnen auch, Ihre persönlichen Symbolleisten an jeden Arbeitsplatz ohne weiteres mitzunehmen: Speichern Sie diese Datei einfach unter einem beliebigen Namen ab, jedoch mit der Endung *.*xl*b. Kopieren Sie diese anschließend zum Beispiel auf Diskette. Haben Sie eigene Makros in die Symbolleiste integriert, müssen Sie lediglich darauf achten, dass diese in einem beliebigen Verzeichnis der Festplatte, beispielsweise *C:\EXCEL\Makro*, gespeichert sind. Kopieren Sie auch die Makros auf Ihre Diskette. Am anderen Arbeitsplatz müssen Sie diese dann lediglich in ein gleichnamiges Verzeichnis auf der Festplatte kopieren. Nunmehr sollten auch die Makros problemlos funktionieren.

Die folgende Prozedur berücksichtigt die Excel-Versionen von Excel 2000 bis Excel 2003. Von Ihrer aktuellen *.*xl*b-Datei wird im gleichen Ordner der Original-Datei eine Sicherheitskopie angelegt. In einer *Select Case*-Anweisung wird mittels der *Version*-Eigenschaft die verwendete Excel-Version ausgelesen, und der entsprechende Dateiname an die Variable *strFile* übergeben. An die Variable *strPath* wird der Speicherpfad der *.*xl*b-Dateien übergeben. Dieser setzt sich über die *Environ*-Funktion aus dem Pfad der Anwendungsdaten und einer weiteren Zeichenkette mit den Unterordnern zusammen.

Um Fehlermeldungen zu unterdrücken, wird in einer *If*-Anweisung über die *Dir*-Funktion überprüft, ob der angegebene Pfad und Dateiname existiert.

Die *FileCopy*-Anweisung kopiert aus dem Verzeichnis *strPath* die ermittelte Datei *strFile* in dasselbe Verzeichnis *strPath*. Der neue Dateiname setzt sich aus dem ursprünglichen Dateinamen mit dem Zusatz »_Backup« zusammen. Dadurch, dass die Datei kopiert wird, behält die Sicherheitskopie das Speicherdatum der Originaldatei.

```
'=====
' Auf CD      Buchdaten\Beispiele\06_Befehlsleisten\
' Dateiname   06_01_xlbFile.xls
' Modul       mdl_02_SaveXLB
'=====
```

```
Sub SaveXLB()
    Dim strFile As String
    Dim strPath As String

    Select Case Val(Application.Version)
        Case 9
            strFile = "Excel"
        Case 10
            strFile = "Excel10"
        Case 11
            strFile = "Excel11"
        Case Else
            MsgBox _
                "Die Excel-Version konnte nicht identifiziert werden." & _
```

```
        vbNewLine & _
        "Die Prozedur wird beendet.", _
        vbExclamation
    Exit Sub
End Select

strPath = Environ("APPDATA") & "\Microsoft\Excel\"

' Prüfung, ob Datei existiert
If Dir(strPath & strFile & ".xlb") = "" Then
    MsgBox _
        "Für die verwendete Version existiert " & _
        "noch keine Befehlsleistendatei.", _
        vbExclamation
    Exit Sub
Else
    FileCopy strPath & strFile & ".xlb", _
        strPath & strFile & "_Backup.xlb"
End If

MsgBox _
    "Die Sicherheitskopie: " & strFile & "_Backup.xlb" & _
    vbNewLine & vbNewLine & _
    "wurde im Ordner: " & strPath & _
    vbNewLine & _
    "gespeichert.", _
    vbInformation

End Sub
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

166 Symbolleistenkonfigurationen speichern

Sie können auch mehrere Symbolleistenkonfigurationen in verschiedenen Symbolleistendateien (*.xlb) speichern.

Erstellen Sie anhand des vorherigen Rezepts 165 eine Sicherheitskopie von der Original-*.xlb-Datei (Schritt 2 der Beispieldatei 06_01_xlbFile.xls). Nehmen Sie nun an den integrierten Menü- und Symbolleisten die gewünschten Änderungen vor. Erstellen Sie die benutzerdefinierten Symbolleisten, die gespeichert werden sollen, und blenden Sie dann die Symbolleisten in der gewünschten Anordnung und Größe ein. Beenden Sie Excel und öffnen Sie erneut die Beispieldatei. Die geänderten Einstellungen wurden in der Originaldatei gespeichert.

Die *.xlb-Datei soll mit den aktuellen Einstellungen (Benutzername und Standardspeicherpfad) laut den Angaben im Menü EXTRAS | OPTIONEN Registerkarte ALLGEMEIN in den Standardspeicherpfad kopiert werden.

Starten Sie in der Beispielmappe über die Schaltfläche SYMBOLLEISTENKONFIGURATION SPEICHERN folgende Prozedur, um die Original-*.xlb-Datei mit der aktuellen Symbolleistenkonfiguration zu kopieren. Um die ursprüngliche Ausgangssituation wieder herzustellen, wird die Originaldatei anschließend durch die Sicherheitskopie ersetzt. Es erfolgt eine erneuter Aufruf der Prozedur SaveXLB, um eine neue Sicherheitskopie zu erstellen.

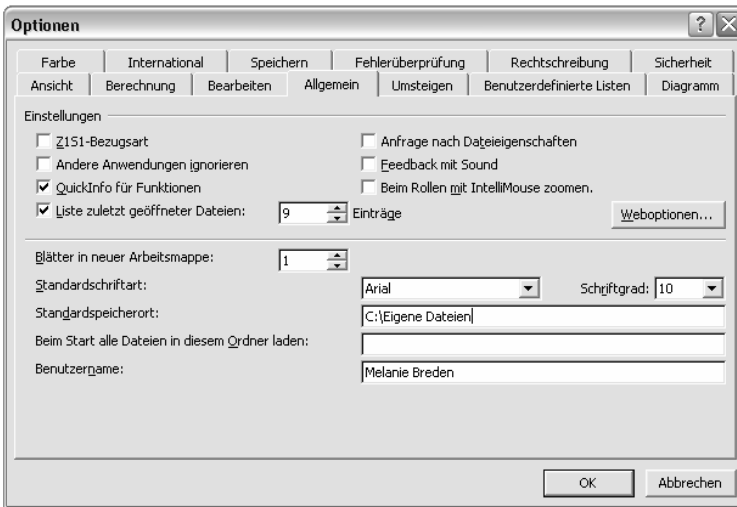


Abbildung 138: Benutzerdefinierte Angaben speichern

In diesem Beispiel gehen wir davon aus, dass Sie Excel 2002 verwenden. Benutzen Sie eine andere Version, ändern Sie im Code den entsprechenden Dateinamen der *.xlb-Datei. Die Vorgehensweise der Prozedur wird im Code durch Kommentare erläutert.

```
'=====
' Auf CD      Buchdaten\Beispiele\06_Befehlsleisten\
' Dateiname  06_01_xlbFile.xls
' Modul      mdl_03_Person\XLB
'=====
```

```
Sub Person\XLB()
    Dim strFile As String
    Dim strPath As String
    Dim strMyPath As String

    strFile = "Excel10.xlb"
    strPath = Environ("APPDATA") & "\Microsoft\Excel\"

    ' Standardspeicherpfad ermitteln
    strMyPath = Application.DefaultFilePath

    If strMyPath = "" Then
        MsgBox _
            "Es ist kein Standardspeicherort eingetragen.", _
            vbInformation
        Exit Sub
    Else
        If Right(strMyPath, 1) <> "\" Then
            strMyPath = strMyPath & "\"
        End If
    End If
End Sub
```

```
' Prüfung, ob Datei existiert
If Dir(strPath & strFile) = "" Then
    MsgBox _
        "Für die verwendete Version existiert " & _
        "keine Befehlsleistendatei.", _
        vbExclamation
Else
    ' Originaldatei kopieren
    FileCopy strPath & strFile, _
        strMyPath & Application.UserName & strFile

    ' Originaldatei löschen
    Kill strPath & strFile

    ' Backupdatei in Originaldatei umbenennen
    Name strPath & "Excel10" & "_Backup.xlb" As strPath & strFile

    ' Originaldatei öffnen
    Workbooks.Open strPath & strFile

    ' erneute Backupdatei erstellen
    SaveXLB
End If
End Sub
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

**Befehl
leisten**

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

167 Befehlsleistendatei laden

Um eine gespeicherte Konfiguration zu verwenden, klicken Sie im Menü DATEI auf ÖFFNEN und öffnen dann eine umbenannte *.xlb-Datei. Excel aktualisiert beim nächsten Beenden der Anwendung die Datei *Excel10.xlb* mit den aktuellen Einstellungen der Symbolleisten.

Hinweis

Um die Original-*.xlb-Datei dabei nicht zu überschreiben, laden Sie die Original-*.xlb-Datei, bevor Sie Excel beenden, oder aktivieren Sie das Schreibschutz-Attribut der Originaldatei.

Über die Schaltfläche SYMBOLLEISTENKONFIGURATION LADEN wird die Symbolleistendatei, welche im Standardspeicherpfad `DefaultFilePath` gespeichert ist, geöffnet und zeigt die gespeicherte Konfiguration an.

```
'=====
' Auf CD      Buchdaten\Beispiele\06_Befehlsleisten\
' Dateiname   06_01_xlbFile.xls
' Modul       mdl_04_OpenMyXLB
'=====

Sub OpenMyXLB()
    Dim strFile As String
    Dim strMyPath As String

    strFile = Application.UserName & "Excel10.xlb"
```

```

strMyPath = Application.DefaultFilePath

If strMyPath = "" Then
    MsgBox _
        "Es ist kein Standardspeicherort eingetragen.", _
        vbInformation
    Exit Sub
Else
    If Right(strMyPath, 1) <> "\" Then
        strMyPath = strMyPath & "\"
    End If
End If

If Dir(strMyPath & strFile) = "" Then
    MsgBox _
        "Unter Ihrem Namen existiert " & _
        "noch keine Befehlsleistendatei.", _
        vbExclamation
    Exit Sub
Else
    Workbooks.Open strMyPath & strFile
End If
End Sub

```

168 Originaldatei *.xlb laden

Um die originale Symbolleistenkonfiguration in Excel 2002 anzuzeigen, öffnen Sie die Datei *Excel10.xlb* (entsprechend der Datei mit dem Dateinamen Ihrer verwendeten Excel-Version). In unserer Beispieldatei erfolgt dieser Vorgang über die Schaltfläche ORIGINALDATEI LADEN. Dieser wurde die folgende Prozedur zugewiesen.

Hinweis

In diesem Beispiel gehen wir davon aus, dass Sie Excel 2002 verwenden. Benutzen Sie eine andere Version, ändern Sie im Code den entsprechenden Dateinamen der *.xlb-Datei.

```

'=====
' Auf CD      Buchdaten\Beispiele\06_Befehlsleisten\
' Dateiname   06_01_xlbFile.xls
' Modul       mdl_05_OpenOriginalXLB
'=====

Sub OpenOriginalXLB()
    Dim strPath As String

    strPath = Environ("APPDATA") & "\Microsoft\Excel\Excel10.xlb"

    If Dir(strPath) = "" Then
        MsgBox _
            "Für die verwendete Version existiert " & _
            "keine Befehlsleistendatei.", _
            vbExclamation
    End If
End Sub

```

```
Exit Sub
Else
    Workbooks.Open strPath
End If
End Sub
```

169 Befehlsleisten ermitteln

Die Prozedur zu diesem Beispiel listet alle integrierten und benutzerdefinierten Befehlsleisten im Tabellenblatt MyCOMMANDBARS auf. In einer For...Each-Schleife wird jede Befehlsleiste durchlaufen. Über die Type-Eigenschaft wird anschließend in einer Select Case-Anweisung der Typ der Befehlsleiste ermittelt und an die Variable strType übergeben.

Konstante	Wert	Befehlsleiste
msoBarTypeMenuBar	1	Menüleiste
msoBarTypeNormal	0	Symbolleiste
msoBarTypePopup	2	Kontextmenü

Tabelle 38: Konstanten des CommandBar-Objektes

Klicken Sie in der Beispieldatei auf die Schaltfläche BEFEHLSLEISTEN AUFLISTEN. Die hinterlegte Prozedur ListCommandBars wird gestartet und gibt im Tabellenblatt MYCOMMANDBARS zu jeder Leiste zeilenweise Informationen aus. Über die Index-Eigenschaft erfahren Sie die Indexnummer. strType gibt den ermittelten Leistentyp zurück.

Excel kennt zu jeder Befehlsleiste zwei Namen. Bei einer integrierten Befehlsleiste gibt die Name-Eigenschaft den US-englischen Namen der Befehlsleiste zurück. Mit der NameLocal-Eigenschaft erhalten Sie den übersetzten Namen. Der lokale Name einer integrierten Befehlsleiste wird auf der Titelleiste (wenn die Befehlsleiste nicht angedockt ist) und in der Liste der verfügbaren Befehlsleisten angezeigt. Dies ist unabhängig davon, wo diese Liste in der Containeranwendung angezeigt wird. Die BuiltIn-Eigenschaft gibt einen booleschen Wert zurück, der besagt, ob die Befehlsleiste integriert oder benutzerdefiniert ist. True steht dabei für integriert, False für benutzerdefiniert.

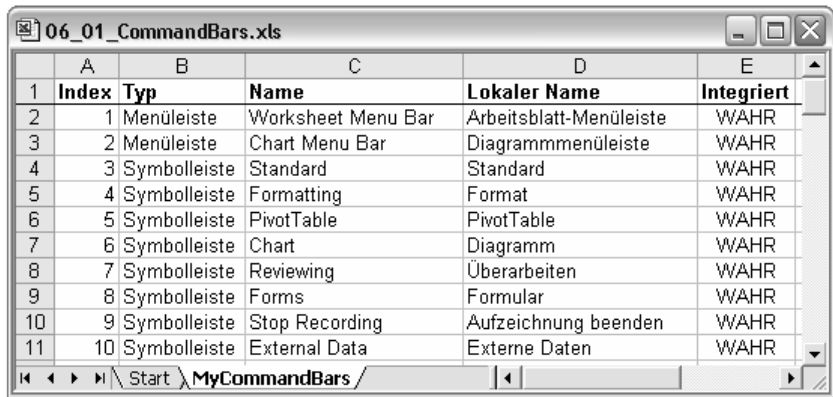


Abbildung 139: Auflistung aller Befehlsleisten

Grundlagen

Allgemein

Datum
Zeit

ungen

Steuer
elemente

Befehls
leisten

Objekt

Diagramm

Ereignisse

User-
Form

Web/
Mail

Extern

Gemischtes

Spezial

```

' =====
' Auf CD      Buchdaten\Beispiele\06_Befehlsleisten\
' Dateiname   06_02_CommandBarsAndControls.xls
' Tabelle     Start
' Modul       mdl_01_ListCommandBars
' =====

Sub ListCommandBars()
    Dim cmb As CommandBar
    Dim intRow As Integer
    Dim strType As String

    Application.ScreenUpdating = False
    intRow = 1

    With Worksheets("MyCommandBars")
        .Cells.Clear
        .Range("A1").Value = "Index"
        .Range("B1").Value = "Typ"
        .Range("C1").Value = "Name"
        .Range("D1").Value = "Lokaler Name"
        .Range("E1").Value = "Integriert"
        .Range("A1:E1").Font.Bold = True

        For Each cmb In Application.CommandBars
            Select Case cmb.Type
                Case msoBarTypeMenuBar
                    strType = "Menüleiste"
                Case msoBarTypeNormal
                    strType = "Symbolleiste"
                Case msoBarTypePopup
                    strType = "Kontextmenü"
            End Select

            intRow = intRow + 1
            .Cells(intRow, 1) = cmb.Index
            .Cells(intRow, 2) = strType
            .Cells(intRow, 3) = cmb.Name
            .Cells(intRow, 4) = cmb.NameLocal
            .Cells(intRow, 5) = cmb.BuiltIn
        Next cmb
    End With
    ' Benutzerdefinierte Ansicht anzeigen
    ActiveWorkbook.CustomViews("ShowList").Show
    Application.ScreenUpdating = True
End Sub

```

Im Tabellenblatt **START** sind die Spalten **F** und **G** zunächst ausgeblendet. Dieser Ansichts-Einstellung haben wir im Menü **ANSICHT | BENUTZERDEFINIERT ANSICHTEN** den Namen »Start« zugewiesen. In der benutzerdefinierten Ansicht *ShowList* sind diese beiden Spalten eingeblendet und zeigen weitere Schaltflächen zur Verwendung im folgendem Rezept.

170 Menüs und Steuerelemente ermitteln

Auf jeder Befehlsleiste können sich beliebig viele Steuerelemente befinden, so genannte Controls. Im Tabellenblatt `START` der Beispieldatei haben wir in der Zelle `F12` über den Menüpunkt `DATEN | GÜLTIGKEIT | LISTE` den benannten Bereich »`LokalerName`« zugewiesen. Dieser enthält den Bereich der Spalte `D` des ausgeblendeten Tabellenblattes `MYCOMMANDBARS`, mit den lokalen Namen der Befehlsleisten. Durch Auswahl einer Befehlsleiste aus der Gültigkeitsliste wird in Zelle `G12` über eine verschachtelte Tabellenfunktion der entsprechende englische Befehlsleistenname ausgegeben.

Bei Klick auf die Schaltfläche `SCHALTFLÄCHEN AUFLISTEN` wird die Prozedur `ListControls` aufgerufen. Auf die Variable `strcmb` wird ein Verweis auf die ausgewählte Befehlsleiste (aus Zelle `G12`) gesetzt. In der Zelle `G12` muss entweder die Indexnummer der Leiste oder der englische Name zur Identifizierung angegeben werden. In unserem Tabellenblatt wird die Zelle `G12` durch Auswahl der Nebenzelle (`F12`) ausgefüllt.

Über die `Add`-Methode wird an die letzte Stelle der Tabellenregister ein neues Tabellenblatt in die Arbeitsmappe eingefügt. Zu Beginn der `With`-Anweisung werden die Überschriften für das neue Tabellenblatt aufbereitet. Diese werden in den Bereich `A1:D2` geschrieben.

In der darauf folgenden `For Each...Next`-Anweisung wird jedes Steuerelement der Befehlsleiste durchlaufen. Über die `Type`-Eigenschaft wird ermittelt, ob es sich bei dem jeweiligen Befehlsleisten-Steuerelement um eine Schaltfläche vom Typ `msoControlButton` handelt. Ist dies der Fall, wird über die `CopyFace`-Methode das Symbol des angegebenen Schaltflächensteuerelements in die Zwischenablage kopiert. Mit der `Paste`-Methode fügen wir das Symbol der Schaltfläche, auch `FaceId` genannt, als eigenständige Grafik im Tabellenblatt ein. Da die Grafik durch den Kopierprozess bereits markiert ist, können wir mit der `Top`- und `Left`-Eigenschaft des `ShapeRange`-Objektes die Grafik in die gewünschte Zelle platzieren. In die jeweilige Zeile des Steuerelementes wird noch die `Index`-Nummer, die `ID` für ein integriertes Befehlsleisten-Steuerelement und die Beschriftung des Elementes ausgegeben.

Da das letzte eingefügte Grafiksymboll markiert ist, setzen wir die Markierung mit der `Goto`-Methode in die Zelle `A1` des aktiven Tabellenblattes. Das `Scroll`-Argument wird auf `True` gesetzt, um Bildläufe im Zielfenster durchzuführen.

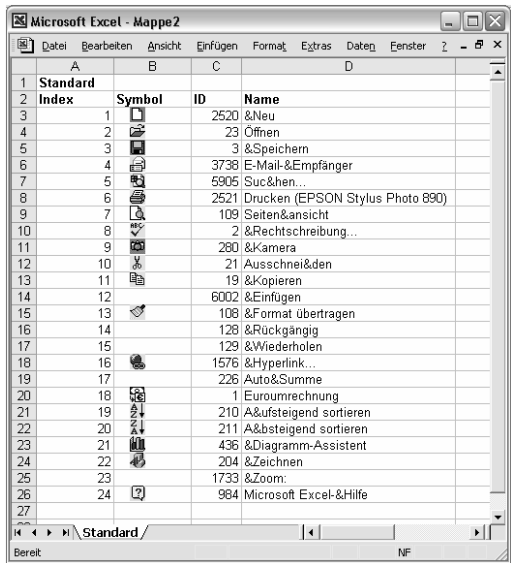


Abbildung 140: Schaltflächen der Standard-Symboleiste

Grund-lagen

Allge-mein

Datu Zeit

tungen

Steuer elemen

Befehl leisten

Objekt

Dia-gramm

Ereig-nisse

User-Form

Web/Mail

Extern

Ge-misch tes

Specia

```
'=====
' Auf CD      Buchdaten\Beispiele\06_Befehlsleisten\
' Dateiname   06_02_CommandBarsAndControls.xls
' Tabelle     Start
' Modul       mdl_02_ListControls
'=====
```

```
Sub ListControls()
    Dim cmb As CommandBar
    Dim ctl As CommandBarControl
    Dim strcmb As String
    Dim intRow As Integer

    intRow = 2
    On Error Resume Next
    strcmb = Range("G12").Value
    Set cmb = Application.CommandBars(strcmb)

    Worksheets.Add After:=Worksheets(Worksheets.Count)
    With ActiveSheet
        .Name = cmb.NameLocal
        .Range("A1").Value = cmb.Name
        .Range("A2").Value = "Index"
        .Range("B2").Value = "Symbol"
        .Range("C2").Value = "ID"
        .Range("D2").Value = "Name"
        .Range("A1:D2").Font.Bold = True

        For Each ctl In cmb.Controls
            intRow = intRow + 1
            If ctl.Type = msoControlButton Then
                ctl.CopyFace
                ActiveSheet.Paste
                Selection.ShapeRange.Top = Cells(intRow, 2).Top
                Selection.ShapeRange.Left = Cells(intRow, 2).Left + 13
            End If
            Cells(intRow, 1) = ctl.Index
            Cells(intRow, 3) = ctl.ID
            Cells(intRow, 4) = ctl.Caption
        Next ctl
    End With

    Application.Goto Range("A1"), True
End Sub
```

171 Menü-Auflistung im Direktfenster

Um ein komplettes Menü der Arbeitsblatt-Menüleiste zu deaktivieren, muss es über seine ID-Nummer oder die Beschriftung der Schaltfläche angesprochen werden. Folgende Prozedur listet zur schnellen Übersicht aller vorhandenen Menüs der Arbeitsblatt-Menüleiste die entsprechende ID und Beschriftung Caption im Direktfenster des Visual Basic-Editors auf.

Hinweis

Um den Text sehen zu können, muss das Direktfenster eingeblendet sein. Sie finden die Einstellung im Menü ANSICHT | DIREKTFENSTER. Alternativ betätigen Sie die Tastenkombination `[Strg]+[G]`.

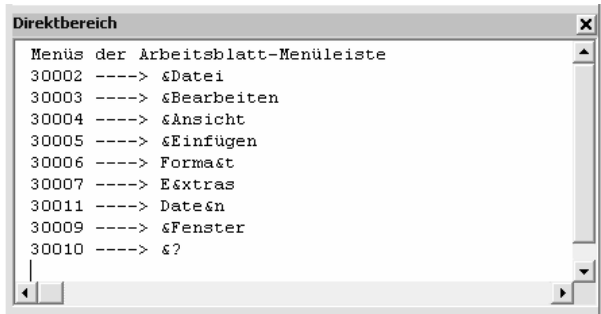


Abbildung 141: IDs und Beschriftung aller Menüs der Arbeitsblatt-Menüleiste

```
'=====
' Auf CD      Buchdaten\Beispiele\06_Befehlsleisten\
' Dateiname   06_03_EditControls.xls
' Modul       mdl_01_ControlsID
'=====

Sub ControlsID()
    Dim intI As Integer
    Dim cmdBar As CommandBar

    Set cmdBar = Application.CommandBars(1)

    Debug.Print "Menüs der Arbeitsplatz-Menüleiste"

    For intI = 1 To cmdBar.Controls.Count
        Debug.Print cmdBar.Controls(intI).ID & _
            " ----> " & _
            cmdBar.Controls(intI).Caption
    Next

End Sub
```

172 Ganzes Menü deaktivieren oder aktivieren

In folgendem Beispiel wird das Menü EXTRAS der ARBEITSPLATZ-Menüleiste deaktiviert bzw. aktiviert. Um ein anderes Menü zu bearbeiten, setzen Sie die entsprechende Beschriftung ein.

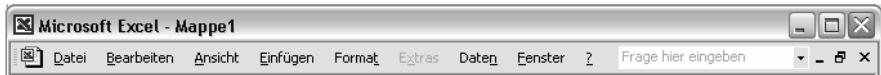


Abbildung 142: Deaktiviertes Extras-Menü

Wenn Sie Menüs oder Schaltflächen im VBA-Code über deren `Caption`-Eigenschaft (Beschriftung) ansprechen, muss die tatsächliche Aufschrift, so wie sie in Excel angezeigt wird, angegeben werden. Wird die Beschriftung geändert oder die Datei in einer anderssprachigen Excel-Version verwendet, verursacht die Prozedur eine Fehlermeldung. Hingegen bleibt das `ID`-Argument der `FindControl`-Methode für Menüs immer gleich. Ein Beispiel dazu finden Sie im nachfolgenden Rezept 173.

```
'=====
' Auf CD      Buchdaten\Beispiele\06_Befehlsleisten\
' Dateiname   06_03_EditControls.xls
' Modul       mdl_02_MenuExtras
'=====

Sub ExtrasDeaktiviere()
    Dim ctlBarControl As CommandBarControl

    Set ctlBarControl = Application.CommandBars(1) _
                        .Controls("Extras")
    ctlBarControl.Enabled = False
End Sub

Sub ExtrasAktiviere()
    Dim ctlBarControl As CommandBarControl

    Set ctlBarControl = Application.CommandBars(1) _
                        .Controls("Extras")
    ctlBarControl.Enabled = True
End Sub
```

173 Ganzes Menü über ID aus- und einblenden

Im vorherigen Beispiel wurde das Menü EXTRAS anhand der `Caption`-Eigenschaft deaktiviert. Dadurch kann es durch den Anwender nicht benutzt werden, ist aber in der Menüleiste ausgegraut sichtbar. Ebenso können Sie das Menü auch ausblenden. In folgender Prozedur wird in der `FindControl`-Methode das EXTRAS-Menü über seine ID angesprochen und aus- bzw. eingeblendet.

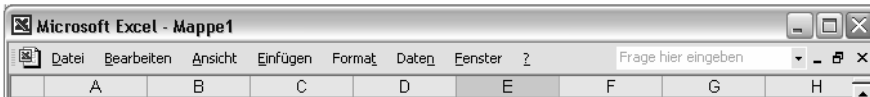


Abbildung 143: Ausgeblendetes Extras-Menü

```
'=====
' Auf CD      Buchdaten\Beispiele\06_Befehlsleisten\
' Dateiname   06_03_EditControls.xls
' Modul       mdl_02_MenuExtras
'=====
```

```
Sub ExtrasVisibleFalse()  
    Dim cmdBar As CommandBar  
  
    Set cmdBar = Application.CommandBars(1)  
    cmdBar.FindControl(ID:=30007).Visible = False  
End Sub  
  
Sub ExtrasVisibleTrue()  
    Dim cmdBar As CommandBar  
  
    Set cmdBar = Application.CommandBars(1)  
    cmdBar.FindControl(ID:=30007).Visible = True  
End Sub
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

174 Untermenü-Auflistung im Direktfenster

Am Beispiel des Menüs EXTRAS der ARBEITSBLATT-MENÜLEISTE erläutern wir den Zugriff auf verschachtelte Menüs und Untermenüs. Mit der Prozedur aus Rezept 171 erzeugen Sie eine Auflistung aller Schaltflächen `CommandBarControls` einer Befehlsleiste `Commandbar`-Objekt. Mit folgender Syntax setzen Sie einen Verweis auf das Steuerelement EXTRAS in der ARBEITSBLATT-MENÜLEISTE:

Befehl
leisten

Objekt

Dia-
gramm

```
Dim ctlBarControl As CommandBarControl  
Set ctlBarControl = Application.CommandBars(1).Controls("Extras")
```

Ereig-
nisse

Die Steuerelemente (Popupmenüs) der ARBEITSBLATT-MENÜLEISTE sind vom Typ `msoControl-
Popup` und stellen jeweils ein eigenständiges `CommandBar`-Objekt dar. Bei der Auflistung der Steuer-
elemente eines Menüs verweisen Sie direkt auf das jeweilige Menü. Dabei muss der englische
Name der Befehlsleiste verwendet werden. Die Syntax für den Verweis auf das Menü EXTRAS lautet
in dem Fall:

User-
Form

Web/
Mail

```
Dim cmdBar As CommandBar  
Set cmdBar = Application.CommandBars("Tools")
```

Extern

Hinweis

Im Gegensatz zu der sonst üblichen, sprachspezifischen Ansprache bei Menüs und Steuer-
elementen muss bei direkter Ansprache des Menüs als Befehlsleiste die englische Menü-
beschriftung angegeben werden.

Ge-
misch
tes

Specia

In der nachfolgenden Übersicht finden Sie die englischen Übersetzungen zu den deutschen
Menübeschriftungen der ARBEITSBLATT-MENÜLEISTE.

Deutsche Menübeschriftung	Englische Menübeschriftung
Datei	File
Bearbeiten	Edit
Ansicht	View

Tabelle 39: Englische Menübefehle

Deutsche Menübeschriftung	Englische Menübeschriftung
Einfügen	Insert
Format	Format
Extras	Tools
Daten	Data
Fenster	Window
?	Help

Tabelle 39: Englische Menübefehle (Forts.)

Setzen Sie anstelle des Menünamens TOOLS eine beliebige englische Menübeschriftung aus der Tabelle ein, um eine Auflistung der entsprechenden Untermenüs zu erhalten. Die ID-Nummierungen variieren zwischen ein- und fünfstelligen Zahlen. Um im Direktfenster eine übersichtliche Ansicht darzustellen, wird die ID über die `Format`-Funktion fünfstellig angezeigt.

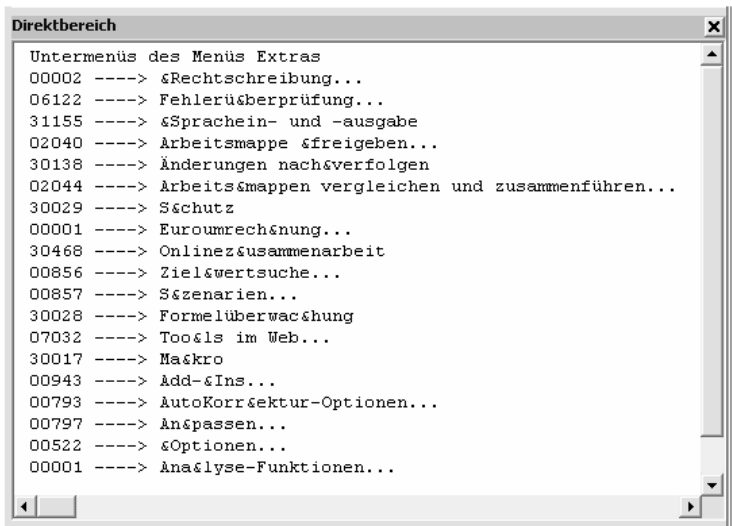


Abbildung 144: IDs und Beschriftung aller Untermenüs des Extras-Menüs

```
'=====
' Auf CD      Buchdaten\Beispiele\06_Befehlsleisten\
' Dateiname   06_03_EditControls.xls
' Modul       mdl_03_ControlsID2
'=====

Sub ControlsID2()
    Dim intI As Integer
    Dim cmdBar As CommandBar

    Set cmdBar = Application.CommandBars("Tools")

    Debug.Print "Untermenüs des Menüs Extras"
```



```
For intI = 1 To cmdBar.Controls.Count
    Debug.Print Format(cmdBar.Controls(intI).ID, "00000") & _
        " ----> " & _
        cmdBar.Controls(intI).Caption
Next
End Sub
```

175 Untermenüs deaktivieren und aktivieren

Um Untermenüs zu bearbeiten, können Sie die Reihenfolge der genauen Verschachtelung angeben. Folgende Prozedur deaktiviert das Untermenü SCHUTZ im Menü EXTRAS.

```
'=====
' Auf CD      Buchdaten\Beispiele\06_Befehlsleisten\
' Dateiname   06_03_EditControls.xls
' Modul       mdl_04_MenuProtect
'=====

Sub ProtectDeaktiviere()
    Application.CommandBars(1) _
        .Controls("Extras") _
        .Controls("Schutz").Enabled = False
End Sub
```

Am Beispiel des SCHUTZ-Menüs greifen wir nicht über die ARBEITSBLATT-MENÜLEISTE und das Steuerelement EXTRAS auf das Menü zu, sondern sprechen das Popupmenü EXTRAS direkt über seinen englischen Namen TOOLS an.

```
'=====
' Auf CD      Buchdaten\Beispiele\06_Befehlsleisten\
' Dateiname   06_03_EditControls.xls
' Modul       mdl_04_MenuProtect
'=====

Sub ProtectAktiviere()
    Application.CommandBars("Tools") _
        .Controls("Schutz").Enabled = True
End Sub
```

176 Untermenü des Untermenüs deaktivieren und aktivieren

Um eine Auflistung der Untermenüs SCHUTZ im Menü EXTRAS zu erhalten, setzen Sie die Prozedur aus Rezept 174 ein und ändern lediglich den Verweis:

```
Dim ctlBarControl As CommandBarControl
Set ctlBarControl = Application.CommandBars(1) _
    .Controls("Extras") _
    .Controls("Schutz")
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

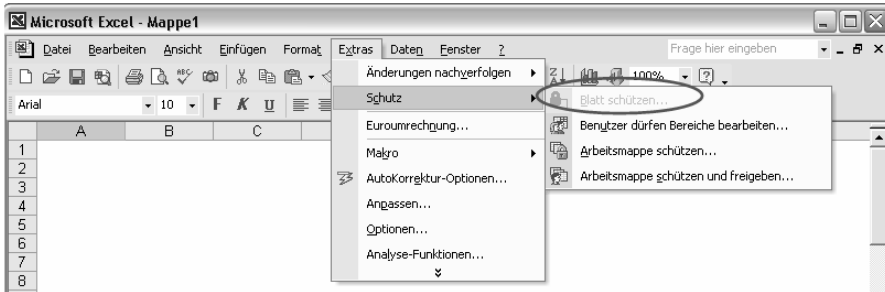


Abbildung 145: Verschachteltes Untermenü deaktivieren

Hinweis

Sie können eine Menü- oder eine Symbolleiste in der Liste der verfügbaren Menü- und Symbolleisten der Containeranwendung anhand des Namens oder der Indexnummer bestimmen. Um jedoch ein Menü, ein Kontextmenü oder ein Untermenü anzugeben (alle dargestellt durch `CommandBar`-Objekte), müssen Sie den Namen verwenden.

Um das Menü `BLATT SCHÜTZEN` im Menü `EXTRAS | SCHUTZ` zu deaktivieren bzw. zu aktivieren, verwenden Sie folgende Prozeduren.

```
'=====
' Auf CD      Buchdaten\Beispiele\06_Befehlsleisten\
' Dateiname   06_03_EditControls.xls
' Modul       mdl_06_SheetProtect
'=====
```

```
Sub SheetProtectDeaktiviere()
    Application.CommandBars(1) _
        .Controls("Extras") _
        .Controls("Schutz") _
        .Controls("Blatt schützen...").Enabled = False
End Sub
```

```
Sub SheetProtectAktiviere()
    Application.CommandBars(1) _
        .Controls("Extras") _
        .Controls("Schutz") _
        .Controls("Blatt schützen...").Enabled = True
End Sub
```

177 Befehlsleisten zurücksetzen

Die in der Beispieldatei vorgenommenen Änderungen an der `ARBEITSBLATT-MENÜLEISTE` und an verschiedenen Steuerelementen gelten für die gesamte Excel-Anwendung. Diese Konfiguration wird beim Schließen von Excel in der Befehlsleistendatei `*.xlb` gespeichert.

Mit der `Reset`-Methode setzen Sie eine integrierte Befehlsleiste auf die Standardkonfiguration der Steuerelemente zurück. Des Weiteren kann ein integriertes Befehlsleisten-Steuerelement auf dessen Standardfunktion und -symbol zurückgesetzt werden.

Der schnellste Weg, eine integrierte Befehlsleiste zurückzusetzen, führt über das Direktfenster. Geben Sie für die ARBEITSBLATT-MENÜLEISTE folgende Syntax im Direktfenster ein und beenden Sie Ihre Eingabe mit der -Taste.

```
CommandBars(1).Reset
```

Diesen Befehl können Sie auch in einer Prozedur verwenden. Folgende Prozedur setzt alle integrierten Befehlsleisten auf ihre Standardkonfiguration zurück. Über die `BuiltIn`-Eigenschaft wird ermittelt, ob es sich um eine integrierte Befehlsleiste handelt. Excel gibt eine Fehlermeldung aus, wenn Sie versuchen, eine benutzerdefinierte Befehlsleiste zurückzusetzen.

Achtung

Benutzerdefinierte Befehlsleisten können mit der `Reset`-Methode nicht zurückgesetzt werden.

```
'=====
' Auf CD      Buchdaten\Beispiele\06_Befehlsleisten\
' Dateiname  06_03_EditControls.xls
' Modul      mdl_07_ResetCommandBar
'=====

Sub ResetCommandBar()
    Dim cmdBar As CommandBar

    For Each cmdBar In Application.CommandBars
        If cmdBar.BuiltIn Then
            cmdBar.Reset
        End If
    Next cmdBar

End Sub
```

Hinweis

Durch das Zurücksetzen einer integrierten Befehlsleiste werden benutzerdefinierte Steuerelemente (ActiveX) entfernt und die integrierten Steuerelemente wiederhergestellt.

Um Ihre persönlichen Einstellungen an den Symbolleisten nicht zu beeinflussen, werden im `Workbook_Deactivate`-Ereignis der Beispieldatei alle Prozeduren zur Aktivierung der verwendeten Steuerelemente aufgerufen. Diese Ereignis-Prozedur wird beim Wechsel in eine andere Excel-Arbeitsmappe oder beim Schließen der Beispieldatei oder Excel-Anwendung automatisch ausgelöst.

```
'=====
' Auf CD      Buchdaten\Beispiele\06_Befehlsleisten\
' Dateiname  06_03_EditControls.xls
' Ereignis   DieseArbeitsmappe
'=====
```

Grundlagen

Allgemein

Datum und Zeit

Formeln

Steuerelemente

Befehlsleisten

Objekt

Diagramm

Ereignisse

User-Form

Web/Mail

Extern

Gemischtes

Spezial

```
Private Sub Workbook_Deactivate()
    ExtrasAktivate
    ExtrasVisibleTrue
    ProtectAktivate
    SheetProtectAktivate
End Sub
```

178 Menüleisten ein- und ausblenden

Wie eingangs bereits erwähnt, gibt es drei Typen von Befehlsleisten. Die Bearbeitung dieser Leisten ist sehr unterschiedlich, was bereits beim Ein- und Ausblenden der Leisten zu sehen ist.

Im ANPASSEN-Dialog können Sie neben den Symbolleisten auch die ARBEITSBLATT-Menüleiste deaktivieren. Entfernen Sie dazu den Haken vor der entsprechenden Leiste. Zunächst verschwindet die Menüleiste auch aus der Anwendung. Sobald Sie aber den ANPASSEN-Dialog schließen, wird die Menüleiste wieder angezeigt. Es ist also mit reinen Excel-Bordmitteln nicht möglich, eine Menüleiste dauerhaft auszublenden. Unsere Tests mit dem Makrorekorder haben ergeben, dass beim Deaktivieren von Befehlsleisten über den ANPASSEN-Dialog die `Visible`-Eigenschaft des `CommandBar`-Objektes auf `False` gesetzt wird. Hingegen wird ein Fehler ausgelöst, wenn man mit VBA die `Visible`-Eigenschaft einer Menüleiste auf `False` setzt.

Um eine Menüleiste auszublenden, kommen Sie an VBA nicht vorbei. Setzen Sie dazu die `Enabled`-Eigenschaft der Leiste auf `False`. Mit `True` wird die Menüleiste wieder angezeigt. Am Beispiel der Arbeitsblatt-Menüleiste lautet die Syntax:

```
Application.CommandBars("Worksheet Menu Bar").Enabled = False
```

```
Application.CommandBars("Worksheet Menu Bar").Enabled = True
```

Tipp

Um eine Befehlsleiste anzusprechen, können Sie auch die Indexnummer der Leiste angeben. Die Arbeitsblatt-Menüleiste hat den Index 1, somit verkürzt sich die Syntax auf:

```
Application.CommandBars(1).Enabled = False ' True für Einblenden
```

Um die aktive Menüleiste anzusprechen, verwenden Sie:

```
CommandBars.ActiveMenuBar.Enabled = False ' True für Einblenden
```

Eine Auflistung aller Befehlsleisten mit deren Indexnummern erhalten Sie mit der Prozedur aus dem Rezept 169, Befehlsleisten ermitteln.

Wenn ein Diagramm oder Diagrammblatt aktiviert ist, wird anstelle der Arbeitsblatt-Menüleiste die Diagrammmenüleiste angezeigt. Um ganz sicher zu gehen, dass keine integrierte Menüleiste angezeigt wird, müssen beide integrierten Menüleisten in Excel deaktiviert werden. Verwenden Sie eine `For...Each...Next`-Anweisung. In einer `If`-Anweisung wird der Leistentyp ermittelt und über die `BuiltIn`-Eigenschaft festgestellt, ob die Menüleiste integriert ist. Treffen beide Abfragen zu, wird die Leiste mit `cmb.Enabled = False` ausgeblendet. Um alle integrierten Menüleisten wieder zu aktivieren, setzen Sie die `Enabled`-Eigenschaft auf `True`.

```
' =====
' Auf CD      Buchdaten\Beispiele\06_Befehlsleisten\
' Dateiname   06_04_ShowCommandBars.xls
```

```
' Tabelle      MenuBar
' Modul        mdl_01_MenuBar
' =====

Sub DeaktiviereBultInMenuBars()
    Dim cmb As CommandBar

    For Each cmb In Application.CommandBars
        If cmb.Type = msoBarTypeMenuBar And _
            cmb.BuiltIn = True Then
            cmb.Enabled = False
        End If
    Next cmb
End Sub
```

In der folgenden Prozedur wird der englische Name der Menüleiste aus der Zelle G12 der Beispieldatei ausgelesen und ein Verweis auf die Variable `cmb` gesetzt. Der Anzeigestatus einer Menüleiste wird über die `Enabled`-Eigenschaft ermittelt. Über den Operator `Not` wird eine logische Negation, also eine Umkehrung des Anzeigezustands, bewirkt. Eine aktivierte Menüleiste wird deaktiviert und umgekehrt.

```
' =====
' Auf CD       Buchdaten\Beispiele\06_Befehlsleisten\
' Dateiname    06_04_ShowCommandBars.xls
' Tabelle      MenuBar
' Modul        mdl_01_MenuBar
' =====

Sub ChangeMenuBarStatus()
    Dim cmb As CommandBar

    Set cmb = Application.CommandBars(Range("G12").Value)

    With cmb
        If .Type = msoBarTypeMenuBar Then
            .Enabled = Not .Enabled ' Anzeigewechsel
        End If
    End With
End Sub
```

179 Symbolleisten ein- und ausblenden

Es gibt zwei Möglichkeiten, um Symbolleisten zu verwalten. Wie bei den Menüleisten werden Symbolleisten durch die `Enabled`-Eigenschaft deaktiviert bzw. aktiviert, was ebenfalls zum Ein- oder Ausblenden führen kann. Beachten Sie hierbei, dass die `Visible`-Eigenschaft letztendlich über die tatsächliche Anzeige einer Symbolleiste entscheidet.

Deaktivieren Sie zum Beispiel die `FORMAT`-Symbolleiste mit

```
Application.CommandBars("Formatting").Enabled = False
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

Unabhängig von der `Visible`-Eigenschaft wird die Symbolleiste ausgeblendet. Zusätzlich wird die `FORMAT`-Symbolleiste dadurch nicht mehr in der Auswahl der Befehlsleisten angezeigt. Das kann von Vorteil sein, wenn der Anwender eine Symbolleiste nicht über den Menüpunkt `ANSICHT | SYMBOLLEISTEN` oder im `KONTEXTMENÜ` einer `BEFEHLSLEISTE` anzeigen darf.

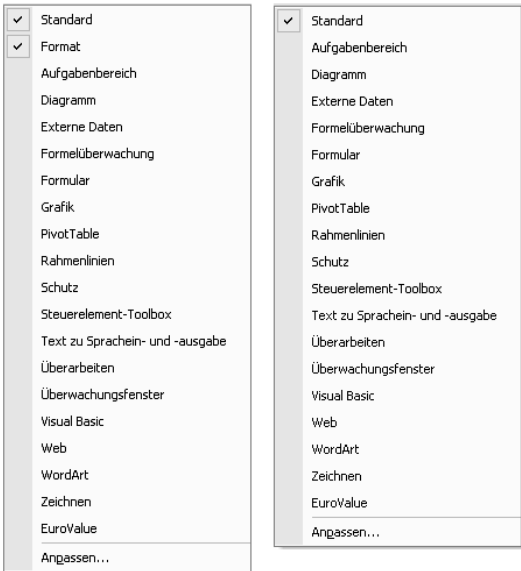


Abbildung 146: Deaktivierte Format-Symbolleiste

Hinweis Wenn die `Enabled`-Eigenschaft einer Symbolleiste auf `False` gesetzt wurde, kann sie weder über die Excel-Oberfläche noch in VBA mit der `Visible`-Eigenschaft zur Ansicht gebracht werden. Excel verursacht einen Fehler, wenn Sie versuchen, die `Visible`-Eigenschaft einer deaktivierten Symbolleiste zu ändern. Symbolleisten werden nur angezeigt, wenn sowohl die `Enabled`- als auch die `Visible`-Eigenschaft auf `True` eingestellt sind. Dabei ist zuerst die `Enabled`- und anschließend die `Visible`-Eigenschaft einzustellen.

In der folgenden Prozedur soll die im Tabellenblattes `NORMALBAR` in Zelle `G12` namentlich genannte Symbolleiste ein- bzw. ausgeblendet werden. Durch die Anweisung `With cmb` muss die Variable `cmb` in nachfolgenden Anweisungen nicht mehr aufgeführt werden. Eine Eigenschaft oder Methode mit vorangestelltem Punkt bezieht sich innerhalb des `With`-Blocks auf das genannte Objekt `cmb`. Über die `Type`-Eigenschaft wird ermittelt, ob es sich um eine Symbolleiste handelt. Des Weiteren wird mit der `Enabled`-Eigenschaft überprüft, ob die angesprochene Leiste aktiviert ist. Anschließend wird der Anzeigestatus, wie im vorherigen Beispiel, über die `Visible`-Eigenschaft in Verbindung mit dem `Not`-Operator gewechselt.

```
'=====
' Auf CD      Buchdaten\Beispiele\06_Befehlsleisten\
' Dateiname  06_04_ShowCommandBars.xls
' Tabelle    NormalBar
' Modul     mdl_02_NormalBar
'=====
```

```
Sub ChangeNormalBarStatus()  
    Dim cmb As CommandBar  
  
    Set cmb = Application.CommandBars(Range("G12").Value)  
  
    With cmb  
        If .Type = msoBarTypeNormal And _  
            .Enabled = True Then  
            .Visible = Not .Visible ' Anzeigewechsel  
        End If  
    End With  
  
End Sub
```

180 Kontextmenüs einblenden

Kontextmenüs sind Befehlsleisten vom Typ `msoBarPopup`. Sie werden zum Beispiel durch Drücken der rechten Maustaste an den aktuellen Mauszeigerkoordinaten angezeigt. In VBA zeigen Sie Kontextmenüs mit der `ShowPopup`-Methode an. Bei dieser Methode stehen Ihnen die optionalen Parameter `X` und `Y` für die Platzierung eines Kontextmenüs zur Verfügung. Die `X`-Koordinate gibt die vertikale Position, die `Y`-Koordinate die horizontale Position des Kontextmenüs an. Wird eines oder beide Argumente ausgelassen, wird die jeweilige aktuelle Mauszeigerkoordinate zur Positionierung verwendet.

In der folgenden Prozedur soll das Kontextmenü aus Zelle G12 des Tabellenblattes `POPUP` eingeblendet werden. Das Anzeigen eines Kontextmenüs kann von dem Objekt abhängig sein, das angeklickt wird. Wenn Sie zum Beispiel die Befehlsleiste `BEFEHL HINZUFÜGEN` anzeigen wollen, führt dies in unserer Beispieldatei zu einer Fehlermeldung. Um solche Fehlermeldungen abzufangen, verweisen wir mit der Sprungmarke `ERRORHANDLER` an das Ende der Prozedur. Dort erfolgt ein Meldungsfenster.

Das angegebene Kontextmenü wird mit der `ShowPopup`-Methode angezeigt. Die VBA-Prozedur wird für die Dauer der Kontextmenü-Anzeige angehalten. Sobald Sie mit der linken Maustaste auf eine freie Stelle im Tabellenblatt klicken, wird das Kontextmenü ausgeblendet und die laufende VBA-Prozedur weiter ausgeführt. Wenn Sie mit der linken Maustaste einen Eintrag aus dem Kontextmenü anwählen, wird der gewählte Befehl ausgeführt und anschließend das Kontextmenü ausgeblendet. Durch die im Code angegebenen Koordinaten wird das Kontextmenü etwa in Bildschirmmitte angezeigt. Sie können angezeigte Kontextmenüs nicht mit VBA ausblenden. Dazu bedarf es eines Mausclicks in der Anwendung oder auf eine beliebige Stelle der Bildschirmfläche. Einige Kontextmenüs können im Tabellenblatt nicht angezeigt werden. Ein eventuell auftretender Fehler würde durch eine Fehlerroutine abgefangen.

Hinweis

Wird die `Visible`-Eigenschaft auf ein deaktiviertes Kontextmenü angewandt, erfolgt im Gegensatz zu den Symbolleisten keine Fehlermeldung. Stellen Sie die `Visible`-Eigenschaft eines deaktivierten Kontextmenüs auf `True`, wird die Leiste lediglich nicht angezeigt.

'=====

' Auf CD Buchdaten\Beispiele\06_Befehlsleisten\

' Dateiname 06_04_ShowCommandBars.xls

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

```
' Tabelle      ShowPopUp
' Modul       mdl_03_PopUp
' =====

Sub MyShowPopup()
    Dim cmb As CommandBar

    On Error GoTo ERRORHANDLER
    Set cmb = Application.CommandBars(Range("G12").Value)

    If cmb.Type = msoBarTypePopup Then
        cmb.ShowPopup X:=400, Y:=300
    End If

    Exit Sub

ERRORHANDLER:
    MsgBox "Das Kontextmenü: " & _
        vbNewLine & _
        "      " & Range("F12").Value & "' " & _
        vbNewLine & _
        "kann nicht angezeigt werden.", _
        vbCritical

End Sub
```

181 Zelle-Kontextmenü deaktivieren

Über die Enabled-Eigenschaft können Sie den Zugriff auf Kontextmenüs unterbinden. In der nachfolgenden Übersicht finden Sie die gängigsten Kontextmenüs.

Kontextmenü	Beschreibung
Cell	Kontextmenü der Zellen
Column	Kontextmenü der Spaltenköpfe
Row	Kontextmenü der Zeilenköpfe
Ply	Kontextmenü der Blattregisterkarten
Workbook Tabs	Kontextmenü der Arbeitsmappen-Registerkarte (Auflistung aller Tabellen- und Diagrammblätter der aktiven Arbeitsmappe)
Toolbar List	Kontextmenü der Symbolleistenliste

Tabelle 40: Die häufigsten Kontextmenüs

Tipp

Eine Übersicht aller Kontextmenüs finden Sie in der Beispieldatei aus Rezept 6.17.

Folgende Prozeduren deaktivieren bzw. aktivieren das Kontextmenü der Zelle.


```
'=====
' Auf CD      Buchdaten\Beispiele\06_Befehlsleisten\
' Dateiname   06_04_ShowCommandBars.xls
' Tabelle     DeactivatePopUp
' Modul       mdl_03_PopUp
'=====

Sub DisableCellCmb()
    Application.CommandBars("Cell").Enabled = False
End Sub

Sub EnableCellCmb()
    Application.CommandBars("Cell").Enabled = True
End Sub
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

182 Alle Befehlsleisten ein- und ausblenden

Sie können in Excel über eine For Each...Next-Anweisung alle Befehlsleisten ausblenden, indem Sie deren Enabled-Eigenschaft auf False setzen.

Befehl
leisten

Objekt

```
'=====
' Auf CD      Buchdaten\Beispiele\06_Befehlsleisten\
' Dateiname   06_04_ShowCommandBars.xls
' Tabelle     CommandBars
' Modul       mdl_05_DisabledCommandBars
'=====

Sub DisabledCommandBars()
    Dim cmb As CommandBar
    ' Alle Leisten deaktivieren
    For Each cmb In Application.CommandBars
        cmb.Enabled = False
    Next
End Sub
```

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Excel merkt sich die jeweilige Visible-Eigenschaft der einzelnen Leisten. Wenn Sie über die Schleife alle Leisten mit Enabled = True wieder aktivieren, zeigt Excel nur die Leisten an, welche vor der Deaktivierung sichtbar waren.

Ge-
misch
tes

Specia

```
'=====
' Auf CD      Buchdaten\Beispiele\06_Befehlsleisten\
' Dateiname   06_04_ShowCommandBars.xls
' Tabelle     CommandBars
' Modul       mdl_05_DisabledCommandBars
'=====

Sub EnabledCommandBars()
    Dim cmb As CommandBar
    ' Alle Leisten aktivieren
    For Each cmb In Application.CommandBars
        cmb.Enabled = True
    Next
End Sub
```

183 Einzelne Symbolleistentypen ein- und ausblenden

Um alle Befehlsleisten oder nur jeweils alle Menüleisten, Symbolleisten oder Kontextmenüs zu deaktivieren bzw. zu aktivieren, benötigen Sie normalerweise acht einzelne Prozeduren. Diese würden sich jeweils nur durch die Angabe des Leistentyps und den booleschen Wert `True` oder `False` zur Aktivierung respektive Deaktivierung unterscheiden.

In diesem Beispiel stellen wir Ihnen eine Prozedur vor, die alle acht vorgenannten Unterscheidungen berücksichtigt, ohne dass jede davon einzeln angesprochen werden muss. Es wird eine Sub-Anweisung verwendet, um den Prozedurnamen `ChangeBarStatus`, die Variablenliste mit den Argumenten `blnEnabled` und `varType` und den Prozedurrumpf der Sub-Anweisung zu definieren. Die Prozedur kann nicht direkt aufgerufen werden, sondern muss aus einer anderen Prozedur mit Angabe der Argumente aufgerufen werden.

```
'=====
' Auf CD      Buchdaten\Beispiele\06_Befehlsleisten\
' Dateiname   06_04_ShowCommandBars.xls
' Tabelle     MenuBar
' Modul       mdl_06_ChangeBarStatus
'=====

Sub ChangeBarStatus(blnEnabled As Boolean, _
                    Optional varType As Variant = "")

    Dim cmb As CommandBar

    Application.ScreenUpdating = False

    For Each cmb In Application.CommandBars
        With cmb
            If varType = "" Then
                ' alle Leisten deaktivieren/aktivieren
                .Enabled = blnEnabled
            Else
                If .Type = varType Then
                    ' alle Leisten des übergebenen Typs deaktivieren/aktivieren
                    .Enabled = blnEnabled
                End If
            End If
        End With
    Next

    Application.ScreenUpdating = True
End Sub
```

Die Prozedur bietet folgende Aufrufmöglichkeiten.

Ausführung	blnEnabled	varType
Alle Befehlsleisten deaktivieren	False	"" (optional)
Alle Befehlsleisten aktivieren	True	"" (optional)
Symbolleisten deaktivieren	False	0

Tabelle 41: Parameter der `ChangeBarStatus`-Prozedur

Ausführung	blnEnabled	varType
Symbolleisten aktivieren	True	0
Menüleisten deaktivieren	False	1
Menüleisten aktivieren	True	1
Kontextmenüs deaktivieren	False	2
Kontextmenüs aktivieren	True	2

Tabelle 41: Parameter der ChangeBarStatus-Prozedur (Forts.)

In der Beispieldatei, welche Sie auf der beiliegenden CD finden, sind in verschiedenen Tabellenblättern Schaltflächen zum Aufruf der Prozeduren hinterlegt.

Das folgende Beispiel steht stellvertretend für die einzelnen Leistentypen. Geben Sie für die anderen Leistentypen die in Tabelle 41 angegebenen Argumente zum Aufruf der Prozedur `ChangeBarStatus` an.

```
'=====
' Auf CD      Buchdaten\Beispiele\06_Befehlsleisten\
' Dateiname  06_04_ShowCommandBars.xls
' Tabelle    MenuBar
' Modul      mdl_01_MenuBar
'=====

Sub DeaktiviereMenuBar()
    ' Alle Menüleisten deaktivieren
    ChangeBarStatus False, 1
End Sub

Sub AktiviereMenuBar()
    ' Alle Menüleisten aktivieren
    ChangeBarStatus True, 1
End Sub
```

Um alle Befehlsleisten zu deaktivieren oder zu aktivieren, wird eine leere Zeichenkette "" als Argument für den Parameter `varType` verwendet. Sie können beim Aufruf der Prozedur für alle Befehlsleisten auch auf die Angabe des zweiten Parameters `varType` verzichten. Dieser wurde in der Sub-Anweisung mit dem Schlüsselwort `Optional` deklariert und enthält bereits den Standardwert "".

```
'=====
' Auf CD      Buchdaten\Beispiele\06_Befehlsleisten\
' Dateiname  06_04_ShowCommandBars.xls
' Tabelle    MenuBar
' Modul      mdl_04_CommandBars
'=====

Sub DeaktiviereCommandBars()
    ' Alle Befehlsleisten deaktivieren
    ChangeBarStatus False
```

Grundlagen

Allgemein

Datum
Zeit

tungen

Steuer
elemente

Befehls
leisten

Objekt

Diagramm

Ereignisse

User-
Form

Web/
Mail

Extern

Gemischtes

Spezial

```

        ' oder
        ' ChangeBarStatus False, ""
    End Sub

Sub AktiviereCommandBars()
    ' Alle Befehlsleisten aktivieren
    ChangeBarStatus True

    ' oder
    ' ChangeBarStatus True, ""
End Sub

```

Damit in anderen Excel-Arbeitsmappen und nach Beendigung der Beispieldatei wieder alle Symbolleisten aktiviert werden, wird im Workbook_Deactivate-Ereignis der Beispielmappe die Prozedur `ChangeBarStatus` aufgerufen. Dabei wird das Argument `True` an den Parameter `blnEnabled` übergeben. Damit alle drei Leistentypen aktiviert werden, wird der optionale Parameter `varType` nicht angegeben.

```

' =====
' Auf CD      Buchdaten\Beispiele\06_Befehlsleisten\
' Dateiname   06_04_ShowCommandBars.xls
' Ereignis    DieseArbeitsmappe
' =====

Private Sub Workbook_Deactivate()
    ChangeBarStatus True
End Sub

```

184 Einzelne Symbolleisten vor Änderungen schützen

Über die `Protection`-Eigenschaft des `CommandBar`-Objektes werden Verfahren festgelegt, wie die angegebene Befehlsleiste vor Anpassungen durch den Benutzer geschützt werden kann. Es stehen folgende Konstanten zur Verfügung:

- ▶ `msoBarNoChangeDock`
- ▶ `msoBarNoChangeVisible`
- ▶ `msoBarNoCustomize`
- ▶ `msoBarNoHorizontalDock`
- ▶ `msoBarNoMove`
- ▶ `msoBarNoProtection`
- ▶ `msoBarNoResize`
- ▶ `msoBarNoVerticalDock`

Sie können einer Befehlsleiste eines der Verfahren oder eine Kombination aus mehreren Konstanten zuweisen. Die folgende Prozedur verhindert durch die `msoBarNoCustomize`-Konstante, dass Benutzer Schaltflächen auf der STANDARD-Symbolleiste hinzufügen oder entfernen können. Durch die `msoBarNoMove`-Konstante kann die Symbolleiste nicht verschoben werden.

```
' =====
' Auf CD      Buchdaten\Beispiele\06_Befehlsleisten\
' Dateiname  06_05_a_ProtectCommandBars.xls
' Modul      mdl_01_ProtectCommandBar
' =====

Sub ProtectCommandBar()
    Dim cmdBar As CommandBar

    Set cmdBar = CommandBars("Standard")
    cmdBar.Protection = msoBarNoCustomize + msoBarNoMove
End Sub
```

Um alle eingestellten Verfahren wieder zu deaktivieren, verwenden Sie die `msoBarNoProtection`-Konstante.

```
' =====
' Auf CD      Buchdaten\Beispiele\06_Befehlsleisten\
' Dateiname  06_05_a_ProtectCommandBars.xls
' Modul      mdl_01_ProtectCommandBar
' =====

Sub UnProtectCommandBar()
    Dim cmdBar As CommandBar

    Set cmdBar = CommandBars("Standard")
    cmdBar.Protection = msoBarNoProtection
End Sub
```

Beim Wechsel in andere Excel-Arbeitsmappen und beim Schließen der Beispielmappe wird im `Workbook_Deactivate`-Ereignis die Prozedur `UnProtectCommandBar` zum Aufheben der Schutzfunktionen aufgerufen.

```
' =====
' Auf CD      Buchdaten\Beispiele\06_Befehlsleisten\
' Dateiname  06_05_a_ProtectCommandBars.xls
' Ereignis    DieseArbeitsmappe
' =====

Private Sub Workbook_Deactivate()
    UnProtectCommandBar
End Sub
```

185 Befehlsleisten vor Änderungen schützen in XL97 und XL2000

Mit Hilfe des Dialogfeldes ANPASSEN ist es möglich, sowohl integrierte als auch benutzerdefinierte Befehlsleisten zu ändern oder zu deaktivieren bzw. zu aktivieren. Benutzerdefinierte Leisten können über den Dialog auch gelöscht werden. Ebenso können Sie über diesen Dialog Schaltflächen

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

von Symbolleisten entfernen oder hinzufügen. Das Dialogfeld ANPASSEN kann über vier Zugriffspunkte erreicht werden:

- ▶ Menü ANSICHT | SYMBOLLEISTEN | ANPASSEN
- ▶ Menü EXTRAS | ANPASSEN
- ▶ Rechtsklick auf eine freie Stelle einer Symbolleiste
- ▶ Doppelklick auf eine freie Stelle einer Symbolleiste

Sie können verhindern, dass Benutzer Ihrer Anwendung Änderungen an den Befehlsleisten vornehmen. Bis einschließlich Excel 2000 müssen Sie dafür die entsprechenden Menüpunkte schützen, indem Sie die `Enabled`-Eigenschaft der entsprechenden Schaltflächen auf `False` setzen. Vergessen Sie nicht, auch die Menüpunkte in der Diagrammenüleiste zu deaktivieren. Um den Zugriffspunkt über das Kontextmenü der Befehlsleisten auszuschalten, setzen Sie die `Enabled`-Eigenschaft der Symbolleiste `SYMBOLLEISTENLISTE` auf `False`. Diese Symbolleiste ist nicht in der `CommandBars`-Auflistung enthalten. Um sie anzusprechen, muss deren englischer Name `Toolbar List` verwendet werden.

```
Application.CommandBars("Toolbar List").Enabled = False
```

Mit einem Doppelklick auf eine freie Stelle in einer Symbolleiste wird dennoch der ANPASSEN-Dialog angezeigt. Um die normale Funktionalität des Doppelklicks zu deaktivieren, wird der `OnDoubleClick`-Eigenschaft die Prozedur `NoAction` zugewiesen. Diese führt keine Befehle aus.

Um die ANPASSEN-Funktion der Symbolleisten wieder zu aktivieren, müssen alle deaktivierten Menüpunkte und Eigenschaften wieder aktiviert werden. Setzen Sie dazu die jeweiligen `Enabled`-Eigenschaften auf `True`. Die Funktionalität des Doppelklicks aktivieren Sie wieder, indem Sie der Eigenschaft `OnDoubleClick` einen Leerstring zuweisen.

Anstelle von zwei Prozeduren, welche sich lediglich durch den booleschen Wert `False` oder `True` unterscheiden, verwenden wir eine Prozedur mit dem Parameter `blnProtect`. Dieser Parameter ist vom Datentyp `Boolean` und kann die Werte `True` oder `False` enthalten.

Hinweis

Die Prozeduren aus diesem Beispiel sind in den Excel-Versionen von XL97 bis XL2003 einsetzbar. Ab XL2002 steht zudem eine neue Eigenschaft zum Schutz der Symbolleisten zur Verfügung. Diese stellen wir Ihnen im nachfolgenden Rezept 186 vor.

```
' =====
' Auf CD      Buchdaten\Beispiele\06_Befehlsleisten\
' Dateiname   06_05_ProtectCommandBars_97_2000.xls
' Modul       mdl_01_AllowCustomization
' =====
```

```
Sub AllowCustomization(blnProtect As Boolean)
    Dim cmb As CommandBar

    With Application.CommandBars("Worksheet Menu Bar")
        .Controls("Ansicht").Controls("Symbolleisten") _
            .Enabled = blnProtect
    End With
End Sub
```

```
.Controls("Extras").Controls("Anpassen...") _
.Enabled = blnProtect
End With

Application.CommandBars("Toolbar List").Enabled = blnProtect

If blnProtect = True Then
    Application.OnDoubleClick = ""
Else
    Application.OnDoubleClick = "NoAction"
End If
End Sub

'=====
' Auf CD      Buchdaten\Beispiele\06_Befehlsleisten\
' Dateiname   06_05_ProtectCommandBars_97_2000.xls
' Modul       mdl_01_AllowCustomization
'=====

' Prozedur wird beim Doppelklick ausgeführt
Private Sub NoAction()
End Sub
```

Zum Schutz des ANPASSEN-Dialoges wird beim Aufruf der Prozedur AllowCustomization das Argument False übergeben. Der Parameter blnProtect enthält somit den Wert False und setzt ihn jeweils in der Prozedur ein. Die Prozedur wird im Workbook_Activate-Ereignis hinterlegt. Dadurch wird beim Öffnen der Datei und beim Wechsel von einer anderen Arbeitsmappe zu der Beispieldatei die ANPASSEN-Funktion nur in dieser Beispieldatei deaktiviert.

```
'=====
' Auf CD      Buchdaten\Beispiele\06_Befehlsleisten\
' Dateiname   06_05_ProtectCommandBars_97_2000.xls
' Ereignis    DieseArbeitsmappe
'=====

Private Sub Workbook_Activate()
    ' Anpassen-Funktion deaktivieren
    AllowCustomization False
End Sub
```

186 Befehlsleisten-Schutz aufheben

Im Workbook_Deactivate-Ereignis übergeben Sie das Argument True an den Parameter blnProtect, um die ANPASSEN-Funktion beim Wechsel zu einer anderen Arbeitsmappe oder beim Schließen der Beispieldatei zu aktivieren.

```
'=====
' Auf CD      Buchdaten\Beispiele\06_Befehlsleisten\
' Dateiname   06_05_ProtectCommandBars_97_2000.xls
' Ereignis    DieseArbeitsmappe
'=====
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

```
Private Sub Workbook_Deactivate()
    ' Anpassen-Funktion aktivieren
    AllowCustomization True
End Sub
```

187 Befehlsleisten-Schutz wechseln

Zum Umschalten der ANPASSEN-Funktion haben wir auf dem Tabellenblatt der Beispieldatei eine UMSCHALTFLÄCHE aus der STEUERELEMENTE-TOOLBOX aufgezogen. Klicken Sie auf die Schaltfläche, um das `ToggleButton1_Click`-Ereignis im Codemodul der Tabelle auszulösen und damit den `Enabled`-Wert der Symbolleisten-Anpassung zu wechseln. Beim Aufruf der Prozedur `AllowCustomization` wird als Argument für den booleschen Parameter `blnProtect` die `Value`-Eigenschaft des `ToggleButton1` an die aufgerufene Prozedur übergeben. Dementsprechend wird der Wert `True` oder `False` übergeben. Die Beschriftung der Umschaltfläche ergibt sich in der `IIf`-Funktion nach dem Wert der Umschaltfläche. Anschließend wird die aktive Zelle markiert, um den Fokus von der Umschaltfläche zu nehmen.

```
' =====
' Auf CD      Buchdaten\Beispiele\06_Befehlsleisten\
' Dateiname   06_05_ProtectCommandBars_97_2000.xls
' Ereignis    Tabelle1 (XL97_2000)
' =====

Private Sub ToggleButton1_Click()
    ' Prozeduraufruf mit Argumentenübergabe
    ' True zum Deaktivieren des Schutzes
    ' False zum Aktivieren des Schutzes

    AllowCustomization ToggleButton1.Value

    With ToggleButton1
        .Caption = ">>Symbolleisten Anpassen<< " & _
            IIf(.Value, "deaktivieren", "aktivieren")
    End With

    ActiveCell.Select
End Sub
```

188 Befehlsleisten vor Änderungen schützen ab XL2002

In Excel 2002 ist die `DisableCustomize`-Eigenschaft neu hinzugekommen. Wenn Sie das Anpassen von Symbolleisten deaktivieren wollen, setzen Sie die Eigenschaft auf `True`. Durch diese Eigenschaft werden alle vier in Rezept 184 genannten Zugriffspunkte zur Anpassung von Symbolleisten deaktiviert. Mit `False` aktivieren Sie die Anpassung wieder.

Diese Eigenschaft deaktiviert bei einem Rechtsklick auf eine freie Stelle einer Symbolleiste und im Menüpunkt **ANSICHT | SYMBOLLEISTEN** nicht die gesamte **SYMBOLLEISTENLISTE**, sondern entfernt lediglich den Menüpunkt **ANPASSEN** aus der Symbolleiste. Der Menüpunkt **EXTRAS | ANPASSEN** wird ausgegraut. Der Doppelklick auf eine freie Fläche einer Symbolleiste wird ebenfalls deaktiviert.

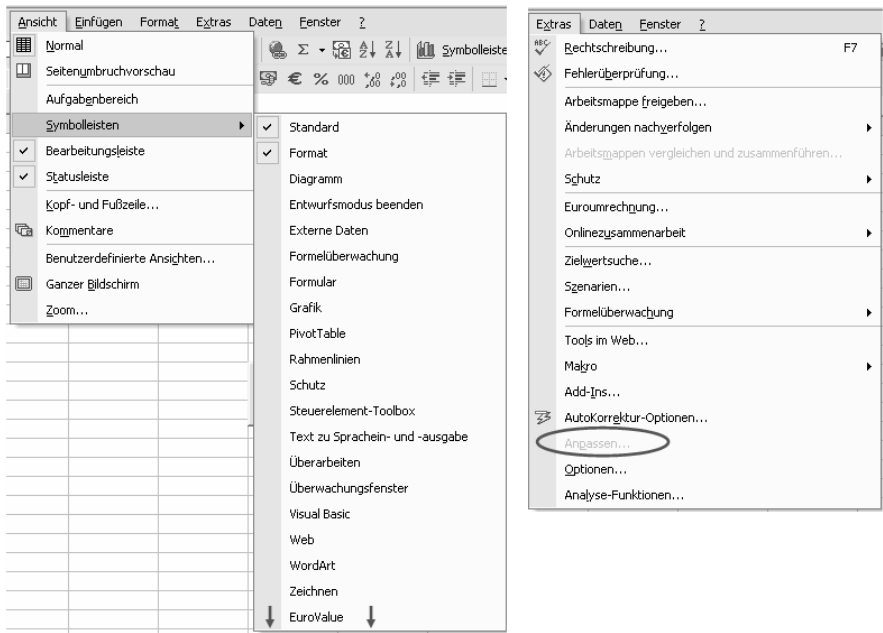


Abbildung 147: Entfernte und deaktivierte Anpassen-Funktion

In unserer Beispieldatei wird die Anpassung beim Öffnen und Schließen der Arbeitsmappe sowie beim Wechsel von und zu anderen Excel-Dateien über die Ereignisse `Workbook_Activate` und `Workbook_Deactivate` gesteuert. Dadurch wird die Anpassung der Symbolleisten nur in der Beispieldatei verändert.

Achtung

Diese Beispieldatei ist ab Excel 2002 lauffähig. Vorherige Excel-Versionen unterstützen die `DisableCustomize`-Eigenschaft nicht. Um das Anpassen von Symbolleisten in früheren Excel-Versionen zu steuern, verwenden Sie die Prozeduren aus den Rezepten 184 bis 187.

```
'=====
' Auf CD      Buchdaten\Beispiele\06_Befehlsleisten\
' Dateiname   06_06_ProtectCommandBars_2002_up.xls
' Ereignis    DieseArbeitsmappe
'=====

Private Sub Workbook_Activate()
    ' Anpassen von Symbolleisten deaktivieren
    Application.CommandBars.DisableCustomize = True
End Sub

Private Sub Workbook_Deactivate()
    ' Anpassen von Symbolleisten aktivieren
    Application.CommandBars.DisableCustomize = False
End Sub
```

Grundlagen
Allgemein
Datum und Zeit
Diagramme
Steuerungselemente
Befehlsleisten
Objekt
Diagramm
Ereignisse
User-Form
Web/Mail
Extern
Gemischtes
Spezial

Zum Ausprobieren können Sie im Tabellenblatt über eine Umschaltfläche `ToggleButton` aus der Steuerelemente-Toolbox-Symbolleiste den ANPASSEN-Status wechseln.

```
'=====
' Auf CD      Buchdaten\Beispiele\06_Befehlsleisten\
' Dateiname   06_06_ProtectCommandBars_2002_up.xls
' Ereignis    Tabelle1 (XL2002_up)
'=====

Private Sub ToggleButton1_Click()
' DisableCustomize-Eigenschaft umschalten
With Application.CommandBars
    .DisableCustomize = Not .DisableCustomize
End With

' Beschriftung der Umschaltfläche
With ToggleButton1
    .Caption = "Symbolleisten Anpassen " & _
        IIf(.Value, "aktivieren", "deaktivieren")
End With
ActiveCell.Select
End Sub
```

189 Fehlerrountinen vor Erstellung von Befehlsleisten

Bevor Sie eine Befehlsleiste erstellen, vergewissern Sie sich, dass eine Leiste mit diesem Namen nicht bereits existiert. Excel verursacht eine Fehlermeldung, wenn Sie eine neue Befehlsleiste erstellen, welche in der Anwendung bereits besteht. Dabei ist es gleichgültig, ob diese angezeigt wird. Diese Fehlermeldung können Sie zum Beispiel über die vorangestellte Fehleroutine `On Error Resume Next` oder über eine Sprungmarke `On Error Goto ERRORHANDLER` abfangen. Des Weiteren können Sie, wie in folgender Prozedur, über eine `For Each...Next`-Anweisung alle Leisten durchlaufen und anhand der Befehlsleistennamen überprüfen, ob die zu erstellende Leiste bereits vorhanden ist.

Hinweis

Die Prozedur zum Überprüfen bestehender Befehlsleisten ist für alle drei Leistentypen anwendbar.

```
'=====
' Auf CD      Buchdaten\Beispiele\06_Befehlsleisten\
' Dateiname   06_07_AddCommandBars.xls
' Modul       mdl_01_CheckCommandBar
'=====

Sub CheckCommandBars()
    Dim cmb As CommandBar

    For Each cmb In Application.CommandBars
        If cmb.Name = "MyBar" Then
            MsgBox "Die Leiste 'MyBar' existiert."
```

```
Exit Sub
End If
Next cmb

MsgBox "Es existiert keine Leiste 'MyBar'."
End Sub
```

190 Löschen von Befehlsleisten

In unseren Beispielen verwenden wir eine weitere Möglichkeit, einen Fehler abzufangen, damit das Hinzufügen einer bereits vorhandenen Befehlsleiste nicht zu einer Fehlermeldung führt. Die gewünschte Leiste wird vorab gelöscht.

Achtung

Es ist nur möglich, *benutzerdefinierte* Befehlsleisten zu löschen. *Integrierte* Befehlsleisten können lediglich ausgeblendet oder deaktiviert werden.

Für den Fall, dass die genannte Leiste nicht existiert, muss eine Fehlerroutine programmiert werden. Die angewandte `On Error Resume Next`-Anweisung legt fest, dass das Programm die Ausführung beim Auftreten eines Laufzeitfehlers mit der Anweisung fortsetzt, die auf die Fehler verursachende Anweisung folgt. Die `On Error GoTo 0`-Anweisung deaktiviert alle aktivierten Fehlerbehandlungsroutinen in der aktuellen Prozedur. Dadurch werden auftretende Fehler wieder angezeigt. Diese Prozedur beinhaltet in der Aufrufzeile den Parameter `strBar` und kann deshalb nicht direkt aufgerufen werden. Sie muss in einer anderen Prozedur mit Angabe des Namens der zu löschenden Befehlsleiste aufgerufen werden. Anhand dieser Prozedur können alle drei Befehlsleistentypen aus der Anwendung gelöscht werden.

```
'=====
' Auf CD      Buchdaten\Beispiele\06_Befehlsleisten\
' Dateiname  06_07_AddCommandBars.xls
' Modul      mdl_02_DeleteCommandBar
'=====

Sub DeleteCommandBar(strBar As String)
    On Error Resume Next
    Application.CommandBars(strBar).Delete
    On Error GoTo 0
End Sub
```

191 Erstellen einer Menüleiste

Menüleisten verhalten sich beinahe so wie normale Symbolleisten. Sie können nach Wunsch ein- und ausgeblendet oder positioniert werden und beliebige Steuerelemente enthalten. Es kann allerdings immer nur eine Menüleiste angezeigt werden. Wird eine Menüleiste an einem Seitenrand positioniert, kann neben dieser keine weitere Symbolleiste platziert werden, da sie die gesamte Breite der Anwendung in Anspruch nimmt.

Hinweis

Wenn Sie eine benutzerdefinierte Menüleiste mit `Visible = True` anzeigen, wird die aktuelle Menüleiste automatisch durch diese ersetzt.

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

Sie erkennen eine Menüleiste zum einen an dem grünen Excel-Icon am linken Seitenrand und zum anderen an den drei Schaltflächen MINIMIEREN, MAXIMIEREN und SCHLIESSEN der Excel-Anwendung. Ab Excel 2002 ist das FRAGE-EINGABEFELD in jeder Menüleiste integriert. Abbildung 148 zeigt eine benutzerdefinierte Menüleiste ohne hinzugefügte Schaltflächen und ohne weitere Symbolleisten.



Abbildung 148: Benutzerdefinierte Menüleiste ohne Schaltflächen

Bevor in unserer Prozedur eine benutzerdefinierte Menüleiste erstellt wird, erfolgt vorab der Aufruf der DeleteCommandBar-Prozedur zum Löschen einer eventuell vorhandenen Leiste gleichen Namens. In einer Set-Anweisung erfolgt ein Verweis auf die mit der Add-Methode hinzugefügte Befehlsleiste. Die Add-Methode erstellt eine neue Befehlsleiste und fügt sie zur Auflistung der Befehlsleisten hinzu. Wenn Sie eine neue Befehlsleiste erstellen, können Sie optional vier Argumente angeben. Diese sind bei allen drei Befehlsleistentypen (Menüleiste, Symbolleiste oder Kontextmenü) anwendbar.

Parameter	Beschreibung
Name	Der Name der neuen Befehlsleiste. Wird dieses Argument weggelassen, wird der Befehlsleiste ein Standardname mit fortlaufender Nummerierung zugewiesen (z.B. Benutzerdefiniert 1).
Position	Die Position der neuen Befehlsleiste. Bestimmung als Kontextmenü.
MenuBar	True, wenn die aktive Menüleiste durch die neue Befehlsleiste ersetzt werden soll. Der Standardwert ist False.
Temporary	True, um die neue Befehlsleiste als temporäre Befehlsleiste zu erstellen. Temporäre Befehlsleisten werden beim Schließen der Containeranwendung gelöscht. Der Standardwert ist False.

Tabelle 42: Mögliche Argumente der Add-Methode für das CommandBars-Objekt

Geben Sie beim Erstellen einer Menüleiste keine Position an, dann wird standardmäßig der obere Seitenrand (unter der Excel-Titelleiste) angenommen. Eine Befehlsleiste wird nur dann als Menüleiste erstellt, wenn die MenuBar-Eigenschaft auf True gesetzt wird. Wenn Sie eine benutzerdefinierte Menüleiste über die Visible-Eigenschaft anzeigen, wird die aktive Menüleiste ausgeblendet. Dieser Wechsel der Menüleisten ist am Bildschirm sichtbar und verursacht ein Flackern der Ansicht. Dementsprechend wird die Bildschirmaktualisierung über die ScreenUpdating-Eigenschaft gesteuert.

```
' =====
' Auf CD      Buchdaten\Beispiele\06_Befehlsleisten\
' Dateiname   06_07_AddCommandBars.xls
' Modul       mdl_03_AddMenuBar
' =====

Sub AddMenuBar()
    Dim cmdMenuBar As CommandBar

    Application.ScreenUpdating = False

    On Error Resume Next
    Application.CommandBars("MyMenuBar").Delete
    On Error GoTo 0

    Set cmdMenuBar = Application.CommandBars.Add( _
        Name:="MyMenuBar", _
        Position:=msoBarTop, _
        MenuBar:=True, _
        Temporary:=True)

    ' neue Menüleiste anzeigen
    cmdMenuBar.Visible = True
    Application.ScreenUpdating = True
End Sub
```

Hinweis Beim Schließen der Beispielmappe müssen die erstellten Befehlsleisten nicht explizit gelöscht werden, da sie temporär mit `Temporary:=True` erstellt wurden und beim Beenden der Arbeitsmappe automatisch gelöscht werden.

192 Wechseln und Löschen von Menüleisten

Sie können eine benutzerdefinierte Menüleiste durch folgende Anweisungen wieder gegen die Standard-Menüleiste austauschen:

- ▶ `cmdMenuBar.Visible = False` ausblenden
- ▶ `cmdMenuBar.Delete` löschen.

Wenn die `Enabled`-Eigenschaft der ARBEITSBLATT-MENÜLEISTE auf `True` eingestellt ist, wird diese anschließend angezeigt.

Hinweis Wenn Sie eine benutzerdefinierte Menüleiste mit `.Enabled = False` deaktivieren, wird diese ausgeblendet und anschließend keine andere, auch nicht die ARBEITSBLATT-MENÜLEISTE, automatisch angezeigt.

Um die Menüleiste `MYMENUBAR` zu löschen, setzen Sie den Namen der zu löschenden Leiste in der `Delete`-Methode ein.

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

```
'=====
' Auf CD      Buchdaten\Beispiele\06_Befehlsleisten\
' Dateiname  06_07_AddCommandBars.xls
' Modul      mdl_03_AddMenubar
'=====

Sub DeleteMenuBar()
    Application.ScreenUpdating = False

    On Error Resume Next
    Application.CommandBars("MyMenuBar").Delete
    On Error GoTo 0

    Application.ScreenUpdating = True
End Sub
```

193 Symbolleisten erstellen und verankern

Beim Erstellen von Symbol- und Menüleisten über die `Add`-Methode können Sie über das optionale `Position`-Argument die Position der Leiste festlegen.

Konstante	Beschreibung
<code>msoBarLeft</code>	Befehlsleiste wird am linken Seitenrand verankert.
<code>msoBarTop</code>	Befehlsleiste wird am oberen Seitenrand verankert.
<code>msoBarRight</code>	Befehlsleiste wird am rechten Seitenrand verankert.
<code>msoBarBottom</code>	Befehlsleiste wird am unteren Seitenrand verankert.
<code>msoBarFloating</code>	Gibt an, dass die neue Befehlsleiste nicht verankert ist. Wenn die erstellte Leiste eine Symbolleiste ist und das Argument nicht angegeben wird, wird die Leiste frei im Tabellenblatt angezeigt.

Tabelle 43: *Position*-Argument legt Seitenrandposition fest

Die folgende Prozedur erstellt am linken Seitenrand eine verankerte Symbolleiste ohne Schaltflächen.

```
'=====
' Auf CD      Buchdaten\Beispiele\06_Befehlsleisten\
' Dateiname  06_07_AddCommandBars.xls
' Modul      mdl_04_AddNormalBarLeft
'=====

Sub AddNormalBarLeft()
    Dim cmdBar As CommandBar

    On Error Resume Next
    Application.CommandBars("MyCommandBarLeft").Delete
    On Error GoTo 0

    Set cmdBar = Application.CommandBars.Add( _
        Name:="MyCommandBarLeft", _
```

```
                Position:=msoBarLeft, _
                Temporary:=True)
    cmdBar.Visible = True
End Sub
```

In der Beispieldatei wurden zwei Schaltflächen die Prozeduren zum Erstellen und Löschen der Leiste zugewiesen.

```
'=====
' Auf CD      Buchdaten\Beispiele\06_Befehlsleisten\
' Dateiname  06_07_AddCommandBars.xls
' Modul      mdl_04_AddNormalBarLeft
'=====

Sub DeleteNormalBarLeft()
    On Error Resume Next
    Application.CommandBars("MyCommandBarLeft").Delete
End Sub
```

194 Symbolleisten schwebend positionieren

Sie können eine Symbolleiste auch an eine bestimmte Stelle auf dem Tabellenblatt positionieren. Die Left- und Top-Eigenschaften legen in Pixel einen Wert für die horizontale und vertikale Position der angegebenen Befehlsleiste relativ zum linken und oberen Rand fest.

In diesem Beispiel wird die obere linke Ecke der schwebenden Befehlsleiste MyCommandBarFloating in einem Abstand von 140 Pixel vom linken Rand und 100 Pixel vom oberen Rand des Bildschirmes positioniert.

```
'=====
' Auf CD      Buchdaten\Beispiele\06_Befehlsleisten\
' Dateiname  06_07_AddCommandBars.xls
' Modul      mdl_05_AddNormalBarFloating
'=====

Sub AddNormalBarFloating()
    Dim cmdBar As CommandBar

    On Error Resume Next
    Application.CommandBars("MyCommandBarFloating").Delete
    On Error GoTo 0

    Set cmdBar = Application.CommandBars.Add( _
        Name:="MyCommandBarFloating", _
        Position:=msoBarFloating, _
        Temporary:=True)

    With cmdBar
        .Left = 700
        .Top = 520
        .Visible = True
    End With
End Sub
```

Grundlagen

Allgemein

Datum und Zeit

Formeln

Statistik

Diagramme

Objekte

Diagramm

Ereignisse

User-Formen

Web/Mail

Extern

Gemischtes

Spezial

Einer Schaltfläche auf dem Tabellenblatt der Beispieldatei wurde die folgende Prozedur zum Löschen der MyCommandBarFloating-Symbolleiste zugewiesen.

```
'=====
' Auf CD      Buchdaten\Beispiele\06_Befehlsleisten\
' Dateiname  06_07_AddCommandBars.xls
' Modul      mdl_05_AddNormalBarFloating
'=====

Sub DeleteNormalBarFloating()
    On Error Resume Next
    Application.CommandBars("MyCommandBarFloating").Delete
End Sub
```

195 Symbolleiste neben Standard-Symbolleiste andocken

Um eine Symbolleiste exakt rechts neben einer bestehenden Symbolleiste auszurichten, muss zunächst die Endposition des rechten Randes ermittelt werden, neben welche die neue Leiste erscheinen soll. Die `Width`-Eigenschaft gibt in Pixel die Breite der angegebenen Befehlsleiste zurück. Die `Left`-Eigenschaft gibt in Pixel den Abstand der Befehlsleiste vom linken Rand des angegebenen Objekts relativ zum Bildschirm zurück. Diese beiden Werte werden addiert und in der Variablen `intWidth` gespeichert.

Nachdem die neue Leiste `MyCommandBar` erstellt wurde, werden in einer `With`-Anweisung die verwendeten Eigenschaften und Einstellungen für die neue Befehlsleiste aufgeführt. Die `Position`-Eigenschaft richtet sich nach der Position der `STANDARD`-Symbolleiste, ebenso wie die `RowIndex`-Eigenschaft. Diese gibt die Andockreihenfolge der angegebenen Befehlsleiste im Verhältnis zu anderen Befehlsleisten im gleichen Andockbereich zurück oder legt sie fest. Dies kann eine Ganzzahl größer als Null oder eine der folgenden `MsoBarRow`-Konstanten sein:

- `msoBarRowFirst`
- `msoBarRowLast`.

Folgende Prozedur positioniert die neue Symbolleiste neben der `STANDARD`-Befehlsleiste. Wenn Sie eine Symbolleiste neben die `FORMAT`-Symbolleiste platzieren wollen, ersetzen Sie den Namen `Standard` durch `Formatting`.

```
'=====
' Auf CD      Buchdaten\Beispiele\06_Befehlsleisten\
' Dateiname  06_07_AddCommandBars.xls
' Modul      mdl_06_AddCommandBarPosition
'=====

Sub AddCommandBarPosition()
    Dim cmdBar As CommandBar
    Dim cmdBarPos As CommandBar
    Dim intWidth As Integer

    Set cmdBarPos = Application.CommandBars("Standard")
    intWidth = cmdBarPos.Width + cmdBarPos.Left
```

```
On Error Resume Next
Application.CommandBars("MyCommandBar").Delete
On Error GoTo 0

Set cmdBar = Application.CommandBars.Add( _
    Name:="MyCommandBar", _
    Temporary:=True)

With cmdBar
    .Position = cmdBarPos.Position
    .RowIndex = cmdBarPos.RowIndex
    .Left = intWidth
    .Visible = True
End With
End Sub

Sub DeleteMyCommandBar()
    On Error Resume Next
    Application.CommandBars("MyCommandBar").Delete
End Sub
```

196 Kontextmenü mit Datum und Zeit erstellen

Üblicherweise werden Kontextmenüs durch Klicken mit der rechten Maustaste angezeigt. Es ist aber ohne weiteres möglich, ein Kontextmenü über eine Prozedur an einer beliebigen Stelle im Tabellenblatt anzuzeigen. Um ein Kontextmenü zu erstellen, erhält das `Position`-Argument der `Add`-Methode die Konstante `msOBarPopup`. Da ein leeres Kontextmenü leicht zu übersehen wäre, fügen wir in folgender Prozedur der Befehlsleiste eine Schaltfläche mit Uhrzeitsymbol `FaceId` hinzu. Die Aufschrift der Schaltfläche enthält das aktuelle Datum und die Uhrzeit. Über die `ShowPopup`-Methode wird das Kontextmenü an den angegebenen Koordinaten angezeigt. Geben Sie keine Koordinaten an, wird das Kontextmenü an der aktuellen Mauszeigerposition angezeigt.

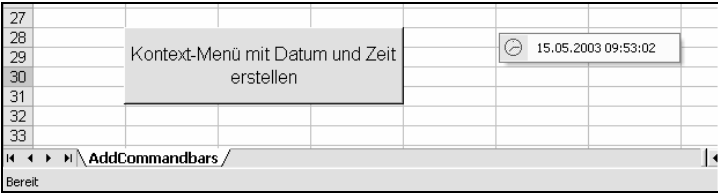


Abbildung 149: Kontextmenü mit Datum und Zeitangaben

Hinweis

Die Anzeige-Koordinaten für Symbolleisten und Kontextmenüs richten sich nicht nach der Größe des Excel-Fensters, sondern nach der gesamten Bildschirmgröße. Wenn Sie das Excel-Fenster verkleinert haben, kann es passieren, dass eine Befehlsleiste mit festen Koordinaten außerhalb der Excel-Anwendung angezeigt wird.

Sobald Sie das Tabellenblatt scrollen oder mit der linken Maustaste an eine beliebige Stelle innerhalb oder außerhalb der Excel-Anwendung klicken, wird das Kontextmenü ausgeblendet und die Prozedur fortgeführt.

Grundlagen

Allgemein

Datum und Zeit

Funktionen

Steuerungselemente

Befehlsleisten

Objekte

Diagramme

Ereignisse

User-Formen

Web/Mail

Extern

Gemischtes

Spezial

Tipp

Diese Art von Kontextmenü eignet sich zum Beispiel, um dem Anwender kurze Mitteilungen anstelle einer MsgBox anzuzeigen oder um darin weitere Prozeduren zur Verfügung zu stellen.

```
' =====
' Auf CD      Buchdaten\Beispiele\06_Befehlsleisten\
' Dateiname   06_07_AddCommandBars.xls
' Modul       mdl_07_AddPopUp
' =====
```

```
Sub AddPopUp()
    Dim cmdBar As CommandBar
    Dim cmdBtn As CommandBarButton

    On Error Resume Next
    Application.CommandBars("MyContext").Delete
    On Error GoTo 0

    Set cmdBar = Application.CommandBars.Add( _
        Name:="MyContext", _
        Position:=msoBarPopup, _
        Temporary:=True)

    Set cmdBtn = cmdBar.Controls.Add
    With cmdBtn
        .FaceId = 126
        .Caption = Date & " " & Time
    End With

    ' Kontextmenü mit Positionsangaben anzeigen
    cmdBar.ShowPopup 430, 600
End Sub
```

197 Neues Farben-Menü erzeugen

In diesem Beispiel wird rechts neben dem ?-Menü ein neues DropDown-Menü mit weiteren Schaltflächen eingefügt. Diesen Schaltflächen sind verschiedene Prozeduren hinterlegt, welche den Zellhintergrund der selektierten Zellen einfärben.

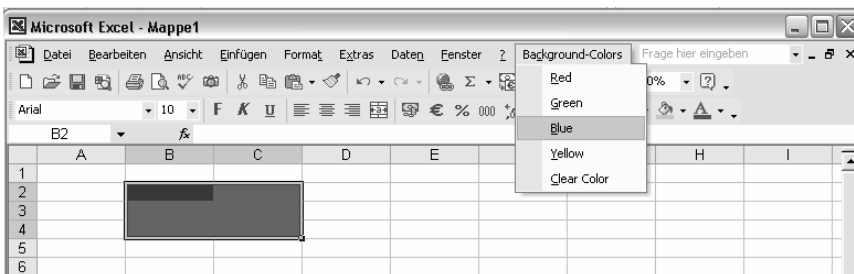


Abbildung 150: Neues DropDown-Menü mit Schaltflächen

Beim Öffnen und beim Wechsel von einer anderen Excel-Arbeitsmappe zu der Beispielmappe wird im `Workbook_Activate`-Ereignis die Prozedur `AddColorMenu` aufgerufen. Diese erstellt das neue Menü. Wechseln Sie zu einer anderen Excel-Datei oder beenden Sie die Beispieldatei bzw. die Excel-Anwendung, wird das `Workbook_Deactivate`-Ereignis ausgelöst. Hier wird das Menü gelöscht, damit es nicht in anderen Excel-Dateien angezeigt wird.

```
'=====
' Auf CD      Buchdaten\Beispiele\06_Befehlsleisten\
' Dateiname   06_08_AddColorMenu.xls
' Ereignis    DieseArbeitsmappe
'=====

Private Sub Workbook_Activate()
    AddColorMenu
End Sub

Private Sub Workbook_Deactivate()
    On Error Resume Next
    Application.CommandBars.ActiveMenuBar. _
        Controls("Background-Colors").Delete
End Sub
```

Das neue Menü soll auf der aktiven Menüleiste erstellt werden. Dementsprechend setzen wir einen Verweis auf die Variable `mCmdBar`, welche als `CommandBar` deklariert wurde.

Bevor ein neues Menü oder Steuerelement erstellt wird, wird vorab ein eventuell gleichnamiges Menü gelöscht. Excel würde sonst bei jedem Durchlauf der Prozedur das Menü erneut erstellen. Für den Fall, dass das Menü nicht existiert, haben wir die `On Error Resume Next`-Fehlerroutine vorangestellt.

Über die `Add`-Methode wird ein neues Steuerelement erzeugt und gleichzeitig ein Verweis auf die Variable `mCmdBarCtrl` gesetzt. Über das `Type`-Argument legen Sie den Typ des Steuerelements, das zur angegebenen Befehlsleiste hinzugefügt werden soll, fest. Es stehen folgende `MsoControlType`-Konstanten zur Verfügung:

Konstante	Beschreibung
<code>msoControlButton</code>	Erstellt ein Schaltflächen-Element
<code>msoControlEdit</code>	Erstellt ein Eingabefeld
<code>msoControlDropdown</code>	Erstellt ein Dropdown-Feld
<code>msoControlComboBox</code>	Erstellt ein Kombinationsfeld
<code>msoControlPopup</code>	Erstellt ein Dropdown-Menü

Tabelle 44: Mögliche Steuerelemente des `Type`-Arguments

Den Wert des optionalen `Temporary`-Argumentes wird auf `True` gesetzt, um das neue Steuerelement als temporäres Steuerelement zu erstellen. Temporäre Steuerelemente werden beim Schließen der Containeranwendung automatisch gelöscht. Über die `Caption`-Eigenschaft wird der Schaltfläche der Beschriftungstext zugewiesen.

```
ctlBarControl.Caption = "&Background-Colors"
```

Durch das vorangestellte &-Zeichen wird der nachfolgende Buchstabe unterstrichen dargestellt. Das jeweilige Menü bzw. Schaltfläche kann dadurch mit der Tastenkombination **[Alt] + [Buchstabe]** aufgerufen werden. Drücken Sie die Tastenkombination **[Alt] + [C]**, um das Menü BACKGROUND-COLORS aufzuklappen. Halten Sie die **[Alt]**-Taste gedrückt und drücken Sie die Taste **[B]** für die Schaltfläche BLUE, erhalten die selektierten Zellen einen blauen Zellenhintergrund.



Abbildung 151: Dropdown-Menü mit Tastenkombinations-Zeichen

Um die einzelnen Schaltflächen des Dropdown-Menüs zu erstellen, wird jeweils die Prozedur `AddControl` aufgerufen. Diese Prozedur erwartet beim Aufruf die Argumente für den Beschriftungstext `Caption` und den Namen des aufzurufenden Makros `OnAction` des Steuerelements. Die verwendeten Variablen werden oberhalb der ersten Prozedur `AddColorMenu` deklariert, damit auch die zweite Prozedur `AddControl` auf diese Variablen zugreifen kann.

```
'=====
' Auf CD      Buchdaten\Beispiele\06_Befehlsleisten\
' Dateiname   06_08_AddColorMenu.xls
' Modul       mdl_01_ColorMenu
'=====
```

```
Private mCmdBar      As CommandBar
Private mCmdBarCtrl  As CommandBarControl
Private mCtrlColor   As CommandBarControl
```

```
Sub AddColorMenu()
    Set mCmdBar = Application.CommandBars.ActiveMenuBar

    On Error Resume Next
    mCmdBar.Controls("Background-Colors").Delete
    On Error GoTo 0

    Set mCmdBarCtrl = mCmdBar.Controls.Add( _
                                                Type:=msoControlPopup, _
                                                Temporary:=True)

    mCmdBarCtrl.Caption = "Ba&ckground-Colors"

    AddControl "&Red", "RedColor"
    AddControl "&Green", "GreenColor"
    AddControl "&Blue", "BlueColor"
    AddControl "&Yellow", "YellowColor"
    AddControl "&Clear Color", "NoColor"
End Sub
```

Beim Aufruf der folgenden Prozedur wird der Beschriftungstext sowie der Name der aufzurufenden Prozedur übergeben. Über die Add-Methode wird dem Dropdown-Menü `mCmdBarCtrl` eine Schaltfläche hinzugefügt. In einer With-Anweisung wird der Beschriftungstext über die Variable `strCaption` an die `Caption`-Eigenschaft übergeben. Die `Caption`-Eigenschaft erhält den Namen des Visual-Basic-Makros, das ausgeführt wird, wenn der Benutzer auf den Wert des Befehlsleisten-Steurelements klickt. Diese ausgelagerte Prozedur erspart die fünffache Angabe der Befehle in der ersten Prozedur `AddColorMenu`.

```
'=====
' Auf CD      Buchdaten\Beispiele\06_Befehlsleisten\
' Dateiname   06_08_AddColorMenu.xls
' Modul       mdl_01_ColorMenu
'=====

Sub AddControl(strCaption As String, strAction As String)

    Set mCtrlColor = ctlBarControl.CommandBar.Controls.Add _
        (Type:=msoControlButton, _
         Temporary:=True)

    With mCtrlColor
        .Caption = strCaption
        .OnAction = strAction
    End With
End Sub
```

Die folgenden Prozeduren werden ausgeführt, wenn der Benutzer auf eine Schaltfläche im Menü BACKGROUND-COLORS klickt.

```
'=====
' Auf CD      Buchdaten\Beispiele\06_Befehlsleisten\
' Dateiname   06_08_AddColorMenu.xls
' Modul       mdl_01_ColorMenu
'=====

Sub RedColor()
    Selection.Interior.ColorIndex = 3
End Sub

Sub GreenColor()
    Selection.Interior.ColorIndex = 4
End Sub

Sub BlueColor()
    Selection.Interior.ColorIndex = 5
End Sub

Sub YellowColor()
    Selection.Interior.ColorIndex = 6
End Sub
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

```

Sub NoColor()
    Selection.Interior.ColorIndex = xlNone
End Sub

```

198 Menü suchen und aufklappen

Im folgenden Beispiel soll ein beliebiges Menü der aktiven Menüleiste angezeigt bzw. aufgeklappt werden. In eine `InputBox` wird der Name des gewünschten Menüs eingegeben und in der Variablen `strCtrl` gespeichert. Wenn die Eingabezeile keinen Wert enthält, wird ein Leerstring "" zurückgegeben. Hat der Benutzer die ABBRECHEN-Schaltfläche der `InputBox` geklickt, enthält `strCtrl` den Wert `Falsch`. In beiden Fällen wird die Prozedur beendet.

Jede Menübeschriftung enthält einen unterstrichenen Buchstaben, damit die Menüs mit der Tastenkombination `[Alt] + [Buchstabe]` aufgerufen werden können. Der Unterstrich wird durch ein vorangestelltes `&`-Zeichen erzeugt. Die Zeichenfolge für das `FORMAT`-Menü lautet zum Beispiel »Forma&t« und wird mit der Tastenkombination `[Alt] + [T]` aufgerufen. In der Prozedur `FindMenu` wird über die `Caption`-Eigenschaft der Menüschaltflächen die Menübeschriftung mit der Eingabe aus der `InputBox` verglichen. Da der Vergleich exakt übereinstimmen muss, müsste normalerweise der Menüname inklusive des Unterstrichs »Forma&t« in der `InputBox` angegeben werden. Um dem Benutzer die Eingabe zu erleichtern, wird bei der Eingabe des Menünamens in die `InputBox` auf die Angabe des `&`-Zeichens verzichtet. Die Eingabe lautet beispielsweise »Format«.

In der folgenden `For Each...Next`-Anweisung wird jedes Menü der aktiven Menüleiste durchlaufen. Um ein Menü anzusprechen, wird dessen Existenz über die `Caption`-Eigenschaft überprüft. Über die Excel-Tabellenfunktionen `WECHSELN()`, in VBA mit `Substitute` angewandt, wird das `&`-Zeichen durch einen Leerstring ausgetauscht. Dadurch ist in der folgenden `If`-Anweisung ein einwandfreier Vergleich zwischen der `Caption`-Eigenschaft und der Variablen `strCtrl` möglich. Ist die `Enabled`-Eigenschaft eines Menüs auf `False` eingestellt, wird es zwar gefunden, kann aber, da es ausgegraut und somit inaktiv ist, nicht angezeigt werden. Das Gleiche gilt für die `Visible`-Eigenschaft. Nur wenn alle drei Kriterien, welche zum Anzeigen eines Menüs erforderlich sind, in der `If`-Anweisung zutreffen, wird die Variable `blnCtrl` auf `True` gestellt, ein Verweis auf das gefundene Steuerelement `cmdBarCtrl` gesetzt und die `For Each...Next`-Anweisung verlassen.

In der abschließenden `If...Then...Else`-Anweisung wird anhand des Wahrheitswertes der Variablen `blnCtrl` ein Meldungsfenster ausgegeben, ob das Menü gefunden wurde.

Hinweis

Der Standardwert einer booleschen Variablen ist `True` und muss deshalb nicht ausdrücklich in einer `If`-Anweisung angegeben werden.

Die Syntax `If blnCtrl then` entspricht der Syntax `If blnCtrl = True then`

Über die `Execute`-Methode wird das vorangestellte Steuerelement angezeigt.

```

' =====
' Auf CD      Buchdaten\Beispiele\06_Befehlsleisten\
' Dateiname   06_09_FindMenu.xls
' Tabelle      FindMenu
' Modul       mdl_01_FindMenu
' =====

```

```

Sub FindMenu()
    Dim cmdBar As CommandBar

```

```
Dim cmdBarCtrl As CommandBarControl
Dim strCtrl As String
Dim strCtrlCaption As String
Dim blnCtrl As Boolean

Set cmdBar = Application.CommandBars.ActiveMenuBar

strCtrl = Application.InputBox( _
    Prompt:="Geben Sie den Namen eines Menüs" _
    & " der aktiven Menüleiste ein:", _
    Title:="Suche Menü", _
    Default:="Format", _
    Left:=260, _
    Top:=230, _
    Type:=2)

If strCtrl = "" Or strCtrl = "Falsch" Then Exit Sub

For Each cmdBarCtrl In cmdBar.Controls

    strCtrlCaption = Application.WorksheetFunction. _
        Substitute(cmdBarCtrl.Caption, "&", "")

    If strCtrlCaption = strCtrl And _
        cmdBarCtrl.Enabled And _
        cmdBarCtrl.Visible Then

        blnCtrl = True
        Set cmdBarCtrl = cmdBar.Controls(strCtrl)
        Exit For
    End If

Next cmdBarCtrl

If blnCtrl Then
    MsgBox "Menü '" & strCtrl & "' gefunden"
    cmdBarCtrl.Execute
Else
    MsgBox "Menü '" & strCtrl & "' nicht gefunden"
End If
End Sub
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

199 Arbeiten mit personalisierten Menüs

In Excel 2000 wurde die Funktion PERSONALISIERTE MENÜS eingeführt. Standardmäßig sind in allen Microsoft Office-Programmen personalisierte Menüs aktiviert. Wenn die Funktion aktiviert ist, werden alle Befehle in den Menüs angezeigt. Wenn sie deaktiviert wurde, wird eine reduzierte Teilmenge der am häufigsten verwendeten Menüelemente angezeigt. In der Excel-Anwendung kann die Funktion im Menü ANSICHT | SYMBOLLEISTEN | ANPASSEN auf der Registerkarte OPTIONEN im Kontrollkästchen MENÜS IMMER VOLLSTÄNDIG ANZEIGEN deaktiviert und aktiviert werden. Je nach Einstellung im ANPASSEN-Dialog (siehe Abbildung 152) zeigt Excel nach kurzer Zeit alle Befehle in einem Menü an, wenn Sie den Mauszeiger über dem geöffneten Menü positionieren.

Das Aktivieren oder Deaktivieren dieser Funktion wirkt sich auf alle Microsoft Office-Programme aus.



Abbildung 152: Einstellungen für personalisierte Menüs

Mit VBA ändern Sie den Status der `AdaptiveMenus`-Eigenschaft, indem Sie den Wahrheitswert auf `True` oder `False` stellen. `False` deaktiviert, `True` aktiviert personalisierte Menüs.

Die Funktion `PERSONALISIERTE MENÜS` gilt für alle Menü- und Symbolleisten, aber nicht für Kontextmenüs.

In der Beispieldatei wurden die Prozeduren zur Deaktivierung und Aktivierung von personalisierten Menüs zwei Schaltflächen zugewiesen. Beim Aufruf der Prozedur `AdaptiveMenu` wird der entsprechende Wahrheitswert an die aufgerufene Prozedur übergeben. Nachdem die Einstellung geändert wurde, wird das `BEARBEITEN`-Menü über die `Execute`-Methode angezeigt.

```
'=====
' Auf CD      Buchdaten\Beispiele\06_Befehlsleisten\
' Dateiname   06_09_FindMenu.xls
' Tabelle     AdaptiveMenus
' Modul       mdl_02_AdaptiveMenu
'=====
```

```
Sub OnPersonalizedMenu()
    AdaptiveMenu True
End Sub
```

```
Sub OffPersonalizedMenu()
    AdaptiveMenu False
End Sub
```



```
Sub AdaptiveMenu(blnState As Boolean)
    Dim cmdBarCtrl As CommandBarControl

    Set cmdBarCtrl = CommandBars(1).Controls("Bearbeiten")

    Application.CommandBars.AdaptiveMenus = blnState

    cmdBarCtrl.Execute
End Sub
```

200 Steuerelemente kopieren

In Excel können Sie mit der Copy-Methode vorhandene Steuerelemente von einer Befehlsleiste in eine andere kopieren. Die Syntax lautet:

```
Ausdruck.Copy(Bar, Before)
```

In unserem Beispiel wird die Copy-Methode wie folgt angewandt:

```
cmdBarCtrl.Copy Bar:=cmdBar, Before:=1
```

Wird nur das erste Argument bzw. beide optionalen Argumente Bar und Before verwendet, kann auf die Benennung der Argumente verzichtet werden. Durch diese Schreibweise wird der Code allerdings schwerer verständlich:

```
cmdBarCtrl.Copy cmdBar, 1
```

Ausdruck stellt das zu kopierende Steuerelement dar und kann eines der folgenden Objekte sein, welches exakt referenziert werden muss.

Objekt	Beschreibung
CommandBarControl	Befehlsleisten-Steuerelement
CommandBarButton	Schaltflächen-Steuerelement
CommandBarPopup	Popupsteuerelement
CommandBarComboBox	Kombinationsfeld-Steuerelement

Tabelle 45: CommandBar-Objekte, die kopiert werden können

In unserem Beispiel wird für das zu kopierende Steuerelement die Variable cmdBarCtrl reserviert. Es wird ein Verweis auf das Element REGISTERFARBE im Menü FORMAT |BLATT gesetzt, das kopiert werden soll.

Über das optionale Bar-Argument legen Sie die Zielbefehlsleiste fest, in die das kopierte Steuerelement eingefügt werden soll. Wenn Sie dieses Argument auslassen, wird das Steuerelement in dieselbe Befehlsleiste kopiert, in der es sich bereits befindet. Da das Element in die benutzerdefinierte Custom-Symbolleiste eingefügt werden soll, wird in unserem Beispiel ein entsprechender Verweis auf die Variable cmdBar gesetzt.

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

Geben Sie für das optionale *Before*-Argument eine Zahl an, welche die Position für das neue Steuerelement auf der angegebenen Befehlsleiste festlegt. Das neue Steuerelement wird vor dem Steuerelement eingefügt, das sich an dieser Position befindet. Wenn dieses Argument ausgelassen wird, wird das Element an das Ende der Befehlsleiste kopiert. In der Beispieldatei wird das kopierte Element an erster Stelle eingefügt.

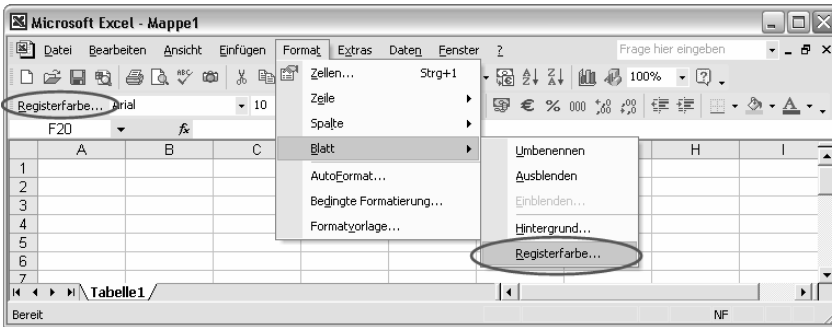


Abbildung 153: Kopiertes Steuerelement

Damit bei erneutem Ausführen der Prozedur nicht weitere gleiche Steuerelemente eingefügt werden, wird vorab ein eventuell vorhandenes gleichnamiges Steuerelement gelöscht.

Hinweis

Befinden sich mehrere gleichnamige Steuerelemente in der genannten Befehlsleiste, wird das von links zuerst gefundene Element gelöscht. Dies kann unter Umständen das Original-Steuerelement der Standardeinstellung sein.

```
'=====
' Auf CD      Buchdaten\Beispiele\06_Befehlsleisten\
' Dateiname   06_09_FindMenu.xls
' Tabelle     CopyControl
' Modul       mdl_03_CopyControl1
'=====
```

```
Sub CopyControl1()
    Dim cmdBar As CommandBar
    Dim cmdBarCtrl As CommandBarControl

    ' Verweis auf Format-Symbolleiste setzen
    Set cmdBar = Application.CommandBars("Formatting")

    ' Evtl. vorhandenes Steuerelement löschen
    On Error Resume Next
    cmdBar.Controls("Registerfarbe...").Delete
    On Error GoTo 0

    ' Verweis auf zu kopierenedes Element setzen
    Set cmdBarCtrl = Application.CommandBars(1) _
        .Controls("Format") _
        .Controls("Blatt") _
```

```
        .Controls("Registerfarbe...")

' Steuerelement kopieren und in Format-Symbolleiste
' an erster Stelle einfügen
cmdBarCtrl.Copy Bar:=cmdBar, Before:=1
End Sub
```

Hinweis Bei Verwendung der Copy-Methode kann das neue Element nicht temporär erstellt werden. Wenn Sie es vor Beendigung der Excel-Anwendung nicht löschen, wird es dauerhaft in der Symbolleistenkonfigurationsdatei *.xlb gespeichert und steht, bis es gelöscht wird, in folgenden Excel-Anwendungen zur Verfügung.

In der Beispieldatei können Sie über eine Schaltfläche das kopierte Element löschen. Dieser haben wir die Prozedur DeleteCopyControl zugewiesen.

```
'=====
' Auf CD      Buchdaten\Beispiele\06_Befehlsleisten\
' Dateiname   06_09_FindMenu.xls
' Tabelle     CopyControl
' Modul       mdl_03_CopyControl1
'=====

Sub DeleteCopyControl()
    On Error Resume Next
    Application.CommandBars("Formatting") _
        .Controls("Registerfarbe...").Delete
End Sub
```

Spätestens beim Beenden der Beispiel-Arbeitsmappe wird das kopierte Steuerelement REGISTERFARBE im Workbook_BeforeClose-Ereignis durch den Aufruf der DeleteCopyControl-Prozedur gelöscht.

```
'=====
' Auf CD      Buchdaten\Beispiele\06_Befehlsleisten\
' Dateiname   06_09_FindMenu.xls
' Ereignis     DieseArbeitsmappe
'=====

Private Sub Workbook_BeforeClose(Cancel As Boolean)
    DeleteCopyControl
End Sub
```

201 Integrierte Steuerelemente in Symbolleiste einfügen

Um ein integriertes Steuerelement über die Copy-Methode in eine benutzerdefinierte Symbolleiste einzufügen, setzen Sie zunächst einen Verweis auf die eingefügte Leiste und anschließend einen Verweis auf das zu kopierende Element. Die Syntax cmdBarCtrl.Copy cmdBar fügt das kopierte Steuerelement in der neuen Befehlsleiste an der letzten Stelle der Controls ein.

Grundlagen

Allgemein

Datum
Zeit

ungen

Steuerelemente

Befehlsleisten

Objekt

Diagramm

Ereignisse

User-Form

Web/Mail

Extern

Gemischtes

Spezial

Das Klicken auf diese Schaltfläche hat denselben Effekt wie das Klicken auf die Original-Schaltfläche.

Die Symbolleiste CUSTOM wird nicht ausdrücklich gelöscht, da sie temporär erstellt wurde und nur für die Dauer der Excel-Instanz angezeigt wird.

```
'=====
' Auf CD      Buchdaten\Beispiele\06_Befehlsleisten\
' Dateiname  06_09_FindMenu.xls
' Tabelle    CopyControl
' Modul      mdl_04_CopyControl2
'=====
```

```
Sub CopyControl2()
    Dim cmdBar      As CommandBar
    Dim cmdBarCtrl  As CommandBarControl

    ' Evtl. vorhandene Befehlsleiste löschen
    On Error Resume Next
    cmdBar("Custom").Delete
    On Error GoTo 0

    ' Verweis auf neue Befehlsleiste setzen
    Set cmdBar = CommandBars.Add( _
        Name:="Custom", _
        Position:=msoBarTop, _
        Temporary:=True)

    ' Verweis auf zu kopierendes Element setzen
    Set cmdBarCtrl = Application.CommandBars(1) _
        .Controls("Format") _
        .Controls("Blatt") _
        .Controls("Registerfarbe...")

    ' Steuerelement kopieren und in neue Leiste
    ' an letzter Stelle einfügen
    cmdBarCtrl.Copy cmdBar

    ' neue Symbolleiste anzeigen
    cmdBar.Visible = True
End Sub
```

202 Schaltflächen-ID ermitteln

Um ein Steuerelement über seine ID anzusprechen, müssen Sie zunächst dessen ID ermitteln. Dies erreichen Sie zum einen über eine Auflistung aller Steuerelemente einer Befehlsleiste (siehe Rezepte 170, 171 und 174). Die ID eines einzelnen Steuerelementes erfahren Sie direkt über nachfolgende Prozedur. Geben Sie die genaue Verschachtelung des gewünschten Elementes an.

```
'=====
' Auf CD      Buchdaten\Beispiele\06_Befehlsleisten\
' Dateiname   06_09_FindMenu.xls
' Tabelle     FindID
' Modul       mdl_05_FindID
'=====

Sub FindID()
    MsgBox _
        "Der Menüpunkt 'Registerfarbe' hat die ID:" & _
        vbNewLine & _
        Application.CommandBars(1) _
            .Controls("Format") _
            .Controls("Blatt") _
            .Controls("Registerfarbe...").ID
End Sub
```

203 Schaltflächen mit Quickinfo erstellen

Im folgenden Beispiel werden alle Steuerelemente der FORMAT-Symbolleiste über die Copy-Methode in eine neue Leiste eingefügt. Mittels der TooltipText-Eigenschaft wird für jede Schaltfläche zusätzlich zu der Beschreibung ihre ID in der Quickinfo angezeigt.

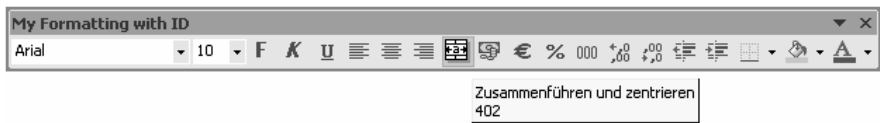


Abbildung 154: Format-Symbolleiste mit ID in Quickinfo

```
'=====
' Auf CD      Buchdaten\Beispiele\06_Befehlsleisten\
' Dateiname   06_09_FindMenu.xls
' Tabelle     FindID
' Modul       mdl_06_IDsInTooltips
'=====

Sub IDsInTooltips()
    Dim strCmdBar As String
    Dim cmdBar As CommandBar
    Dim cmdBarCtrl As CommandBarControl
    Dim intCtrl As Integer

    strCmdBar = "Formatting" ' "Standard"
                                ' kopiert die Befehlsleiste Standard

    On Error Resume Next
    ' evtl. vorhandene Leiste löschen
    CommandBars("My " & strCmdBar & " with ID").Delete

    Set cmdBar = Application.CommandBars.Add( _
        Name:="My " & strCmdBar & " with ID", _
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

```

        Temporary:=True)

For intCtrl = 1 To CommandBars(strCmdBar).Controls.Count
    ' Verweis auf Control setzen
    Set cmdBarCtrl = Application.CommandBars(strCmdBar) _
        .Controls(intCtrl)
    ' Elemente der Original-Leiste in neue Leiste kopieren
    cmdBarCtrl.Copy cmdBar

    ' Text für Quickinfo festlegen
    With cmdBar.Controls(intCtrl)
        .TooltipText = .TooltipText & Chr(10) & .ID
    End With
Next intCtrl

With cmdBar
    .Top = 170
    .Left = 130
    .Width = 1500
    .Visible = True
End With
End Sub

```

204 Symbolleiste mit zwei Ebenen erstellen

Über die Copy-Methode werden in unserem nächsten Beispiel alle Steuerelemente der Standard-Symbolleiste in eine neue Leiste eingefügt. Das Besondere an dieser Symbolleiste ist, dass für jede Schaltfläche eine weitere Schaltfläche mit der jeweiligen ID als Beschriftungstext eingefügt wird. Durch die Zwischenspeicherung der Leistenbreite in `intWidth` wird die Befehlsleiste quasi umbrochen und die zusätzlichen ID-Schaltflächen erscheinen unterhalb der Symbole. Die genaue Vorgehensweise der Prozedur erfahren Sie über die Kommentare im Code.

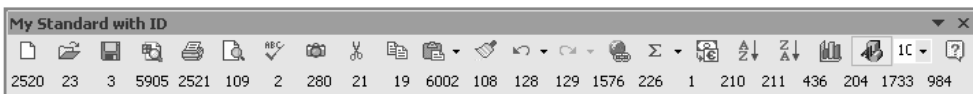


Abbildung 155: Standard-Symbolleiste mit Controls zur ID-Beschriftung

```

' =====
' Auf CD      Buchdaten\Beispiele\06_Befehlsleisten\
' Dateiname   06_09_FindMenu.xls
' Tabelle     FindID
' Modul       mdl_07_StandardIDs
' =====

Sub StandardIDs()
    Dim strCmdBar As String
    Dim cmdBar    As CommandBar
    Dim cmdBarCtrl As CommandBarControl
    Dim intCtrl   As Integer
    Dim intWidth  As Integer

```

```

Dim CtrlDummy As CommandBarButton

strCmdBar = "Standard" ' "Formatting"
                        ' kopiert die Befehlsleiste Format

On Error Resume Next
' evtl. vorhandene Leiste löschen
CommandBars("My " & strCmdBar & " with ID").Delete

Set cmdBar = Application.CommandBars.Add( _
    Name:="My " & strCmdBar & " with ID", _
    Temporary:=True)

For Each cmdBarCtrl In CommandBars(strCmdBar).Controls
    ' nur sichtbare Controls kopieren
    If cmdBarCtrl.Visible = True Then
        ' Elemente der Original-Leiste in neue Leiste kopieren
        cmdBarCtrl.Copy cmdBar
        ' Controls hochzählen
        intCtrl = intCtrl + 1
        ' Breite der neuen Schaltfläche anpassen
        cmdBar.Controls(intCtrl).Width = 30
    End If
Next cmdBarCtrl

' Breite der Menüleiste (Icons) zwischenspeichern
intWidth = cmdBar.Width

For intCtrl = 1 To cmdBar.Controls.Count
    ' Für jedes Element einen weiteren Button einfügen
    ' und mit ID beschriften
    Set CtrlDummy = cmdBar.Controls.Add(Type:=msoControlButton)

    With CtrlDummy
        .Style = msoButtonCaption
        .Caption = cmdBar.Controls(intCtrl).ID
        ' Breite der Icon-Schaltflächen anpassen
        .Width = 30
    End With
Next intCtrl

With cmdBar
    .Top = 320
    .Left = 100
    .Visible = True
    ' Leistenbreite vergrößern, damit letzte Schaltfläche nicht
    ' umbrochen wird
    .Width = intWidth + 20
End With
End Sub

```

Grund-
lagenAllge-
meinDatum
Zeit

tungen

Steuer
elemenBefehl
leisten

Objekt

Dia-
grammEreig-
nisseUser-
FormWeb/
Mail

Extern

Ge-
misch
tes

Specia

205 Steuerelemente mit ID erstellen und anzeigen

In Rezept 200 wurde ein integriertes Steuerelement über die Copy-Methode an eine bestimmte Position in einer Symbolleiste eingefügt. Diesen Vorgang können Sie ebenfalls unter Zuhilfenahme der Control-ID vornehmen. Diese Variante hat den Vorteil, dass das Steuerelement temporär erstellt werden kann. Es muss dadurch bei Beendigung der Excel-Anwendung nicht ausdrücklich gelöscht werden. Des Weiteren können Sie die Style-Eigenschaft verändern, die angibt, auf welche Art ein Schaltflächen-Steuerelement in der Befehlsleiste angezeigt werden soll. Folgende Darstellungsarten stehen dabei zur Verfügung.

Konstante	Darstellungsart
MsoButtonAutomatic	Standardeinstellung der Schaltfläche
msoButtonCaption	Nur Text in einer Zeile (max. 255 Zeichen) Anzeige von max. 86 Zeichen
msoButtonIcon	Nur Symbol
msoButtonIconAndCaption	Symbol und Text in einer Zeile
msoButtonIconAndCaptionBelow	Text in einer Zeile unterhalb Symbol
msoButtonIconAndWrapCaption	Umbrochener Text (Anzeige max. 255 Zeichen) rechts neben Symbol
msoButtonIconAndWrapCaptionBelow	Umbrochener Text unterhalb Symbol
msoButtonWrapCaption	Nur umbrochener Text

Tabelle 46: Darstellungsarten für Schaltflächen-Steuerelemente

Das integrierte Steuerelement REGISTERFARBE weist den msoButtonCaption-Stil auf. Dadurch ist lediglich ein Beschriftungstext möglich. In diesem Beispiel wird die Style-Eigenschaft auf msoButtonIconAndCaption geändert. Über die FaceID-Eigenschaft kann dem angegebenen Schaltflächen-Steuerelement ein Schaltflächensymbol zugewiesen werden. Ebenso können Sie den Beschriftungstext über die Caption-Eigenschaft ändern.

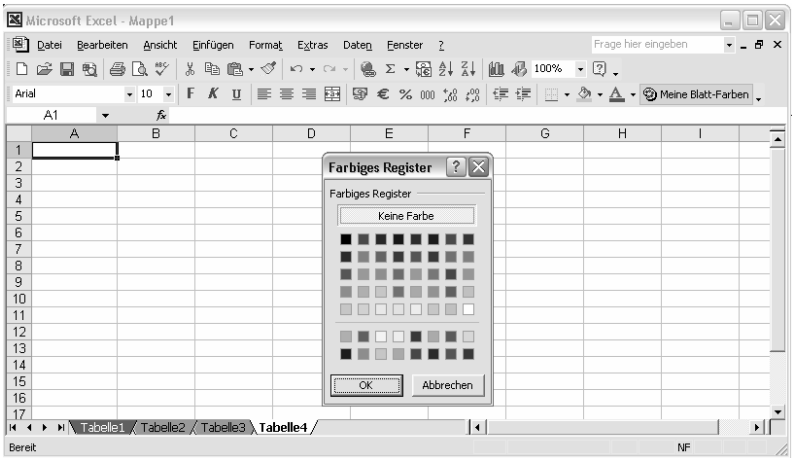


Abbildung 156: Integriertes Steuerelement mit neuem Symbol und Beschriftung

Tipp

Auf der beiliegenden CD finden Sie im Ordner *Buchdaten\Beispiele\06_Befehlsleisten* die Datei *FaceID.xls*. Diese Arbeitsmappe enthält alle in Microsoft-Office-Programmen zur Verfügung stehenden Schaltflächensymbole mit den dazugehörigen FaceId-Nummern.

Übergeben Sie die ID des gewünschten Steuerelements an die Variable `intControl`. Diese wird im weiteren Code an die Stelle der ID-Nummer eingesetzt. Das angegebene Steuerelement wird vor Beendigung der Prozedur durch die `Execute`-Methode angezeigt.

```
'=====
' Auf CD      Buchdaten\Beispiele\06_Befehlsleisten\
' Dateiname  06_09_FindMenu.xls
' Tabelle    ControlWithID
' Modul      mdl_05_ControlWithID
'=====

Sub ControlWithID()
    Dim cmdBarCtrl As CommandBarControl
    Dim intControl As Integer

    intControl = 5747

    On Error Resume Next
    CommandBars("Formatting").FindControl(ID:=intControl).Delete
    On Error GoTo 0

    Set cmdBarCtrl = CommandBars("Formatting").Controls.Add( _
                                                ID:=intControl, _
                                                Temporary:=True)

    With cmdBarCtrl
        .Style = msoButtonIconAndCaption
        .FaceId = 2167
        .Caption = "Meine Blatt-Farben"
        .Execute
    End With
End Sub
```

206 Schaltflächen über ID löschen

Um eine Schaltfläche über die ID-Nummer zu löschen, setzen Sie einen Verweis auf die Befehlsleiste, in welcher sich das Steuerelement befindet.

Wenn die Befehlsleiste zwei oder mehr Steuerelemente enthält, die den Suchkriterien entsprechen, gibt `FindControl` das erste Steuerelement zurück, das gefunden wurde. Wenn kein Steuerelement gefunden wurde, das den Kriterien entspricht, gibt die `FindControl`-Methode `Nothing` zurück. In einer `If`-Anweisung wird überprüft, ob die Objektvariable `cmdBarCtrl` einen Verweis enthält und somit das zu löschende Steuerelement gefunden wurde. Dies stellt eine weitere Fehlerroutine zum Löschen von Steuerelementen dar.

```
'=====
' Auf CD      Buchdaten\Beispiele\06_Befehlsleisten\
' Dateiname  06_09_FindMenu.xls
```

Grundlagen
Allgemein
Datum und Zeit

Diagramme
Steuerelemente
Befehlsleisten

Objekt
Diagramm

Ereignisse
User-Form

Web/Mail
Extern

Gemischtes
Spezial

```
' Tabelle      ControlWithID
' Modul       mdl_05_ControlWithID
' =====

Sub DeleteWithID()
    Dim cmdBarCtrl As CommandBarControl

    Set cmdBarCtrl = CommandBars("Formatting").FindControl(ID:=5747)

    If Not cmdBarCtrl Is Nothing Then cmdBarCtrl.Delete
End Sub
```

207 Menüpunkte mit Hyperlinks

In einer Zelle im Tabellenblatt können Sie über das Menü EINFÜGEN | HYPERLINK oder die Tastenkombination **[Strg]+[K]** Hyperlinks erstellen. Diese können zu verschiedenen Objekten verlinken:

- ▶ Datei oder Webseite
- ▶ Bezug innerhalb der aktuellen Arbeitsmappe
- ▶ Zu einem anderen Dokument
- ▶ E-Mail-Adresse

Ebenso ist es möglich, einer Befehlsschaltfläche einen Hyperlink zu einer Webseite zuzuweisen. Die Prozedur `MenuHyperlinks` zum Erstellen des neuen Hyperlink-Menüs wird beim Öffnen der Beispieldatei im `Workbook_Open`-Ereignis aufgerufen. Wenn Sie die Excel-Anwendung beenden, wird das Menü automatisch aus dem Speicher gelöscht, da es temporär erstellt wurde. Aus diesem Grund haben wir keine Prozedur zum Löschen des Menüs vorgesehen.

```
' =====
' Auf CD      Buchdaten\Beispiele\06_Befehlsleisten\
' Dateiname   06_10_Hyperlink.xls
' Ereignis    DieseArbeitsmappe
' =====

Private Sub Workbook_Open()
    Call MenuHyperlinks
End Sub
```

Die Prozedur `MenuHyperlinks` fügt an die letzte Stelle der aktiven Menüleiste ein neues Dropdown-Menü mit weiteren Schaltflächen ein. Klicken Sie auf eine dieser Schaltflächen, wird Ihr Standard-Browser gestartet und ruft die hinterlegte Adresse der Internetseite auf.

Achtung

Wenn Sie nicht mit dem Internet verbunden sind, während Sie in der Beispieldatei auf einen Hyperlink klicken, erscheint nach kurzer Zeit eine Fehlermeldung.

Um das neue Menü `HYPERLINKS` zu erstellen, wird an die Variable `mCmdBarPopUp` ein Verweis auf dieses Menü gesetzt. Damit die Variable `mCmdBarPopUp` erhalten bleibt und in der zweiten Prozedur `AddButton` ebenfalls verwendet werden kann, muss sie oberhalb der ersten Prozedur deklariert werden.

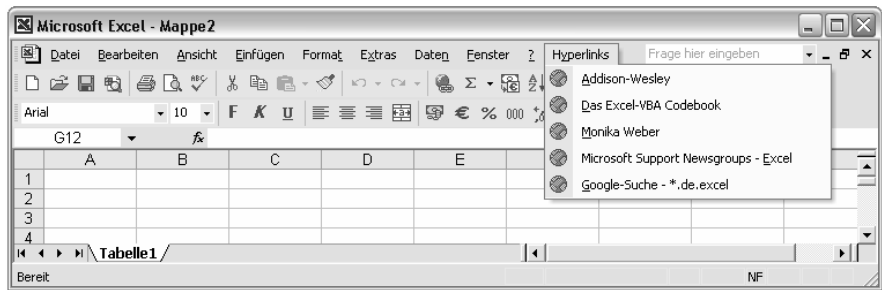


Abbildung 157: Neues Dropdown-Menü mit Hyperlinks

Private mCmdBarPopUp As CommandBarPopup

Das der Variablen vorangestellte m stellt ein Präfix für »Modul« dar. Es signalisiert, dass die Variable mit der Private-Anweisung deklariert wurde. Variablen vom Typ Private stehen nur in dem Modul zur Verfügung, in dem sie deklariert wurden.

Hinweis

Deklariieren Sie Variablen global, um sie in allen Modulen und Klassen des Projektes zur Verfügung zu stellen. Durch die Anweisung
Dim gVarName As Datentyp
oder
Public gVarName As Datentyp
oberhalb der ersten Prozedur werden Variablen global in einem Modul deklariert. Um diese Variablen im Code leichter zu erkennen, erhalten sie das Präfix g.

Nachdem das neue Menü erstellt und über die Caption-Eigenschaft der Beschriftungstext H&yperlinks festgelegt wurde, werden die Schaltflächen mit Hyperlinks im DropDown-Menü erstellt. In unserem Beispiel werden dem Menü fünf Schaltflächen vom Typ msoControlButton hinzugefügt. Man könnte nun in der Prozedur fünfmal den gleichen Codeblock mit den individuellen Angaben hintereinander schreiben. Der folgende Codeblock unterscheidet sich jeweils nur in den Caption- und TooltipText-Angaben.

```
Set cmdBarBtn = mCmdBarPopUp.Controls.Add( _
    Type:=msoControlButton)
With cmdBarBtn
    .Style = msoButtonIconAndCaption
    .FaceId = 610
    .Caption = "Beschriftungstext"
    .TooltipText = "URL der Webseite"
    .HyperlinkType = msoCommandBarButtonHyperlinkOpen
End With
```

Für jede weitere Schaltfläche müssten Sie den kompletten Block wiederholt in die Prozedur einfügen. Diese mehrfache Wiederholung verbraucht unnötig Speicherplatz in einem Modul und der Überblick geht schnell verloren. Die Erstellung einer neuen Schaltfläche wird aus vorgenannten Gründen in der Prozedur AddButton ausgelagert. Beim Aufruf der Prozedur AddButton müssen

- Grundlagen
- Allgemein
- Datum und Zeit
- Diagramme
- Steuerelemente
- Benutzereingaben
- Objektmodelle
- Diagramme
- Ereignisse
- User-Formen
- Web/Mail
- Extern
- Gemischtes
- Spezial

lediglich der Beschriftungstext und die Adresse der Webseite als Argumente an die Parameter `strCaption` und `strTooltip` übergeben werden.

Beim Schreiben der Anweisung zeigt der Editor die Parameter der Funktion an. Um nachträglich die Parameter abzulesen, platzieren Sie den Mauszeiger innerhalb der Aufrufzeile und wählen im Kontextmenü den Eintrag `QUICKINFO` oder `PARAMETERINFO`.

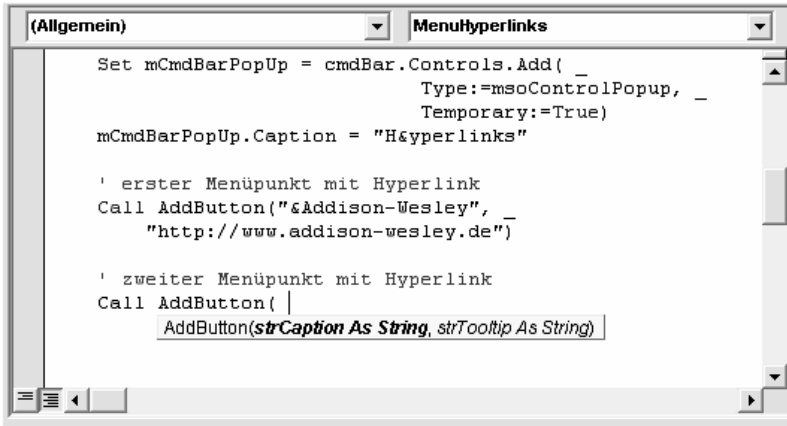


Abbildung 158: Syntax mit Parametern der AddButton-Prozedur

```

'=====
' Auf CD      Buchdaten\Beispiele\06_Befehlsleisten\
' Dateiname   06_10_Hyperlink.xls
' Modul       mdl_01_MenuHyperlinks
'=====

```

```
Private mCmdBarPopUp As CommandBarPopup
```

```
Sub MenuHyperlinks()
```

```
    Dim cmdBar As CommandBar
```

```
    Set cmdBar = CommandBars.ActiveMenuBar
```

```
    On Error Resume Next
```

```
    cmdBar.Controls("Hyperlinks").Delete
```

```
    On Error GoTo 0
```

```
    ' Menü »Hyperlinks« erstellen
```

```
    Set mCmdBarPopUp = cmdBar.Controls.Add( _
        Type:=msoControlPopup, _
        Temporary:=True)
    mCmdBarPopUp.Caption = "H&yperlinks"
```

```
    ' erster Menüpunkt mit Hyperlink
```

```
    Call AddButton("&Addison-Wesley", _
        "http://www.addison-wesley.de")
```

```
' zweiter Menüpunkt mit Hyperlink
Call AddButton("&Das Excel-VBA Codebook", _
               "http://excel.codebooks.de")

' dritter Menüpunkt mit Hyperlink
Call AddButton("&Monika Weber", _
               "http://www.jumper.ch")

' vierter Menüpunkt mit Hyperlink
Call AddButton("Microsoft Support Newsgroups - &Excel", _
               "http://support.microsoft.com/newsgroups/default.aspx?" & _
               "ICP=GSS3NewsGroup=microsoft.public.de.excel" & _
               "SLCID=DE&sd=GN&id=fh;DE;NEWSGROUPS")

' fünfter Menüpunkt mit Hyperlink
Call AddButton("&Google-Suche - *.de.excel", _
               "http://www.google.de/advanced_group_search?hl=de")
End Sub

' Prozedur zum Erstellen der Schaltflächen im Dropdown-Menü
Private Sub AddButton(strCaption As String, strTooltip As String)
    Dim cmdBarBtn As CommandBarButton

    Set cmdBarBtn = mCmdBarPopUp.Controls.Add( _
        Type:=msoControlButton)

    With cmdBarBtn
        .Style = msoButtonIconAndCaption
        .FaceId = 610
        .Caption = strCaption
        .TooltipText = strTooltip
        .HyperlinkType = msoCommandBarButtonHyperlinkOpen
    End With
End Sub
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Errei-
gnisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

208 Schaltfläche einem Menü hinzufügen

In diesem Beispiel wird dem Menü EINFÜGEN (ID:=30005) der ARBEITSBLATT-MENÜLEISTE Application.CommandBars(1) an der zehnten Position ein Befehlsleisten-Steuerelement temporär hinzugefügt. Bei jedem Klick auf das Steuerelement MY COMMENT wird die Prozedur MyComment gestartet. In der aktiven Zelle wird ein Kommentar mit der aktuellen Datum- und Zeitangabe und dem Zelleninhalt erstellt.

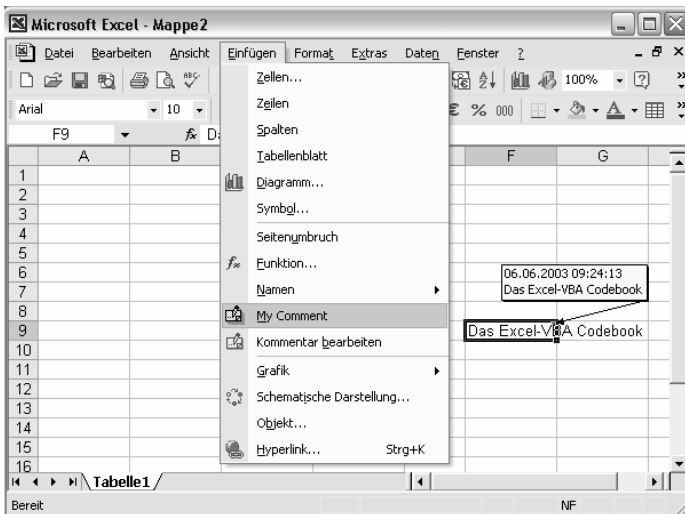


Abbildung 159: Neue Befehlsschaltfläche in Menü einfügen

```
' =====
' Auf CD      Buchdaten\Beispiele\06_Befehlsleisten\
' Dateiname   06_11_MyComment.xls
' Modul       mdl_01_AddMyComment
' =====
```

```
Sub AddMyComment()
    Dim cmdPopUp As CommandBarPopup
    Dim cmdButton As CommandBarButton

    Set cmdPopUp = Application.CommandBars(1) _
        .FindControl(ID:=30005)

    Call CmdButtonDelete

    Set cmdButton = cmdPopUp.Controls.Add _
        (Before:=11, _
        Temporary:=True)

    With cmdButton
        .Style = msoButtonIconAndCaption
        .FaceId = 2056
        .Caption = "&My Comment"
        .OnAction = "MyComment"
    End With
End Sub

Sub CmdButtonDelete()
    On Error Resume Next
    Application.CommandBars(1).FindControl(ID:=30005) _
        .Controls("My Comment").Delete
End Sub
```

Wenn Sie auf die Befehlsschaltfläche MY COMMENT im Menü EINFÜGEN klicken, wird die hinterlegte Prozedur `MyComment` aufgerufen. In dieser Prozedur wird die aktive Zelle bearbeitet. Wurde ein Bereich aus mehreren Zellen selektiert, wird die aktive, weiß hinterlegte Zelle angesprochen. Das Makro wird beendet, wenn die Zelle keinen Inhalt aufweist. Enthält diese Zelle bereits einen Kommentar, muss dieser zuerst gelöscht werden, da Excel sonst eine Fehlermeldung ausgeben würde.

Anschließend wird mittels der `AddComment`-Methode dem Bereich ein neuer Kommentar hinzugefügt. Der Text setzt sich aus dem Wert der `Now`-Funktion, einem Zeilenumbruch `Chr(10)` und dem Wert der aktiven Zelle `.Value` zusammen. Durch die Verwendung der `AutoSize`-Eigenschaft wird die Größe des Kommentarrahmens automatisch dem zugehörigen Text angepasst und zum Schluss der Kommentar über die `Visible`-Eigenschaft angezeigt.

```
'=====
' Auf CD      Buchdaten\Beispiele\06_Befehlsleisten\
' Dateiname   06_11_MyComment.xls
' Modul       mdl_01_AddMyComment
'=====

Sub MyComment()
    Dim com As Comment

    With ActiveCell
        If .Value = "" Then Exit Sub

        ' Wenn Zelle Kommentar enthält, diesen löschen
        If Not .Comment Is Nothing Then .Comment.Delete

        .AddComment (Now & Chr(10) & .Value)
        .Comment.Shape.TextFrame.AutoSize = True
        .Comment.Visible = True
    End With
End Sub
```

209 Tabellenreiter über Dropdownfeld färben

Die Farbe eines Registerreiters können Sie entweder über das Menü **FORMAT | BLATT | REGISTERFARBE** oder über das Kontextmenü der Registerreiter **REGISTERFARBE** wechseln. In diesem Beispiel zeigen wir Ihnen, wie Sie über ein DropDown-Feld bequem die Farben der Registerreiter ändern können. Beim Öffnen der Beispielmappe wird im `Workbook_Activate`-Ereignis die Prozedur `AddTabColorIndex` zur Erstellung der neuen Symbolleiste `MY TABBAR` aufgerufen. Da diese Leiste temporär erstellt wird, erübrigt sich ein ausdrückliches Löschen der Leiste beim Beenden der Excel-Anwendung. Beim Deaktivieren der Mappe durch den Wechsel in eine andere geöffnete Excel-Arbeitsmappe und auch beim Beenden der Mappe wird die neu erstellte Befehlsleiste in `Workbook_Deactivate` gelöscht.

```
'=====
' Auf CD      Buchdaten\Beispiele\06_Befehlsleisten\
' Dateiname   06_12_TabColorIndex.xls
' Ereignis    DieseArbeitsmappe
'=====
```

Grundlagen

Allgemein

Datum
Zeit

Formeln
Funktionen

Steuerung
Elemente

Befehls
leisten

Objekt

Diagramm

Ereignisse

User-
Form

Web/
Mail

Extern

Gemischtes

Spezial

```

Private Sub Workbook_Activate()
    AddTabColorIndex
End Sub

Private Sub Workbook_Deactivate()
    On Error Resume Next
    Application.CommandBars("My TabBar").Delete
End Sub

```

Die benutzerdefinierte Symbolleiste MY TABBAR wird über die `Position`-Eigenschaft an den unteren Seitenrand der Excel-Anwendung positioniert. Die `Left`-Eigenschaft gibt den Pixelabstand vom linken Seitenrand zum Beginn der Leiste an. Der Leiste werden drei Befehlsschaltflächen hinzugefügt.

Das erste Control vom Typ `msoControlButton` ist eine einfache Schaltfläche und weist die Darstellungsart `msoButtonIconAndCaption` auf. Dadurch können ein Schaltflächensymbol `Icon` und ein Beschriftungstext `Caption` angezeigt werden. Die Auswirkung beim Klick auf diese Schaltfläche wird dem Anwender über die `TooltipText`-Eigenschaft vermittelt. Hier wird ein beschreibender Text hinterlegt.

Die `Tag`-Eigenschaft gibt Informationen zum Befehlsleisten-Steuerelement zurück oder legt sie fest. Dies können z.B. Daten sein, die als Argument in Prozeduren verwendet werden können, oder Informationen, die das Steuerelement kennzeichnen. Wir verwenden die `Tag`-Eigenschaft, um beim Klick auf diese Schaltfläche wechselnde Informationen darin zu speichern und in weiteren Prozeduren auszulesen. Zu Beginn weist die erste Schaltfläche die Beschriftung AKTIVER REGISTERREITER auf. Änderungen der Farbe wirken sich dementsprechend nur auf das Tabellenregister des aktiven Tabellenblattes aus (siehe Abbildung 160). Beim Klick auf die erste Schaltfläche wird die Prozedur `AllOrActiveSheet` aufgerufen, welche über die `OnAction`-Eigenschaft hinterlegt ist. Die `State`-Eigenschaft wird zu Beginn auf `msoButtonUp` festgelegt. Die Schaltfläche wird dadurch als »nicht gedrückt« dargestellt.

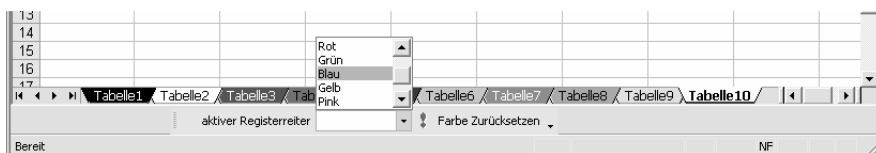


Abbildung 160: Register des aktiven Blattes wird blau gefärbt

Die zweite Schaltfläche ist vom Typ `msoControlComboBox` und stellt ein Kombinationsfeld-Steuerelement dar. Über die `DropDownLines`-Eigenschaft wird die Anzahl der sichtbaren Zeilen im angegebenen `DropDown`-Feld festgelegt. Die `DropDownWidth`-Eigenschaft gibt die Breite (in Pixel) für das angegebene Element an.

Die `AddItem`-Methode fügt dem `DropDown`-Feld Listenelemente hinzu. Dabei wird zum einen der anzuzeigende Text festgelegt. Die Angabe des Textes ist zwingend notwendig. Das zweite optionale Argument `Index` gibt die Position des Elements in der Liste an. Wenn Sie dieses Argument auslassen, wird das Element am Ende der Liste hinzugefügt. Im Grunde könnte in diesem Beispiel auf die `Index`-Angabe verzichtet werden, da die Elemente in aufgeführter Reihenfolge fortlaufend angezeigt werden sollen. Die Angabe dient lediglich als Hinweis auf die `ColorIndex`-Eigenschaft. Diese ist in unserem Beispiel identisch mit der `ListIndex`-Eigenschaft, welche die

Indexnummer des markierten Elements im Listenbereich des Kombinationsfeld-Steuerelements der Befehlsleiste zurückgibt. Wenn die Farben der Standardpalette nicht verändert wurden, hat die Farbe Schwarz den Farbindex 1, Weiß den Farbindex 2 usw.

Die `ListHeaderCount`-Eigenschaft wird verwendet, um eine Trennlinie zwischen zwei Listenelementen zu erstellen. Der angegebene Wert gibt an, wie viele Listenelemente über der Trennlinie angezeigt werden. In unserem Beispiel wird für die `ListHeaderCount`-Eigenschaft ein Wert von `-1` angegeben. Dieser zeigt an, dass im Kombinationsfeld-Steuerelement keine Trennlinie vorhanden ist.

Für das dritte Steuerelement der benutzerdefinierten Symbolleiste `MY TABBAR` wird wieder eine einfache Schaltfläche verwendet. Auf die Funktionsweisen der bei Klick auf die Schaltflächen aufgerufenen Prozeduren gehen wir weiter unten ein. Nachfolgend sehen Sie zunächst die Prozedur zur Erstellung der Symbolleiste.

```
' =====
' Auf CD      Buchdaten\Beispiele\06_Befehlsleisten\
' Dateiname  06_12_TabColorIndex.xls
' Modul      mdl_01_AddTabColorIndex
' =====

Sub AddTabColorIndex()
    Dim cmdBar As CommandBar
    Dim cmdBarCbo As CommandBarComboBox
    Dim cmdBarBtn As CommandBarButton

    ' Evtl. vorhandene Leiste löschen
    On Error Resume Next
    CommandBars("My TabBar").Delete
    On Error GoTo 0

    ' Neue Symbolleiste am unteren Seitenrand erstellen
    Set cmdBar = CommandBars.Add( _
        Name:="My TabBar", _
        Position:=msoBarBottom, _
        Temporary:=True)

    With cmdBar
        .Left = 150
        .Visible = True
    End With

    ' erste Schaltfläche erstellen
    Set cmdBarBtn = cmdBar.Controls.Add(Type:=msoControlButton)
    With cmdBarBtn
        .Style = msoButtonIconAndCaption
        .FaceId = 1
        .Caption = "aktiver Registerreiter"
        .TooltipText = "Wechsel zwischen aktivem- " & _
            "und allen Registerreitern"
        .Tag = "ActiveSh"
        .State = msoButtonUp
        .OnAction = "AllOrActiveSheet"
    End With
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

```

' Dropdown-Menü erstellen
Set cmdBarCbo = cmdBar.Controls.Add(Type:=msoControlComboBox)
With cmdBarCbo
    .Caption = "Register-Farben"
    .ToolTipText = "Farbe auswählen"
    .DropDownLines = 5
    .DropDownWidth = 70
    .AddItem "Schwarz", Index:=1
    .AddItem "Weiß", Index:=2
    .AddItem "Rot", Index:=3
    .AddItem "Grün", Index:=4
    .AddItem "Blau", Index:=5
    .AddItem "Gelb", Index:=6
    .AddItem "Pink", Index:=7
    .AddItem "Türkis", Index:=8
    .OnAction = "PaintTabs"
    .ListHeaderCount = -1
End With

' dritte Schaltfläche erstellen
Set cmdBarBtn = cmdBar.Controls.Add(Type:=msoControlButton)
With cmdBarBtn
    .Style = msoButtonIconAndCaption
    .FaceId = 459
    .Caption = " Farbe Zurücksetzen"
    .ToolTipText = "Standardfarbe (grau) setzen"
    .OnAction = "ResetColor"
End With
End Sub

```

Klicken Sie in der neuen Symbolleiste auf die erste Schaltfläche AKTIVER REGISTERREITER, wird die Prozedur `AllOrActiveSheet` aufgerufen. Die Schaltfläche ist als Umschaltfläche zu betrachten. Da ein `ToggleButton` als Befehlsleisten-Steuerelement nicht zur Verfügung steht, wird der Anzeigestatus einer gedrückten bzw. nicht gedrückten Schaltfläche über die `State`-Eigenschaft festgelegt. In dieser Prozedur werden vier Eigenschaften des angeklickten Steuerelementes verändert, oder genauer gesagt: gewechselt. Am Beispiel der `State`-Eigenschaft verdeutlichen wir die Funktionsweise der für jede Eigenschaft angewandten `IIf`-Funktion.

Die `IIf`-Funktion gibt einen von zwei Teilen, abhängig von der Auswertung eines Ausdrucks, zurück. Die Syntax lautet:

```
Objekt/Variable = IIf(expr, truepart, falsepart)
```

Geben Sie für `expr` einen auszuwertenden Ausdruck an, dessen Ergebnis Wahr oder Falsch sein kann. Ist der Ausdruck Wahr, wird der Wert des ersten Argumentes `truepart` zurückgegeben. Ist der Wahrheitswert Falsch, wird `falsepart` an das angegebene Objekt oder die Variable zurückgegeben.

```

' =====
' Auf CD      Buchdaten\Beispiele\06_Befehlsleisten\
' Dateiname   06_12_TabColorIndex.xls
' Modul       mdl_02_AllOrActiveSheet

```

```
' =====  
  
Sub IIFTest()  
    Dim intWert As Integer  
    intWert = 5  
  
    MsgBox intWert & _  
        " ist " & _  
        IIf(intWert > 10, "größer", "kleiner") & _  
        " 10"  
  
End Sub
```



Abbildung 161: Meldung mit integrierter Auswertung

Hinweis

IIf wertet immer sowohl den Teil truepart als auch den Teil falsepart aus, auch dann, wenn nur einer von beiden Teilen zurückgegeben wird. Aus diesem Grund kann es zu unerwünschten Nebeneffekten kommen. Wenn z.B. die Auswertung von falsepart zu einem Fehler aufgrund einer Division durch Null führt, tritt ein Fehler auch dann auf, wenn expr den Wert True hat.

In unseren Beispielen ermitteln wir den Wahrheitswert eines vorgegebenen Textausdruckes. Dieser kann nur True oder False sein und wird deswegen zu keinem Fehler führen.

Die Codezeile

```
.State = IIf(.State = msoButtonUp, msoButtonDown, msoButtonUp)
```

wechselt den Status der State-Eigenschaft und kann ebenso in einer If...Then...Else-Anweisung programmiert werden, das Ergebnis ist identisch:

```
If .State = msoButtonUp Then  
    .State = msoButtonDown  
Else  
    .State = msoButtonUp  
End If
```

Es obliegt nun dem Programmierer, für welche Methode er sich entscheidet. In folgender Prozedur werden die angegebenen Eigenschaften entsprechend dem Ergebnis aus den nachstehenden IIf-Funktionen eingestellt.

```
' =====  
' Auf CD    Buchdaten\Beispiele\06_Befehlsleisten\  
' Dateiname 06_12_TabColorIndex.xls
```

Grundlagen

Allgemein

Datum
Zeit

Formeln

Steuerelemente

Befehlsleisten

Objekt

Diagramm

Ereignisse

User-Form

Web/
Mail

Extern

Gemischtes

Spezial

```
' Modul      mdl_02_AllOrActiveSheet
' =====

Sub AllOrActiveSheet()
    With Application.CommandBars("My TabBar") _
        .Controls(1)

        .State = IIf(.State = msoButtonUp, msoButtonDown, msoButtonUp)

        .FaceId = IIf(.FaceId = 1, 608, 1)

        .Caption = IIf(.Caption = "aktiver Registerreiter", _
            "alle Registerreiter", _
            "aktiver Registerreiter")

        .Tag = IIf(.Tag = "AllSh", "ActiveSh", "AllSh")

    End With
End Sub
```

Klicken Sie in der benutzerdefinierten Symbolleiste MY TABBAR auf eine Schaltfläche des aufgeklappten DropDown-Feldes, dann wird die Prozedur `PaintTabs` aufgerufen. Es muss ermittelt werden, ob nur der Registerreiter des aktiven Tabellenblattes oder alle Registerreiter mit einer bestimmten Farbe versehen werden sollen. Diese Informationen werden aus der `Tag`-Eigenschaft des ersten Steuerelementes und der `ListIndex`-Eigenschaft des zweiten Steuerelementes (DropDown-Feld) ausgelesen. In einer `If...Then...Else`-Anweisung werden diese Informationen umgesetzt und der/die Registerreiter gefärbt.

```
' =====
' Auf CD      Buchdaten\Beispiele\06_Befehlsleisten\
' Dateiname   06_12_TabColorIndex.xls
' Modul       mdl_03_PaintTabs
' =====

Sub PaintTabs()
    Dim strTag As String
    Dim intColor As Integer
    Dim intI As Integer

    strTag = Application.CommandBars("My TabBar") _
        .Controls(1) _
        .Tag

    intColor = Application.CommandBars("My TabBar") _
        .Controls(2) _
        .ListIndex

    ' Anhand der Tag-Informationen »alle« Blattregister einfärben
    If strTag = "AllSh" Then

        For intI = 1 To Sheets.Count
            Sheets(intI).Tab.ColorIndex = intColor
```

```
Next

' bzw. »aktives« Blattregister einfärben
ElseIf strTag = "ActiveSh" Then
    ActiveSheet.Tab.ColorIndex =intColor
End If
End Sub
```

Die dritte Schaltfläche FARBE ZURÜCKSETZEN ruft die Prozedur `ResetColor` auf. Hier wird anhand der `Tag`-Eigenschaft des ersten Steuerelementes festgestellt, von welchem Registerreiter die Farbe zurückgesetzt werden soll. Diese Anweisung entspricht dem Befehl KEINE FARBE im Dialog REGISTERFARBE ÄNDERN. Enthält die `Tag`-Eigenschaft den Wert `AllSh`, werden in einer `For...Next`-Anweisung alle Blätter der Beispieldatei durchlaufen und jeweils die eingestellte Farbe zurückgesetzt. Nur die Farbe des aktiven Arbeitsblattes wird zurückgesetzt, wenn die `Tag`-Eigenschaft des ersten Steuerelementes den Wert `ActiveSh` aufweist.



Abbildung 162: Farben aller Registerreiter werden zurückgesetzt

```
' =====
' Auf CD      Buchdaten\Beispiele\06_Befehlsleisten\
' Dateiname  06_12_TabColorIndex.xls
' Modul      mdl_04_ResetColor
' =====

Sub ResetColor()
    Dim strTag As String
    Dim intI As Integer

    strTag = Application.CommandBars("My TabBar") _
        .Controls(1) _
        .Tag

    ' Anhand der Tag-Informationen »alle« Blattregister
    ' zurücksetzen
    If strTag = "AllSh" Then

        For intI = 1 To Sheets.Count
            Sheets(intI).Tab.ColorIndex = -4142
        Next

        ' bzw. »aktives« Blattregister zurücksetzen
        ElseIf strTag = "ActiveSh" Then
            ActiveSheet.Tab.ColorIndex = -4142
        End If

    End Sub
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

210 Zellschutz im Kontextmenü anzeigen und ändern

Über das Menü **FORMAT | ZELLEN** oder das Zellkontext-Menü **ZELLEN FORMATIEREN** Register **SCHUTZ** können Sie den Schutzstatus der aktiven Zelle ablesen.

In diesem Beispiel wird der Schutzstatus direkt im Kontextmenü der Zelle angezeigt und kann mit einem Klick auf die jeweilige Schaltfläche geändert werden.

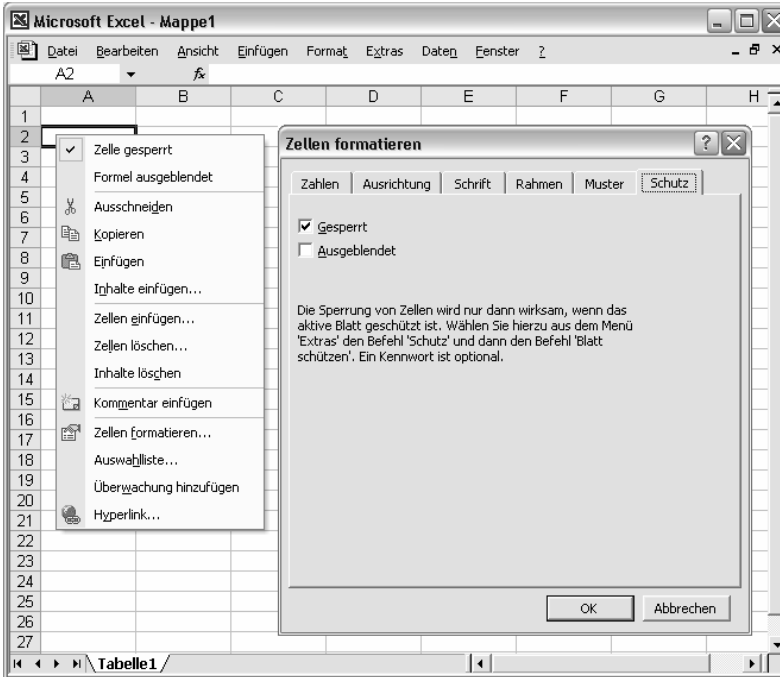


Abbildung 163: Zusätzliche Schaltflächen im Kontextmenü der Zelle

Wenn Sie die Beispieldatei öffnen, wird im `Workbook_Activate`-Ereignis die Prozedur `AddLockedInContext` aufgerufen. Diese fügt der Befehlsleiste `CELL`, also dem Kontextmenü der Zelle, zwei Schaltflächen hinzu. Da diese Schaltflächen temporär erstellt werden, erübrigt sich ein ausdrückliches Löschen der Leiste beim Beenden der Excel-Anwendung. Die neuen Kontexteinträge stehen nur in der Beispielmappe zur Verfügung. Beim Wechsel in eine andere geöffnete Excel-Arbeitsmappe werden sie über das `Workbook_Deactivate`-Ereignis gelöscht.

```
' =====
' Auf CD      Buchdaten\Beispiele\06_Befehlsleisten\
' Dateiname   06_13_ChangeLockedStatus.xls
' Ereignis    DieseArbeitsmappe
' =====
```

```
Private Sub Workbook_Activate()
    AddLockedInContext
End Sub
```

```
Private Sub Workbook_Deactivate()
```

```
On Error Resume Next
With Application.CommandBars("Cell")
    .Controls("Cell Locked").Delete
    .Controls("Formula Hidden").Delete
End With
End Sub
```

Zu Beginn der Prozedur wird ein Verweis von der Befehlsleiste CELL (Zellenkontext-Menü) auf die Variable `cmdBar` gesetzt. Im weiteren Code wird die CELL-Symbolleiste über `cmdBar` angesprochen. Nachdem eventuell vorhandene Schaltflächen gleichen Namens gelöscht wurden, werden die beiden neuen Schaltflächen ZELLE GESPERRT und FORMEL AUSGEBLENDET an den Anfang des Zellen-Kontextmenüs eingefügt. Die Namen der Prozeduren, die bei einem Klick auf die Schaltflächen ausgeführt werden sollen, werden der `OnAction`-Eigenschaft zugewiesen.

Um die beiden eingefügten Steuerelemente auch optisch von den restlichen Schaltflächen des Menüs zu trennen, wird vor dem dritten Element eine Trennlinie mittels der `BeginGroup`-Eigenschaft eingefügt.

```
'=====
' Auf CD      Buchdaten\Beispiele\06_Befehlsleisten\
' Dateiname  06_13_ChangeLockedStatus.xls
' Modul      mdl_01_AddLockedInContext
'=====
```

```
Sub AddLockedInContext()
    Dim cmdBar As CommandBar
    Dim cmdBarCtrl As CommandBarControl

    Set cmdBar = Application.CommandBars("Cell")

    On Error Resume Next
    cmdBar.Controls("Zelle gesperrt").Delete
    cmdBar.Controls("Formel ausgeblendet").Delete
    On Error GoTo 0

    Set cmdBarCtrl = cmdBar.Controls.Add( _
        Type:=msoControlButton, _
        Before:=1, _
        Temporary:=True)

    With cmdBarCtrl
        .Caption = "Zelle gesperrt"
        .OnAction = "LockedCells"
    End With

    Set cmdBarCtrl = cmdBar.Controls.Add( _
        Type:=msoControlButton, _
        Before:=2, _
        Temporary:=True)

    With cmdBarCtrl
        .Caption = "Formel ausgeblendet"
        .OnAction = "FormulaHidden"
    End With
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

```
cmdBar.Controls(3).BeginGroup = True
End Sub
```

Damit zu jeder Zelle im Tabellenblatt deren aktueller Schutzstatus angezeigt wird, setzen wir das `Worksheet_BeforeRightClick`-Ereignis ein. Sobald Sie mit der rechten Maustaste in eine Zelle klicken, werden zunächst die Schaltflächen über die `State`-Eigenschaft aktualisiert. Mit der `msoButtonDown`-Konstante wird der Schaltfläche ein Häkchen hinzugefügt und signalisiert, dass die betreffende Eigenschaft aktiviert ist.

Hinweis

Über die `Locked`-Eigenschaft wird überprüft, ob die selektierte Zelle bzw. in einem markierten Bereich die aktive Zelle gesperrt ist. In der Prozedur wird bewusst nicht die Variable `Target` für diese Kontrolle verwendet. Wenn ein Bereich aus mehreren Zellen markiert wurde, werden alle Zellen des Bereichs und deren Eigenschaften in der Variablen `Target` gespeichert. Wenn diese Zellen unterschiedliche Schutzeigenschaften aufweisen, kann Excel sich nicht zwischen `True` und `False` für die `Locked`-Eigenschaft entscheiden und liefert den Wert `NULL` an `Target` zurück. Das wiederum würde in der `IIf`-Funktion zu einem Fehler führen.

Das Dialogfenster ZELLEN FORMATIEREN (siehe Abbildung 163) zeigt bei unterschiedlichen Zellenformatierungen ein graues Häkchen im jeweiligen Kontrollkästchen an.

```
' =====
' Auf CD      Buchdaten\Beispiele\06_Befehlsleisten\
' Dateiname   06_13_ChangeLockedStatus.xls
' Ereignis    Tabelle1 (Tabelle1)
' =====
```

```
Private Sub Worksheet_BeforeRightClick(ByVal Target As Range, _
                                         Cancel As Boolean)

    With Application.CommandBars("Cell")
        .Controls(1).State = _
            IIf(ActiveCell.Locked, msoButtonDown, msoButtonUp)

        .Controls(2).State = _
            IIf(ActiveCell.FormulaHidden, msoButtonDown, msoButtonUp)
    End With
End Sub
```

Wenn Sie auf eine der eingefügten Schaltflächen klicken, wird die jeweils hinterlegte Prozedur aufgerufen. Die Einstellung der selektierten Zelle bzw. aller Zellen im markierten Bereich `Selection` wird entsprechend der aktiven Zelle gewechselt und die `State`-Eigenschaft der Schaltflächen angepasst.

```
' =====
' Auf CD      Buchdaten\Beispiele\06_Befehlsleisten\
' Dateiname   06_13_ChangeLockedStatus.xls
' Modul       mdl_02_LockedCells
' =====
```



```
Sub LockedCells()  
    Selection.Locked = Not ActiveCell.Locked  
  
    Application.CommandBars("Cell").Controls(1).State = _  
        If(ActiveCell.Locked, msoButtonDown, msoButtonUp)  
End Sub  
  
' =====  
' Auf CD      Buchdaten\Beispiele\06_Befehlsleisten\  
' Dateiname   06_13_ChangeLockedStatus.xls  
' Modul       mdl_03_FormulaHidden  
' =====  
  
Sub FormulaHidden()  
    Selection.FormulaHidden = Not ActiveCell.FormulaHidden  
  
    Application.CommandBars("Cell").Controls(2).State = _  
        If(ActiveCell.FormulaHidden, msoButtonDown, msoButtonUp)  
End Sub
```

211 Kontextmenü durch Kaskaden-Menü ersetzen

Wenn Sie mit der rechten Maustaste auf eine Zelle oder eine Zeilen- bzw. Spaltenüberschrift klicken, wird das jeweilige Kontextmenü angezeigt. Diese Anzeige können Sie unterbinden, indem Sie die Enabled-Eigenschaft der Befehlsleisten auf False setzen. Folgende Prozeduren deaktivieren bzw. aktivieren die gängigsten Kontextmenüs im Tabellenblatt:

```
' =====  
' Auf CD      Buchdaten\Beispiele\06_Befehlsleisten\  
' Dateiname   06_14_KaskadeContext.xls  
' Modul       mdl_01_ContextOnOff  
' =====  
  
' Kontextmenüs deaktivieren  
Sub OnContext()  
    ContextOnOff True  
End Sub  
  
' Kontextmenüs aktivieren  
Sub OffContext()  
    ContextOnOff False  
End Sub  
  
' =====  
' Auf CD      Buchdaten\Beispiele\06_Befehlsleisten\  
' Dateiname   06_14_KaskadeContext.xls  
' Modul       mdl_01_ContextOnOff  
' =====  
  
Sub ContextOnOff(blnContext As Boolean)  
    CommandBars("Cell").Enabled = blnContext
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

```

CommandBars("Row").Enabled = blnContext
CommandBars("Column").Enabled = blnContext
CommandBars("System").Enabled = blnContext
CommandBars("Toolbar List").Enabled = blnContext
CommandBars("Ply").Enabled = blnContext

```

```
End Sub
```

In unserer Beispieldatei sollen die Kontextmenüs nur im Tabellenblatt CONTEXTOFF deaktiviert werden. Schreiben Sie dazu die Prozeduren OnContext und OffContext in die entsprechenden Ereignisprozeduren.

```

' =====
' Auf CD      Buchdaten\Beispiele\06_Befehlsleisten\
' Dateiname   06_14_KaskadeContext.xls
' Ereignis     DieseArbeitsmappe
' =====

Private Sub Workbook_Activate()
    If ActiveSheet.Name = "ContextOff" Then
        ContextOnOff False
    End If
End Sub

Private Sub Workbook_Deactivate()
    If ActiveSheet.Name = "ContextOff" Then
        ContextOnOff True
    End If
End Sub

' =====
' Auf CD      Buchdaten\Beispiele\06_Befehlsleisten\
' Dateiname   06_14_KaskadeContext.xls
' Ereignis     Tabelle1 (ContextOff)
' =====

Private Sub Worksheet_Activate()
    ContextOnOff False
End Sub

Private Sub Worksheet_Deactivate()
    ContextOnOff True
End Sub

```

Erstellen Sie bei einem Rechtsklick im Tabellenblatt CONTEXTOFF über das Worksheet_Before-RightClick-Ereignis ein benutzerdefiniertes Kontextmenü. In diesem Beispiel besteht die erzeugte Befehlsleiste aus einem Kaskaden-Menü. Die einzelnen Stufen der Kaskade werden durch PopUp-Menüs erzeugt. Zeigen Sie das Kontextmenü im Anschluss mit der ShowPopUp-Methode an.

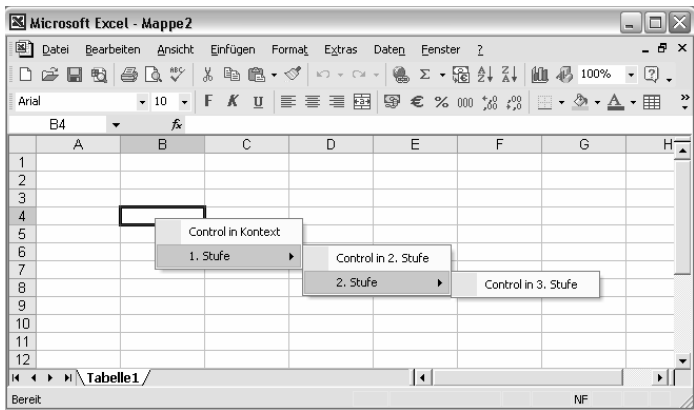


Abbildung 164: Benutzerdefiniertes Kontextmenü ersetzt das Kontextmenü der Zelle

Hinweis

Wenn das auf den Rechtsklick folgende Kontextmenü nicht deaktiviert ist, wird es im Anschluss an das benutzerdefinierte Kontextmenü ebenfalls angezeigt.

```
'=====
' Auf CD      Buchdaten\Beispiele\06_Befehlsleisten\
' Dateiname  06_14_KaskadeContext.xls
' Ereignis   Tabelle1 (ContextOff)
'=====

Private Sub Worksheet_BeforeRightClick(ByVal Target As Range, _
                                         Cancel As Boolean)

    Dim cmdBar As CommandBar
    Dim cmdBarBtn As CommandBarButton
    Dim CtrlPop1 As CommandBarControl
    Dim CtrlPop2 As CommandBarPopup

    On Error Resume Next
    CommandBars("Context").Delete
    On Error GoTo 0

    ' Befehlsleiste Kontextmenü
    Set cmdBar = Application.CommandBars.Add("Context", _
                                              Position:=msoBarPopup, _
                                              Temporary:=True)

    With cmdBar
        ' Schaltfläche in Kontextmenü
        Set cmdBarBtn = cmdBar.Controls.Add(Type:=msoControlButton)
        With cmdBarBtn
            .Caption = "Control in Kontext"
            ' weitere Eigenschaften
        End With

        ' Untermenü 1 in Kontextmenü
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

```

Set CtrlPop1 = cmdBar.Controls.Add(Type:=msoControlPopup)
With CtrlPop1
    .Caption = "1. Stufe"
    ' weitere Eigenschaften
End With

' Schaltfläche im Untermenü 1
Set cmdBarBtn = CtrlPop1.Controls.Add(Type:=msoControlButton)
With cmdBarBtn
    .Caption = "Control in 2. Stufe"
    ' weitere Eigenschaften
End With

' Untermenü 2 im Untermenü 1
Set CtrlPop2 = CtrlPop1.Controls.Add(Type:=msoControlPopup)
With CtrlPop2
    .Caption = "2. Stufe"
    ' weitere Eigenschaften
End With

' Schaltfläche im Untermenü 2
Set cmdBarBtn = CtrlPop2.Controls.Add(Type:=msoControlButton)
With cmdBarBtn
    .Caption = "Control in 3. Stufe"
    ' weitere Eigenschaften
End With
End With

cmdBar.ShowPopup
End Sub

```

212 Welches Kontextmenü wurde aufgerufen?

Das `Worksheet_BeforeRightClick`-Ereignis wird ausgelöst, wenn Sie mit der rechten Maustaste auf einen Spalten- oder Zeilenkopf oder eine bzw. mehrere Zellen klicken. Bevor die Prozedur beendet wird, wird das besagte Spalten-, Zeilen- oder Zellen-Kontextmenü angezeigt. Es gibt kein Objekt, das angibt, welches der drei Kontextmenüs durch den Rechtsklick aufgerufen wird. Um dennoch gezielt auf den Rechtsklick zu reagieren und ein einzelnes Kontextmenü ansprechen zu können, wird anhand der markierten Zellen, Zeilen oder Spalten eine Berechnung durchgeführt. Anhand der Berechnung kann eindeutig bestimmt werden, welches Kontextmenü angezeigt werden soll. Setzen Sie den Parameter `Cancel` auf `True`, damit das Kontextmenü nicht angezeigt wird. Folgende `If...Then...Else`-Anweisung überprüft, ob eine oder mehrere Spalten markiert wurden.

```

If Target.Count / 65536 = Target.Columns.Count Then
    Cancel = True
    ' Aufruf oder Code eines benutzerdefinierten Kontextmenüs
End If

```

Das SPALTEN-Kontextmenü wird im Wahrheitsfall nur in dem Tabellenblatt deaktiviert, in welchem die Prozedur hinterlegt ist. Das ZEILEN- und ZELLEN-Kontextmenü wird beim jeweiligen Rechtsklick weiterhin angezeigt.

In der Beispieldatei wird nach einem Rechtsklick ein Meldungsfenster angezeigt. Dieses enthält die Information, wo der Rechtsklick ausgeführt wurde. Das üblicherweise folgende Kontextmenü wird nicht angezeigt, da die Cancel-Eigenschaft im Anschluss an die If...Then...Else-Anweisung auf True gesetzt wurde.

```
'=====
' Auf CD      Buchdaten\Beispiele\06_Befehlsleisten\
' Dateiname  06_15_CellRowColumnContext.xls
' Ereignis    Tabelle1 (Tabelle1)
'=====

Private Sub Worksheet_BeforeRightClick(ByVal Target As Range, _
                                         Cancel As Boolean)

    Dim strInfo As String

    If Target.Count / 65536 = Target.Columns.Count Then
        strInfo = "Spaltenkopf"
    ElseIf Target.Count / 256 = Target.Rows.Count Then
        strInfo = "Zeilenkopf"
    ElseIf Target.Count / 65536 <> Target.Columns.Count And _
            Target.Count / 256 <> Target.Rows.Count Then
        strInfo = "Zelle "
    End If

    MsgBox strInfo & " wurde angeklickt"

    ' alle Kontextmenüs deaktivieren
    Cancel = True
End Sub
```

213 Symbole über Kontextmenü einfügen

Ab Excel 2002 können Sie Sonderzeichen über das Menü EINFÜGEN | SONDERZEICHEN in eine Zelle einfügen. In vorherigen Excel-Versionen ist dies nur sehr umständlich zu bewerkstelligen. In diesen Versionen können Sie zum Beispiel in Microsoft Word ein Sonderzeichen über das Menü EINFÜGEN | SONDERZEICHEN erstellen, kopieren und dann über die Zwischenablage in Ihre Excel-Datei einfügen. Diese Möglichkeit bietet auch die Zeichentabelle, welche Sie über die Windows-Schaltfläche START | PROGRAMME | ZUBEHÖR | SYSTEMPROGRAMME | ZEICHENTABELLE erreichen. In der Zeichentabelle ersehen Sie zu den ausgewählten Zeichen den entsprechenden vierstelligen Unicode-Zeichensatz. Mit diesem Zeichensatz ist es möglich, ein Zeichen direkt in ein Dokument oder eine Zelle einzufügen. Setzen Sie die Einfügemarke an die Stelle, an der das Sonderzeichen eingefügt werden soll, und halten Sie die **[Alt]**-Taste gedrückt, während Sie über die Tasten der Zehnertastatur die Unicode-Zeichenwerte eingeben.

In Excel verwenden Sie die ZEICHEN-Funktion, um Zahlen in Zeichen umzuwandeln. Die Funktion erwartet eine Zahl zwischen 1 und 255, die das von Ihnen gewünschte Zeichen in der gewählten Schriftart angibt. Das jeweilige Zeichen ist Bestandteil des Zeichensatzes, der auf Ihrem Computer verwendet wird. Die Zahlen von 0 bis 31 entsprechen den nicht darstellbaren Standard-ASCII-Codes. Chr(10) gibt zum Beispiel ein Zeilenvorschubzeichen zurück.

Im Tabellenblatt SONDERZEICHEN der Beispielmappe werden in einer Spalte alle verfügbaren Sonderzeichen der Schriftart Arial über die Zeichen-Funktion dargestellt. Die fortlaufende Angabe

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

des Zahlenwertes erreichen wir durch die Funktion `Zeile()`. Diese gibt die Zeilenzahl der Zeile wieder, in welcher die Formel steht.

Die Formel `=ZEICHEN(ZEILE())` in Zelle A191 liefert das Zeichen `:`.

Die erzeugten Sonderzeichen können Sie kopieren und über das ZELL-KONTEXTMENÜ | INHALTE EINFÜGEN | WERTE in feste Werte umwandeln. Ebenfalls können Sie den ermittelten Zeichencode mit gedrückter `[Alt]`-Taste und Eingabe des Zahlencodes über die Zehnertastatur eingeben. Damit der Unicode-Zeichensatz vervollständigt wird, müssen Sie der dreistelligen Codezahl eine 0 voranstellen. Das Promillezeichen `‰` erzeugen Sie zum Beispiel mit der Tastenkombination `[Alt] + [0137]`.

Die Eingabe von Sonderzeichen lässt sich über ein zusätzliches Menü im Zellkontextmenü wesentlich vereinfachen. Klicken Sie auf eine der Schaltflächen, dann wird das auf der Schaltfläche dargestellte Zeichen an der aktuellen Cursorposition eingefügt (siehe Abbildung 165).

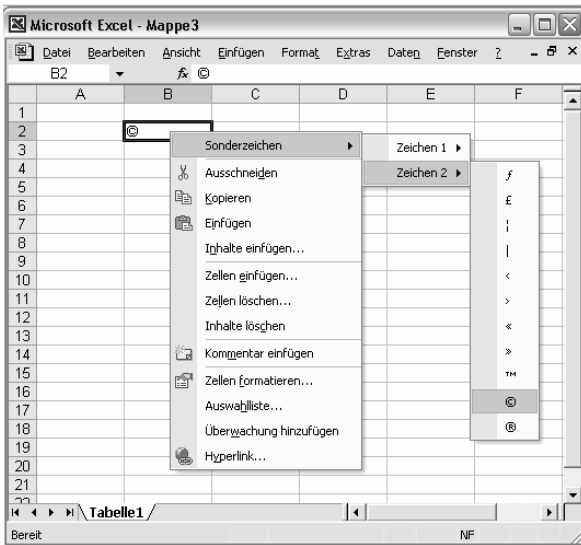


Abbildung 165: Sonderzeichen über Kontextmenü einfügen

Ermitteln Sie zunächst im Tabellenblatt SONDERZEICHEN der Beispieldatei die Zahlencodes der Zeichen, welche Sie im Kontextmenü anbieten wollen. Diese werden in den beiden Arrays `varSymbol1` und `varSymbol2` hinterlegt.

In diesem Beispiel verwenden wir die `Find`-Methode, um das eventuell bereits vorhandene Steuerelement SONDERZEICHEN zu löschen. Zur Identifizierung wird die `Tag`-Eigenschaft der Schaltfläche herangezogen. Diese gibt Informationen zum Befehlsleisten-Steuerelement zurück oder legt sie fest. Dies können z. B. Daten sein, die als Argument in Prozeduren verwendet werden können, oder Informationen, die das Steuerelement kennzeichnen. Der `Tag`-Eigenschaft der zu löschenden Schaltfläche wurde der Wert `SpecialTag` zugewiesen. Dadurch ist das Steuerelement in der `Find`-Methode eindeutig zu erkennen.

Bevor dem Zellkontextmenü ein neues Menü hinzugefügt wird, erstellen Sie über die `BeginGroup`-Eigenschaft vor dem ersten Element eine neue Gruppe. Diese erkennen Sie an einer Trennlinie zwischen den Elementen auf der Befehlsleiste. Anschließend wird an erster Stelle des Zellenkontextmenüs das neue Menü SONDERZEICHEN vom Typ `msoControlPopup` eingefügt. Diesem wieder-

rum werden zwei einfache Schaltflächen ZEICHEN2 und ZEICHEN1 vom Typ `msoControlButton` zugewiesen. Die Abfolge der Erstellung erfolgt in umgekehrter Reihenfolge, da das neue Element jeweils an erster Stelle im Kontextmenü eingefügt wird und somit die anderen Elemente eine Position nach unten verschiebt.

Die in den Arrays aufgeführten Zahlencodes werden nacheinander in einer `For...Next`-Anweisung den Schaltflächen ZEICHEN2 und ZEICHEN1 zugewiesen. Die Beschriftung der Schaltflächen wird über die `Caption`-Eigenschaft vorgenommen.

```
.Caption = Chr(Val(varSymbol2(intI)))
```

Diese Syntax setzt sich aus folgenden Teilen zusammen:

- ▶ `varSymbol2(intI)`
Gibt den fortlaufenden Zahlencode des zweiten Arrays `varSymbol2` als String zurück
- ▶ `Val(varSymbol2(intI))`
Über die `Val`-Funktion wird der – in String-Form – enthaltene Zahlencode in einen numerischen Wert umgewandelt
- ▶ `Chr(Val(varSymbol2(intI)))`
die `Chr`-Funktion wandelt den Zahlenwert in das entsprechende Zeichen, das dem angegebenen Zeichencode zugeordnet ist, um

Den `Tag`-Eigenschaften der SONDERZEICHEN-Schaltflächen wird ebenfalls der Zahlencode zugewiesen. Dadurch kann das aufgerufene Makro `InsertSymbol` eindeutig bestimmen, welche Schaltfläche den Prozeduraufruf verursacht hat.

```
'=====
' Auf CD      Buchdaten\Beispiele\06_Befehlsleisten\
' Dateiname   06_17_InsertSymbol.xls
' Modul       mdl_01_SymbolContext
'=====

Sub ContextInsertSymbol()
    Dim cmdBar As CommandBar
    Dim cmdBarCtrl As CommandBarControl
    Dim ctrlPopUp As CommandBarControl
    Dim cmdBarBtn As CommandBarButton
    Dim varSymbol1() As Variant
    Dim varSymbol2() As Variant
    Dim intI As Integer

    ' Hier die Zeichen ergänzen
    varSymbol1 = Array("137", "177", "178", "179", _
                       "188", "189", "190", "216")

    varSymbol2 = Array("131", "163", "166", "124", _
                       "139", "155", "171", "187", _
                       "153", "169", "174")

    On Error Resume Next
    CommandBars("Cell").FindControl(Tag:="SpecialTag").Delete
```

Grundlagen

Allgemein

Datum
Zeit

tungen

Steuer
elemente

Befehl
leisten

Objekt

Diagramm

Ereignisse

User-
Form

Web/
Mail

Extern

Gemischtes

Spezial

```

On Error GoTo 0

Set cmdBar = CommandBars("Cell")
' Vor dem ersten Steuerelement des Zell-Kontextmenüs
' eine neue Gruppe beginnen
cmdBar.Controls(1).BeginGroup = True

'Flüchtiges Menü im Zell-Kontextmenü erstellen
Set cmdBarCtrl = cmdBar.Controls.Add( _
    Type:=msoControlPopup, _
    Before:=1, _
    Temporary:=True)

With cmdBarCtrl
    .Caption = "Sonderzeichen"
    .Tag = "SpecialTag"
End With

' Zweites Untermenü anlegen
Set ctrlPopUp = cmdBarCtrl.Controls.Add( _
    Type:=msoControlPopup, _
    Before:=1)
ctrlPopUp.Caption = "Zeichen 2"

For intI = 0 To UBound(varSymbol2)
    Set cmdBarBtn = ctrlPopUp.Controls.Add(msoControlButton)
    With cmdBarBtn
        .Caption = Chr(Val(varSymbol2(intI)))
        .Tag = varSymbol2(intI)
        .OnAction = "InsertSymbol"
    End With
Next intI

' Erstes Untermenü anlegen
Set ctrlPopUp = cmdBarCtrl.Controls.Add( _
    Type:=msoControlPopup, _
    Before:=1)
ctrlPopUp.Caption = "Zeichen 1"

For intI = 0 To UBound(varSymbol1)
    Set cmdBarBtn = ctrlPopUp.Controls.Add(msoControlButton)
    With cmdBarBtn
        .Caption = Chr(Val(varSymbol1(intI)))
        .Tag = varSymbol1(intI)
        .OnAction = "InsertSymbol"
    End With
Next intI
End Sub

```

Wenn Sie auf eine SONDERZEICHEN-Schaltfläche klicken, wird die Prozedur `InsertSymbol` aufgerufen. Jetzt gilt es nur noch zu ermitteln, welches Sonderzeichen an die aktuelle Cursorposition eingefügt werden soll. Hierzu verwenden Sie die `ActionControl`-Eigenschaft. Sie gibt das `CommandBarControl`-Objekt zurück, dessen `OnAction`-Eigenschaft auf diese Prozedur eingestellt ist.

Der in der Tag-Eigenschaft hinterlegte Zahlencode wird über die Chr-Funktion in das entsprechende Sonderzeichen umgewandelt und im Tabellenblatt eingefügt.

```
'=====
' Auf CD      Buchdaten\Beispiele\06_Befehlsleisten\
' Dateiname  06_17_InsertSymbol.xls
' Modul      mdl_01_SymbolContext
'=====

Sub InsertSymbol()
    With ActiveCell
        .Value = .Value & _
            Chr(Val(Application.CommandBars.ActionControl.Tag))
    End With
End Sub
```

Wenn Sie die Beispieldatei öffnen, wird über das Workbook_Activate-Ereignis die Prozedur ContextInsertSymbol aufgerufen. Diese erstellt das Menü SONDERZEICHEN mit den beiden Untermenüs ZEICHEN1 und ZEICHEN2 im Kontextmenü der Zelle.

```
'=====
' Auf CD      Buchdaten\Beispiele\06_Befehlsleisten\
' Dateiname  06_17_InsertSymbol.xls
' Ereignis    DieseArbeitsmappe
'=====

Private Sub Workbook_Activate()
    ContextInsertSymbol
End Sub
```

Wenn Sie von der Beispielmappe zu einer anderen Excel-Arbeitsmappe wechseln oder die Beispielmappe beenden, wird das Workbook_Deactivate-Ereignis aufgerufen. Über die FindControl-Methode wird das Menü SONDERZEICHEN anhand der Tag-Information identifiziert und gelöscht.

```
'=====
' Auf CD      Buchdaten\Beispiele\06_Befehlsleisten\
' Dateiname  06_17_InsertSymbol.xls
' Ereignis    DieseArbeitsmappe
'=====

Private Sub Workbook_Deactivate()
    On Error Resume Next
    CommandBars("Cell").FindControl(Tag:="SpecialTag").Delete
End Sub
```

214 Alle FacelDs auf einen Blick

Auf der Buch-CD haben wir im Beispielordner der Kategorie *Befehlsleisten* die Datei *06_18_FacelDs.xls* mit allen in MS-Office zur Verfügung stehenden Schaltflächensymbolen mit ihren ID-Nummern hinterlegt. Die Symbole können z.B. auch in Word und Access verwendet werden.

Grundlagen

Allgemein

Datum
Zeit

Formeln
Funktionen

Steuerungselemente

Befehlsleisten

Objekte

Diagramme

Ereignisse

User-Formen

Web/Mail

Extern

Gemischtes

Spezial

Sie können die vorhandenen Symbole in den Symbolleisten zum Beispiel von Hand durch ein anderes Icon austauschen. Wählen Sie dazu ein Symbol aus der Übersicht aus und kopieren Sie es in die Zwischenablage. Öffnen Sie nun den ANPASSEN-Dialog über das Menü EXTRAS | ANPASSEN. Klicken Sie anschließend mit der rechten Maustaste auf die zu ändernde Schaltfläche und wählen Sie im aufgeklappten Kontextmenü den Eintrag SCHALTFLÄCHE EINFÜGEN.

Entnehmen Sie der Übersicht eine FaceId-Nummer, um das dazugehörige Symbol in Ihren VBA-Codes zum Erstellen von einfachen Schaltflächen `msoControlButtons` einzufügen. Die Symbole können auch direkt in Tabellenblätter eingefügt werden. Man kann sie vergrößern und es können Makros zugewiesen werden.

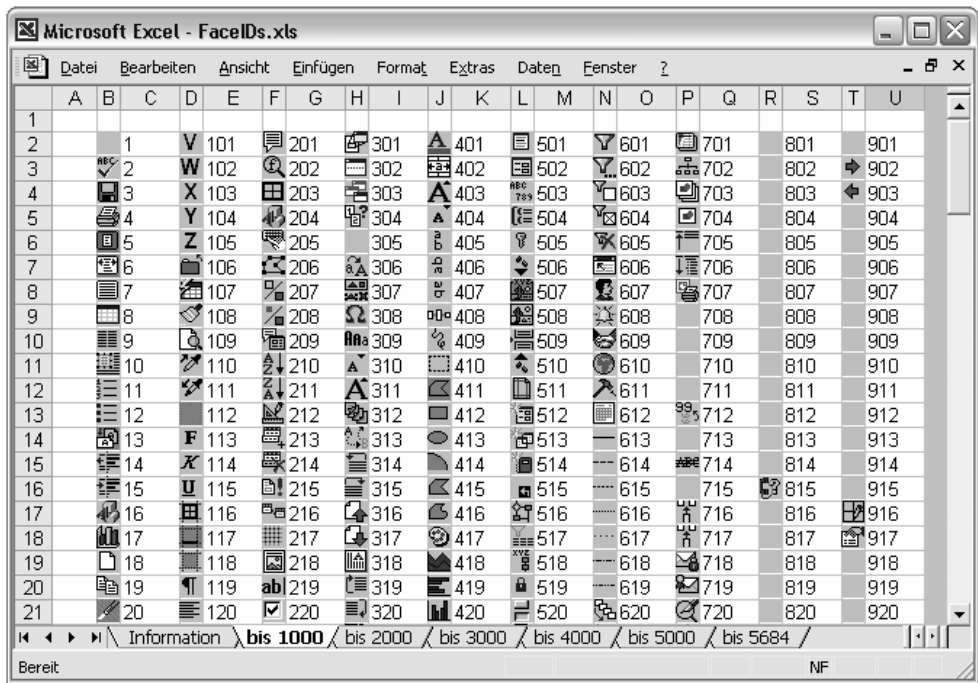


Abbildung 166: Schaltflächensymbole und FaceIDs in MS-Office

Objekte

Bevor wir uns den Rezepten in dieser Kategorie zuwenden, möchten wir kurz einen Überblick darüber geben, was Sie hier erwartet. Diese Kategorie ist unter anderem ein Ausflug in den grafischen Bereich, denn Sie werden Tipps und Tricks erfahren, wie Bilder exakt positioniert in Ihr Tabellenblatt eingefügt werden können. Weitere Schwerpunkte sind Elemente aus der Symbolleiste ZEICHNEN und der Office-Assistent.

Hinweis

Für einige der Beispiele sind Grafiken erforderlich. Sie finden die Grafiken auf der CD im selben Pfad wie die Beispiele selbst.

215 Einen Bereich in eine Grafik umwandeln

Auf den ersten Buchseiten dieses Kapitels werden Sie etwas über den Umgang mit Grafiken und Clipart erfahren.

Hinweis

Beim Arbeiten mit Grafik-Objekten ist es wichtig zu wissen, dass es sich dabei um ein Objekt namens `Picture` handelt. Alternativ kann auch das Objekt `Shape` verwendet werden, dieses bezieht sich jedoch nicht nur auf Grafiken, sondern allgemein auf eingefügte Elemente, wie z. B. auf Steuerelemente (`CommandButton`, `ComboBox` usw.) und `AutoFormen`. Zu beiden Objekten werden Sie in dieser Kategorie Rezepte finden.

Das folgende Beispiel wird Ihnen zeigen, wie Sie einen beliebigen Bereich fotografieren und als Bild in Ihr Tabellenblatt einfügen können. Es wird dabei alles fotografiert, was sich in dem Bereich befindet, den Sie in Ihrem VBA-Code hinterlegen.

Zuerst zeigen wir Ihnen den Weg ohne VBA auf:

1. Markieren Sie den gewünschten Bereich.
2. Klicken Sie mit gehaltener -Taste auf den Menüpunkt BEARBEITEN. Das Drücken der -Taste bewirkt, dass sich im Menü BEARBEITEN einige der verfügbaren Menübefehle ändern.
3. Wählen Sie den Eintrag BILD KOPIEREN aus.



Abbildung 167: Einen Bereich als Bild kopieren

Grundlagen
Allgemein
Datum und Zeit
Diagramme
Diagramm
Ereignisse
User-Form
Web/Mail
Extern
Gemischtes
Spezial

In dem Dialogfenster, das sich nun öffnet (siehe Abbildung 168), können Sie bei DARSTELLUNG auswählen, ob das Bild WIE ANGEZEIGT oder WIE AUSGEDRUCKT fotografiert werden soll. WIE ANGEZEIGT bedeutet, dass das Bild so ausgegeben wird, wie Sie es auf dem Bildschirm sehen können. Wenn Sie WIE AUSGEDRUCKT wählen, hängt das Ergebnis von Ihrem Drucker ab. Sollten Sie keinen Farbdrucker im Einsatz haben, wird das Bild schwarz-weiß angezeigt, genauso wie es in der Seitenansicht zu sehen ist.

Wenn Sie bei DARSTELLUNG die Auswahl WIE ANGEZEIGT treffen, können Sie zudem das FORMAT bestimmen. Zur Auswahl stehen BILD und BITMAP. Wenn Sie BILD auswählen, wird die Grafik im Meta-Dateiformat angezeigt. Eine Meta-Datei kann ohne Verlust an Qualität vergrößert und verkleinert werden. Ein BITMAP wird nur dann korrekt angezeigt, wenn die Auflösung des Bildschirms, auf dem die Grafik angezeigt wird, mit der Auflösung des Bildschirms übereinstimmt, mit dem die Aufnahme gemacht wurde.



Abbildung 168: Gesamter Bereich und ein Foto eines Teils davon

In unserer VBA-Prozedur wird der Bereich A1:C7 fotografiert (CopyPicture). Die Konstanten WIE ANGEZEIGT xlScreen und BILD xlPicture werden verwendet. Danach wird das Bild in die Zelle D10 eingefügt (Paste).

Der Tabelle 47 können Sie die verfügbaren Konstanten zur Methode CopyPicture entnehmen.

CopyPicture-Konstante	Index	Beschreibung
Appearance:=xlScreen	1	Wie angezeigt
Appearance:=xlPrinter	2	Wie ausgedruckt
Format:=xlPicture	-4147	Bild
Format:=xlBitmap	2	Bitmap

Tabelle 47: Konstanten zur Bilderzeugung

' =====	
' Auf CD	Buchdaten\Beispiele\07_Objekte\
' Dateiname	07_01_Pictures.xls
' Tabelle	07_Objects_RangePicture
' Modul	mdl_01_PhotographPicture
' =====	

```
Sub PhotographPicture()  
    With ActiveSheet  
        ' Bereich A1:C5 fotografieren  
        .Range("A1:C7").CopyPicture _  
            Appearance:=xlScreen, _  
            Format:=xlPicture  
  
        ' Bild in Zelle C12 einfügen  
        .Paste _  
            Destination:=ActiveSheet.Range("D10")  
    End With  
End Sub
```

216 Einen Bereich als Grafik exportieren

In der Kategorie »Diagramme« werden Sie erfahren, dass mittels der Methode `Export` ein Diagramm als Grafik, beispielsweise im Format `*.gif`, abgespeichert werden kann. Leider funktioniert diese Methode ausschließlich in Bezug auf Diagramme. Um einen Bereich als Grafik abspeichern zu können, müssen wir uns mit einem Trick behelfen. Was liegt näher, als ein Diagramm zu Hilfe zu nehmen?

Unser Code ist so aufgebaut, dass zuerst ein leeres Diagramm erstellt wird. Ein leeres Diagramm kann basierend auf einer leeren Zelle erzeugt werden. Da in den meisten Fällen davon ausgegangen werden kann, dass die allerletzte Zelle in Ihrem Tabellenblatt, die Zelle IV65536, leer ist, werden wir genau diese Zelle verwenden, um das Diagramm zu erstellen. Da trotz allem nicht ausgeschlossen werden kann, dass die Zelle einen Wert enthält, werden wir diesen in der Variablen `varDummy` zwischenspeichern. Die Zelle wird danach gelöscht. Nach dem Erstellen des Diagramms wird der Wert der Variablen `varDummy` wieder in die Zelle IV65536 zurückgeschrieben. Sie können natürlich auch beliebig auf eine andere leere Zelle Bezug nehmen, gegebenenfalls sogar auf einem anderen Tabellenblatt.

Nach dem Erzeugen des Diagramms werden wir genauso vorgehen, wie im vorangegangenen Rezept beschrieben. Wir erzeugen aus einem Bereich eine Grafik. Diese Grafik wird dann in das zuvor erstellte leere Diagramm eingefügt (`Chart.Paste`).

Da die Größe des Diagramms in der Regel nicht mit der Größe der Grafik übereinstimmt, müssen wir diese beiden Elemente aufeinander abstimmen. Wie übergeben die Höhe und Breite der Grafik an das Diagramm. Da das Diagramm dann noch etwas zu klein ist, um die gesamte Grafik anzuzeigen, geben wir noch sechs Bildpunkte dazu. Das kann abhängig von Ihrer Bildschirmauflösung variieren. Die Grafik wird schließlich durch einen gleichmäßigen Rahmen umgeben. Da die Schaltfläche im Tabellenblatt ebenfalls ein `Shape` ist, genauso wie das Diagramm, müssen wir den Index 2 verwenden. Der Index 1 ist bereits für die Schaltfläche reserviert, da diese ja zuerst eingefügt wurde.

Das Diagramm kann anschließend in ein Grafikformat exportiert werden. In unserem Beispiel verwenden wir `*.gif` als Grafikformat. Denkbar wäre auch `*.jpg` oder `*.png`.

Da sie nun in einer Grafikdatei abgespeichert ist, wird sie auf dem Tabellenblatt nicht mehr benötigt und kann gelöscht werden. Am Ende der Prozedur wird ein Nachrichtenfenster `MsgBox` angezeigt, welches über den erfolgreichen Abschluss der Prozedur informiert. Der Dateiname und der Pfad wurden zu Beginn der Prozedur an die Variable `strPath` übergeben. Der Inhalt dieser Variablen wird ebenfalls im Nachrichtenfenster angezeigt.

Grund-
lagenAllge-
meinDatu
Zeit

tungen

Steuer
elemenBefehl
leisten

Objekt

Dia-
grammEreig-
nisseUser-
FormWeb/
Mail

Extern

Ge-
misch
tes

Specia

Damit von dem ganzen Vorgang nichts zu sehen ist, wird zu Beginn der Bildschirmaktualisierung deaktiviert (ScreenUpdating = False) und am Ende der Prozedur wieder aktiviert. Der Ablauf der Prozedur wird damit zudem beschleunigt.

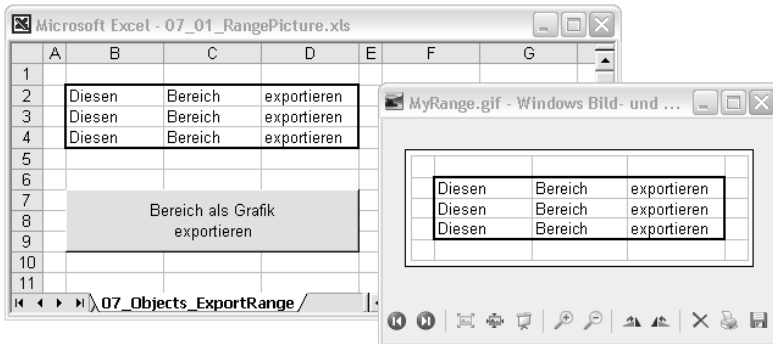


Abbildung 169: Excel im Original und die daraus erzeugte Grafik

```
'=====
' Auf CD      Buchdaten\Beispiele\07_Objekte\
' Dateiname  07_01_Pictures.xls
' Tabelle    07_Objects_ExportRange
' Modul      mdl_02_ExportRange
'=====
```

```
Sub ExportRange()
    Dim strPath As String

    strPath = "C:\MyRange.gif"

    Application.ScreenUpdating = False

    ' Prüfen, ob die Zelle IV65536 leer ist
    If Range("IV65536") <> "" Then
        varDummy = Range("IV65536").Value
        Range("IV65536").Delete
    End If

    ' Leeres Diagramm aus letzter Zelle erstellen
    With Charts.Add
        .SetSourceData Source:=Worksheets(3).Range("IV65536")
        .Location Where:=xlLocationAsObject, Name:=Worksheets(3).Name
    End With

    With ActiveSheet
        ' Bild nach Diagramm kopieren
        .Range("A1:E5").CopyPicture _
            Appearance:=xlScreen, _
            Format:=xlPicture
        .ChartObjects(1).Chart.Paste
    End With
End Sub
```

```
' Höhe und Breite des Fotos an das Diagramm übergeben
With .Shapes(2)
    .Height = Selection.Height + 6
    .Width = Selection.Width + 6
End With

' Das Diagramm als *.gif abspeichern
.ChartObjects(1).Chart.Export (strPath)
End With

' Grafik löschen
ActiveSheet.ChartObjects(1).Delete

MsgBox "Die Grafik wurde erfolgreich erzeugt." & vbCrLf & _
    "Speicherort: " & strPath

' Der Zelle IV65536 den Wert aus varDummy wieder zuweisen
Range("IV65536").Value = varDummy

Application.ScreenUpdating = False
End Sub
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

Hinweis

Bei all den oben genannten Grafikformaten handelt es sich um komprimierte Formate. Die Dateigröße wird auf diese Weise auf ein Minimum reduziert. Damit Sie sich ein Bild davon machen können, worum es sich bei komprimierten Formaten handelt, finden Sie nachfolgend eine kurze Zusammenfassung.

Beim *JPEG* (Joint Photographic Experts Group) werden ähnliche Farbwerte zusammengefasst. Wenn nun zehnmal der Wert 244 enthalten ist, wird dieser Wert nicht zehnmal eingetragen, sondern zusammengefasst als 10x244. Somit wird die Bildqualität natürlich beeinträchtigt. In der Regel sind diese Feinheiten jedoch für das menschliche Auge kaum zu erkennen. Das wiederum ist allerdings vom Komprimierungsgrad abhängig. Je mehr eine Grafik komprimiert wird, desto schlechter die Bildqualität. *JPEG*-Grafiken haben eine Farbtiefe von bis zu 24 Bit. Das entspricht 16,77 Millionen Farben.

GIF (Graphics Interchange Format) ist ein Format, welches durch den Online-Dienst CompuServe entwickelt wurde. Es können nebst komprimierten Grafiken auch animierte Bilder erzeugt werden. Das heißt, es werden mehrere Bilder in einer Datei zusammengefasst und nacheinander abgespielt. *GIFs* werden nach der *LZW*-Methode komprimiert. Das Verfahren wurde nach seinen Entwicklern Lempel, Ziv und Welch benannt. Ein weiteres Plus, nebst den Animationen, ist die Möglichkeit, den Hintergrund einer Grafik transparent darzustellen und sich somit dem Hintergrund anzupassen. Ein Nachteil mag sein, dass nur eine Farbtiefe bis zu 8 Bit erreicht werden kann, was lediglich 256 Farben entspricht.

PNG (Portable Network Graphic), ausgesprochen »Ping«, wurde vom *W3C* (World Wide Web Consortium) entwickelt und als lizenzfreier Standard verabschiedet. *PNG* soll nach und nach *JPEG* und vor allem *GIF* ablösen. *PNG* bietet eine nahezu verlustfreie Komprimierung an.

217 Eine verknüpfte Grafik

Ein sehr interessanter Aspekt ist eine verknüpfte Grafik. Vielleicht fragen Sie sich nun: »Was ist mit einer verknüpften Grafik gemeint«? Wie eine statische Grafik in ein Tabellenblatt eingefügt werden kann, haben Sie im Rezept 215 erfahren. Sie haben jedoch auch die Möglichkeit, eine Grafik einzufügen, die mit einem Zelleninhalt verknüpft ist. Das bedeutet: Wenn sich der Inhalt der Quelle ändert, wird auch die Grafik aktualisiert. Die Quelle kann durchaus auch ein Datenbereich aus einem anderen Tabellenblatt sein. Dort sind beispielsweise Werte enthalten, die aus einer Berechnung resultieren, die sich laufend verändern. Sie haben mit einer verknüpften Grafik die Möglichkeit, diesen Bereich eines anderen Tabellenblattes anzeigen zu lassen. Des Weiteren wäre es denkbar, eine verknüpfte Grafik in einem fixierten Fensterbereich anzeigen zu lassen, so dass Sie beim Scrollen jeweils diesen bestimmten Teil der Tabelle in der Grafik sehen können.

Eine verknüpfte Grafik lässt sich sowohl manuell als auch per VBA erzeugen. Nachfolgend zeigen wir Ihnen zuerst den manuellen Weg auf:

1. Markieren Sie den Bereich, den Sie in einer Grafik verknüpfen möchten.
2. Kopieren Sie den markierten Bereich mit **[Strg] + [C]**.
3. Selektieren Sie eine Zielzelle. Diese kann sich sowohl auf demselben Tabellenblatt als auch auf einem anderen befinden. Die Grafik kann später beliebig verschoben werden.
4. Klicken Sie mit gedrückter **[⇧]**-Taste auf den Menüpunkt **BEARBEITEN** und dann auf den Eintrag **VERKNÜPFTES BILD EINFÜGEN**.

Die Grafik, die Sie nun sehen können, ist mit der Quelle verknüpft. Wenn sich Daten in der Quelle ändern, reflektiert sich dies in der Grafik.

Wenn Sie diese Schritte mit dem Makro-Rekorder aufzeichnen, erkennen Sie, dass es sich bei der eingefügten Grafik (Copy und Paste) um einen Link handelt (`Link:=True`). Mittels VBA-Programmierung haben Sie zudem die Möglichkeit, die Grafik hervorzuheben, indem Sie diese beispielsweise in einer anderen Farbe und mit einem Rahmen darstellen. Genau das werden wir in unserem Beispiel tun. Der Grafik werden ein hellblauer Hintergrund und ein dunkelblauer Rahmen zugewiesen. Die Grafik wird in der Zelle erzeugt, in der sich der Cursor beim Ausführen des Codes befindet.

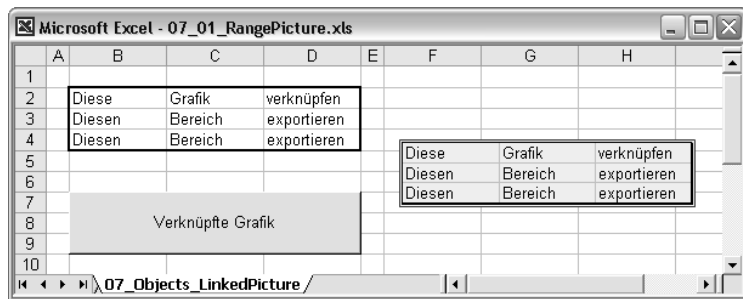


Abbildung 170: Originalbereich und verknüpfte Grafik in hellblauer Farbe

```
' =====
' Auf CD      Buchdaten\Beispiele\07_Objekte\
' Dateiname   07_01_Pictures.xls
' Tabelle     07_Objects_LinkedPicture
' Modul       mdl_03_LinkedRange
```



```
' =====  
  
Sub LinkedPicture()  
    Range("B2:D4").Copy  
  
    With ActiveSheet.Pictures.Paste(Link:=True).ShapeRange  
        ' Blauer Rahmen  
        With .Line  
            .ForeColor.SchemeColor = 12  
            .Visible = msoTrue  
        End With  
  
        ' Hellblaue Füllfarbe  
        With .Fill  
            .ForeColor.SchemeColor = 41  
            .Visible = msoTrue  
            .Solid  
        End With  
    End With  
  
    Application.CutCopyMode = False  
End Sub
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

218 Eine Textbox, die den Cursor verfolgt

Wie eine verknüpfte Grafik erzeugt werden kann, haben Sie im Rezept 217 erfahren. Diesmal soll die verknüpfte Grafik dem Cursor folgen bzw. der jeweils markierten Zelle. Wir erstellen dazu zuerst manuell eine Textbox aus der Symbolleiste ZEICHNEN.

Der Textbox selbst können Sie nach Belieben, auch ohne VBA, eine gewünschte Formatierung zuweisen, zum Beispiel eine transparente rote Farbe. Klicken Sie mit der rechten Maustaste auf die Textbox und wählen Sie aus dem Kontextmenü den Eintrag TEXTFELD FORMATIEREN. Nehmen Sie in diesem Dialogfenster die gewünschten Einstellungen vor. Beachten Sie dabei, dass verschiedene Registerkarten zur Formatierung zur Verfügung stehen.

Per VBA programmieren wir nun, dass in der Textbox der Inhalt der Zellen B2, C2, B3 und C3 angezeigt wird. Damit der Inhalt der Textbox immer aktuell ist, verwenden wir die Ereignis-Prozedur SelectionChange. Mehr zum Thema »Ereignis-Prozeduren« erfahren Sie in der entsprechenden Kategorie.

Nachdem die Textbox mit dem gewünschten Zelleninhalt gefüllt ist, müssen wir uns um die Positionierung kümmern. Die Textbox soll am rechten unteren Rand der aktiven Zelle angezeigt werden. Wir verwenden dazu die Eigenschaft Offset. Offset(1,0).Top bedeutet, dass die Textbox jeweils eine Zeile unter der aktiven Zelle angezeigt wird. Offset(0,1).Left bedeutet, dass die Textbox zudem eine Spalte rechts neben der aktiven Zelle angezeigt wird. Beides kombiniert ergibt den gewünschten Effekt. Die Textbox erscheint jeweils in der rechten unteren Ecke (siehe Abbildung 171).

```
' =====  
' Auf CD      Buchdaten\Beispiele\07_Objekte\  
' Dateiname   07_01_Pictures.xls  
' Tabelle     07_Objects_FollowingTextbox  
' Ereignis    Tabelle4(07_Objects_FollowingTextbox)  
' =====
```

```

Private Sub Worksheet_SelectionChange(ByVal Target As Range)
    Dim shp As Shapes

    With ActiveSheet.Shapes(1)
        .TextFrame.Characters.Text = _
            Range("B2") & " " & _
            Range("C2") & Chr(10) & _
            Range("B3") & " " & _
            Range("C3")

        .Top = ActiveCell.Offset(1, 0).Top
        .Left = ActiveCell.Offset(0, 1).Left
    End With
End Sub

```

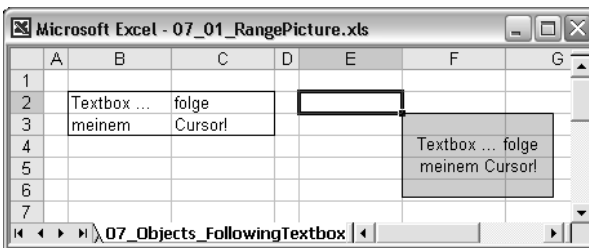


Abbildung 171: Ein Textfeld, das in der rechten unteren Ecke der aktiven Zelle angezeigt wird

Alternativ zu einer Textbox können Sie auch eine verknüpfte Grafik erzeugen, so wie es im Rezept 217 beschrieben wurde, und diese dem Cursor folgen lassen:

```

'=====
' Auf CD      Buchdaten\Beispiele\07_Objekte\
' Dateiname   07_01_Pictures.xls
' Tabelle     07_Objects_FollowingLinkedRange
' Ereignis    Tabelle4(07_Objects_FollowingLinkedRange)
'=====

Private Sub Worksheet_SelectionChange(ByVal Target As Range)
    Dim shp As Shapes

    With ActiveSheet.Shapes(1)
        .Top = ActiveCell.Offset(1, 0).Top
        .Left = ActiveCell.Offset(0, 1).Left
    End With
End Sub

```

219 Eine Grafik in eine Zelle einfügen

Der manuelle Weg, um ein Bild in ein Tabellenblatt einzufügen, führt über den Menüpunkt **EINFÜGEN | GRAFIK | AUS DATEI**. Zuvor wird die Zelle, in der das Bild eingefügt werden soll, aktiviert. Wenn das Bild größer als die Zelle ist, überlagert es rechterhand und unterhalb weitere Zellen.

Eine einzige Codezeile reicht aus, um ein Bild in ein Tabellenblatt einzufügen:

```
ActiveSheet.Pictures.Insert ("C:\Codebook.jpg")
```

Es wird dabei die Eigenschaft `Pictures` (Bild) sowie die Methode `Insert` (Einfügen), gefolgt vom Quellpfad verwendet. Das Bild wird in die momentan aktive Zelle eingefügt. Falls das Bild in einer bestimmten Zelle erscheinen soll, muss diese erst selektiert werden. Um eine bestimmte Zelle per VBA zu selektieren, fügen Sie zu Beginn der Prozedur eine weitere Codezeile ein, die zuerst die Zelle selektiert (`Select`).

Da ein `Select` in einem Code nach Möglichkeit zu vermeiden ist, verwenden wir eine elegantere Lösung. Sie denken nun vielleicht, dass die eine Codezeile um ein `Range`-Objekt erweitert werden kann:

```
' Die folgende Codezeile reicht nicht aus:
ActiveSheet.Range("$A$1").Pictures.Insert ("C:\Codebook.jpg")
```

Ganz so einfach ist es leider nicht, denn das Objekt `Range` unterstützt die Eigenschaft `Picture` nicht. Wir müssen mit einem kleinen Umweg arbeiten.

Zu Beginn der Prozedur wird der Pfad des Bildes an eine Variable übergeben. So kann in der nächsten Codezeile geprüft werden, ob die Pfadangabe korrekt ist. Wenn nicht, wird die Prozedur an dieser Stelle verlassen.

Um den Code möglichst übersichtlich zu gestalten, werden wir das Einfügen des Bildes referenzieren (`Set`). Mit der `With`-Anweisung wird angegeben, dass das Bild in die Zelle C5 eingefügt werden soll. Innerhalb dieser Prozedur wird der Abstand vom linken Tabellenrand (`Left`) bis zur Zelle C5 gemessen. Ebenso wird der Abstand vom oberen Tabellenrand (`Top`) zur Zelle C5 gemessen. Beide Maße werden direkt an die Grafik übergeben. Damit ist gewährleistet, dass der linke und obere Abstand der Grafik dem linken und oberen Rand der Zelle C5 entspricht (siehe Abbildung 172).

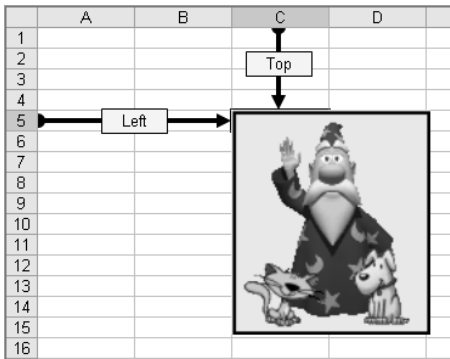


Abbildung 172: Abstand von oben und Abstand von links

```
' =====
' Auf CD   Buchdaten\Beispiele\07_Objekte\
' Dateiname 07_01_Pictures.xls
' Tabelle   07_Objects_InsertPicture
' Modul     mdl_04_InsertPicture
```

```
'=====

Sub InsertPicture()
    Dim pic As Picture
    Dim strPath As String

    strPath = ThisWorkbook.Path & "\Magician.jpg"
    If Dir(strPath) = "" Then Exit Sub

    ' Bild-Pfad referenzieren
    Set pic = ActiveSheet.Pictures.Insert(strPath)

    With ActiveSheet.Range("C5")
        ' Abstand links und oben
        pic.Left = .Left
        pic.Top = .Top
    End With

    Set pic = Nothing
End Sub
```

220 Bei Klick auf Zelle A1 eine Grafik in Zelle B1 einfügen

Dieses Beispiel basiert auf dem Rezept 219. Wir zeigen Ihnen nun, dass Sie ein Bild nur auf Wunsch einblenden lassen können. Das Bild soll in Zelle B1 angezeigt werden, wenn auf die Zelle A1 geklickt wird. Sobald eine andere Zelle aktiviert wird, verschwindet das Bild.

Damit dies umgesetzt werden kann, müssen wir mit einer Ereignis-Prozedur arbeiten. Mehr zum Thema Ereignis-Prozeduren erfahren Sie in der gleichnamigen Kategorie.

Zu Beginn der Prozedur wird festgelegt, dass die Grafik mit `Delete` gelöscht werden soll, wenn die aktive Zelle nicht A1 ist (<> "\$A\$1"). Das weitere Verfahren entspricht dem Rezept 219.

```
'=====
' Auf CD      Buchdaten\Beispiele\07_Objekte\
' Dateiname   07_01_Pictures.xls
' Tabelle     07_Objects_InsertPictureA1
' Ereignis    Tabelle7(07_Objects_InsertPictureA1)
'=====

Private Sub Worksheet_SelectionChange(ByVal Target As Range)
    If Target.Address <> "$A$1" Then
        If ActiveSheet.Shapes.Count > 0 Then
            ActiveSheet.Shapes(1).Delete
        End If
        Exit Sub
    End If

    Dim pic As Picture
    Dim strPath As String

    strPath = ThisWorkbook.Path & "\Magician.jpg"
    If Dir(strPath) = "" Then Exit Sub
```

```
' Bild-Pfad referenzieren
Set pic = ActiveSheet.Pictures.Insert(strPath)

With ActiveSheet.Range("B1")
    ' Abstand links und oben
    pic.Left = .Left
    pic.Top = .Top
End With

Set pic = Nothing
End Sub
```

221 Eine Grafik einfügen und der Zellengröße anpassen

Oftmals wäre es von Vorteil, wenn die eingefügte Grafik exakt der Größe der Zelle entsprechen würde, über welcher die Grafik eingefügt wird. Manuell würden Sie zuerst ein Bild einfügen und könnten dann mit gedrückter **Alt**-Taste das Bild in die Zelle oder den Bereich einpassen. Wenn mehrere Bilder so bearbeitet werden müssten, wäre der Aufwand recht erheblich.

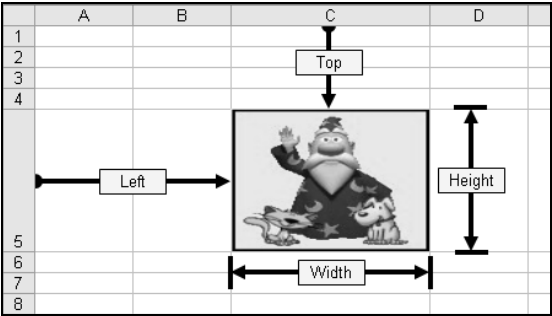


Abbildung 173: Zusätzlich die Höhe und Breite angeben (Height und Width)

Im Rezept 219 haben Sie erfahren, wie ein Bild mittels **Left** und **Top** in eine bestimmte Zelle eingefügt werden kann. Es sind nur zwei weitere Codezeilen erforderlich, um das Bild in der Größe der Zelle zu formatieren. Wir werden zusätzlich die Höhe und Breite der Zelle an die Grafik übergeben (**Height** und **Width**). Mit all den vier Angaben wird die Grafik der Größe der Zelle angepasst.

Wenn Sie das Bild nicht über einer vorgegebenen Zelle einfügen möchten, sondern jeweils über der aktiven Zelle, dann ersetzen Sie die Anweisung `ActiveSheet.Range("C5")` durch `ActiveCell`.

```
'=====
' Auf CD    Buchdaten\Beispiele\07_Objekte\
' Dateiname 07_01_Pictures.xls
' Tabelle   07_Objects_PictureSize
' Modul     mdl_05_PictureSize
'=====

Sub PictureSize()
    Dim pic As Picture
    Dim strPath As String
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

```

strPath = ThisWorkbook.Path & "\Magician.jpg"
If Dir(strPath) = "" Then Exit Sub

' Bild-Pfad referenzieren
Set pic = ActiveSheet.Pictures.Insert(strPath)

With ActiveSheet.Range("C5")
    ' Abstand links und oben
    pic.Left = .Left
    pic.Top = .Top
    ' Höhe und Breite
    pic.Height = .Height
    pic.Width = .Width
End With

Set pic = Nothing
End Sub

```

222 Eine Grafik exakt in einen Bereich einfügen

Im Rezept 221 haben Sie erfahren, wie eine Grafik an die Größe einer *Zelle* angepasst werden kann. Eine Grafik kann auch auf die Größe eines *Bereiches* angepasst werden, beispielsweise in den Bereich C5:D14.

Um den Code übersichtlich zu gestalten, werden wir den Bereich und das Bild inklusive Pfad entsprechend referenzieren (Set). Wir verwenden dazu die Variablen *rng* und *pic*. Der Variablen *rng* wird der Bereich C5:D14 übergeben. Dies entspricht dem Bereich, über dem die Grafik erscheinen soll. Der Variablen *pic* wird das Einfügen der Grafik mitsamt Pfad übergeben. Der Pfad wurde zu Beginn der Prozedur in der Variablen *strPath* gespeichert.

Um die Grafik der Größe des Bereiches anzupassen, müssen wir zuerst die Höhe und Breite des Bereiches ermitteln. Diese Werte werden später an die Grafik übergeben.

Die Höhe des Bereiches C5:D14 errechnet sich aus der Subtraktion von zwei Werten. Der erste Wert entspricht dem Abstand vom oberen Tabellenrand zum oberen Rand der ersten Zeile *nach* dem Bereich C5:D14, also Zeile 15. Als zweiten Wert benötigen wir den Abstand vom oberen Tabellenrand zur ersten Zeile im Bereich C5:D14, das entspricht der Zeile 5. Wenn diese beiden Werte bekannt sind, kann die Subtraktion erfolgen. Daraus resultiert die Höhe des Bereiches, über dem die Grafik eingefügt werden soll.

Um den Abstand vom oberen Tabellenrand zur Zeile 15 zu berechnen, zählen wir die im Bereich enthaltenen Zeilen. Theoretisch können Sie an Stelle der Anweisung `rng.Offset(rng.Rows.Count, 0).Top` auch gleich den Wert 15 angeben (`Rows(15).Top`). Der Effekt wäre in diesem Beispiel derselbe. Nachteilig wird das allerdings, wenn Sie später einen anderen Bereich verwenden möchten, über dem die Grafik eingefügt werden soll, denn dann müssten Sie an drei Stellen die Prozedur anpassen. Den Bereich selbst, die Höhe und die Breite, denn für die Breite verhält es sich genauso wie bei der Höhe. Sie sind deshalb mit der Berechnung der Zeilen für die Höhe und der Spalten für die Breite flexibler.

Nachdem die Höhe und Breite des Bereiches bekannt sind, können diese Werte beim Einfügen der Grafik an selbige übergeben werden. Dies geschieht in der *With*-Anweisung. Die *Größe* der Grafik ist somit festgelegt. Um die Grafik an der richtigen *Position* einzufügen, werden die Werte *Left* für die linke Ausrichtung und *Top* für die Ausrichtung von oben übergeben.

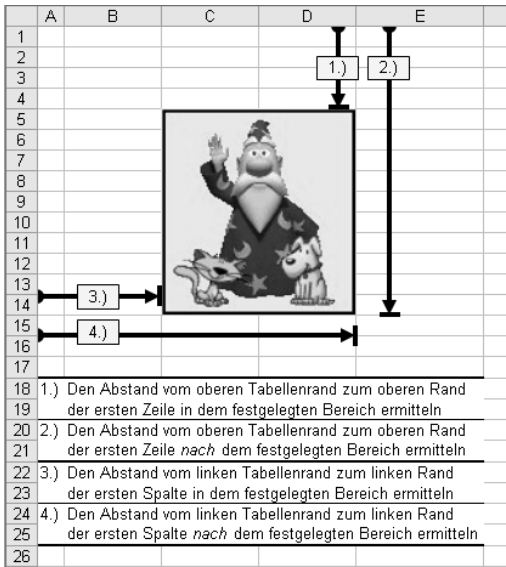


Abbildung 174: Größe des Bereichs ermitteln

```
'=====
' Auf CD      Buchdaten\Beispiele\07_Objekte\
' Dateiname  07_01_Pictures.xls
' Tabelle    07_Objects_PictureRange
' Modul      mdl_06_PictureRange
'=====
```

```
Sub PictureRange()
    Dim strPath As String
    Dim pic As Picture
    Dim dblWidth As Double
    Dim dblHeight As Double
    Dim rng As Range

    strPath = ThisWorkbook.Path & "\Magician.jpg"
    If Dir(strPath) = "" Then Exit Sub

    ' Zielzelle und Bild referenzieren
    Set rng = ActiveSheet.Range("C5:D14")
    Set pic = ActiveSheet.Pictures.Insert(strPath)

    ' Höhe der Zelle ermitteln
    dblHeight = rng.Offset(rng.Rows.Count, 0).Top - rng.Top

    ' Breite der Zelle ermitteln
    dblWidth = rng.Offset(0, rng.Columns.Count).Left - rng.Left

    ' Bild in Zelle einfügen und anpassen
    With pic
        .Width = dblWidth
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

```

        .Height = dblHeight
        .Left = rng.Left
        .Top = rng.Top
    End With

    Set rng = Nothing
    Set pic = Nothing
End Sub

```

223 Eine Grafik in einer Zelle zentrieren

In diesem Beispiel soll eine Grafik über einer Zelle zentriert werden. Der Effekt ist nur dann sichtbar, wenn die Zelle größer ist als die Grafik, denn die Grafik wird bei diesem Vorgang nicht verkleinert.

Nach der Variablen-Deklaration und der Referenzierung wird in zwei verschiedenen Codezeilen die Höhe und Breite der Zielzelle ermittelt. Diese Werte werden an zwei Variablen übergeben, welche später für die Berechnung der Ausrichtung verwendet werden. Das Vorgehen ist ähnlich dem Rezept 222. Nur können wir hier auf das Zählen von Zeilen und Spalten verzichten, da es sich ja nur um eine einzelne Zelle handelt.

Die Variable `dblHeight` enthält somit folgenden Wert: Den Abstand vom oberen Tabellenrand zum oberen Rand der Zelle unterhalb der Zielzelle. Davon wird der Abstand vom oberen Rand der Tabelle zum oberen Rand der Zielzelle subtrahiert. Als Ergebnis erhalten wir die Höhe der Zelle.

Der Variablen `dblWidth` wird der Abstand vom linken Tabellenrand zum linken Rand der Zelle neben der Zielzelle zugewiesen. Davon wird der Abstand vom linken Tabellenrand zum linken Rand der Zielzelle subtrahiert. Als Ergebnis erhalten wir die Breite der Zelle.

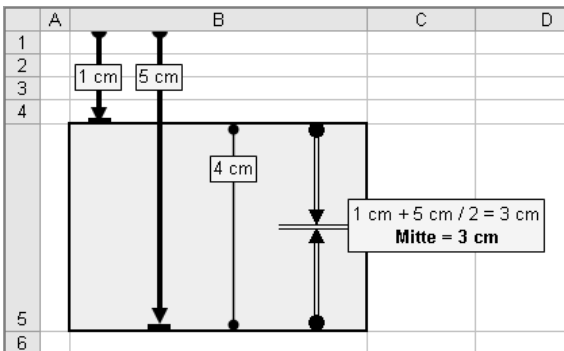


Abbildung 175: Darstellung der Berechnung der horizontalen Mitte

Als Nächstes werden wir die horizontale und vertikale Mitte der Zielzelle berechnen. Um den oberen Abstand zu berechnen, werden wir den Abstand vom oberen Rand der Tabelle zur ersten Zelle mit der zuvor berechneten Variablen `dblHeight` addieren. Dieser Wert wird durch den Wert 2 dividiert. Dasselbe geschieht mit der Grafik. Die Höhe der Grafik wird durch den Wert 2 dividiert. Die halbe Höhe des oberen Abstands wird von der halben Höhe der Grafik subtrahiert. Damit erhalten wir den Wert für die horizontale Mitte der Zielzelle. Dasselbe geschieht auch in der Breite.

In der `With`-Anweisung können nun die beiden zuletzt ermittelten Variablen `dblTop` und `dblLeft` an das Bild übergeben werden. Die Position ist somit genau in der Mitte der Zelle.


```
'=====
' Auf CD      Buchdaten\Beispiele\07_Objekte\
' Dateiname  07_01_Pictures.xls
' Tabelle    07_Objects_CenterPicture
' Modul      mdl_07_CenterPicture
'=====

Sub CenterPicture()
    Dim strPath As String
    Dim pic As Picture
    Dim rng As Range
    Dim dblTop As Double
    Dim dblLeft As Double
    Dim dblWidth As Double
    Dim dblHeight As Double

    strPath = ThisWorkbook.Path & "\Magician.jpg"
    If Dir(strPath) = "" Then Exit Sub

    ' Bild und Zielzelle referenzieren
    Set pic = ActiveSheet.Pictures.Insert(strPath)
    Set rng = ActiveSheet.Range("B2")

    ' Höhe der Zielzelle ermitteln
    dblHeight = rng.Offset(1, 0).Top - rng.Top

    ' Breite der Zielzelle ermitteln
    dblWidth = rng.Offset(0, 1).Left - rng.Left

    ' Den Abstand von oben berechnen
    dblTop = rng.Top + dblHeight / 2 - pic.Height / 2

    ' Den Abstand von links berechnen
    dblLeft = rng.Left + dblWidth / 2 - pic.Width / 2

    ' Grafik in der Mitte der Zelle positionieren
    With pic
        .Top = dblTop
        .Left = dblLeft
    End With

    Set rng = Nothing
    Set pic = Nothing
End Sub
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

224 Grafik(en) benennen

Den Namen eines Bildes können Sie dem Namensfeld entnehmen. Die Grafik muss dabei selektiert sein. Das Namensfeld ist das Feld, das sich links von der Bearbeitungsleiste befindet. Sie können Bilder nach Belieben neu benennen, indem Sie den Eintrag im Namensfeld überschreiben. Falls sich viele Bilder in Ihrem Tabellenblatt befinden und alle neu benannt und nummeriert werden sollen, wäre es ziemlich umständlich, wenn Sie erst jedes einzelne Bild selektieren und dann benennen müssten. Mit einer VBA-Prozedur, welche eine Schleife enthält, können Sie die Neubenennung automatisieren.

Das erste der beiden nachfolgenden Codebeispiele zeigt, wie ein einzelnes Bild mit Name neu benannt werden kann. Die erste Codezeile prüft, ob überhaupt ein Bild im Tabellenblatt vorhanden ist. Wenn nicht, wird die Prozedur verlassen. Dies geschieht um zu verhindern, dass der Debugger gestartet wird, falls keine Grafik in der Tabelle vorhanden ist.

```
' =====
' Auf CD      Buchdaten\Beispiele\07_Objekte\
' Dateiname  07_01_Pictures.xls
' Tabelle     07_Objects_NamePictures
' Modul      mdl_08_NamePictures
' =====

Sub NameOnePicture()
    If ActiveSheet.Pictures.Count < 1 Then Exit Sub
    ActiveSheet.Pictures(1).Name = "Mein Bild 1"
End Sub
```

Im nächsten Beispiel erfahren Sie, wie alle Bilder in einem Tabellenblatt neu benannt werden können. Wir verwenden dazu eine Zählschleife. Den Zähler setzen wir innerhalb der Prozedur ein, um einerseits die Bilder nacheinander anzusprechen und andererseits, um sie mit dem Zähler auch gleich zu nummerieren.

```
' =====
' Auf CD      Buchdaten\Beispiele\07_Objekte\
' Dateiname  07_01_Pictures.xls
' Tabelle     07_Objects_NamePictures
' Modul      mdl_08_NamePictures
' =====

Sub NameAllPictures()
    Dim i As Integer
    For i = 1 To ActiveSheet.Pictures.Count
        ActiveSheet.Pictures(i).Name = "Mein Bild " & i
    Next i
End Sub
```

225 Grafik(en) kopieren

Die beiden nachfolgenden Beispiele zeigen, wie Grafiken kopiert (Copy) werden können. Im ersten Beispiel wird lediglich eine einzelne Grafik kopiert und auf demselben Tabellenblatt wieder eingefügt (Paste).

```
' =====
' Auf CD      Buchdaten\Beispiele\07_Objekte\
' Dateiname  07_01_Pictures.xls
' Tabelle     07_Objects_CopyPictures
' Modul      mdl_09_CopyPictures
' =====

Sub CopyOnePicture()
    With ActiveSheet
```

```
        .Pictures(1).Copy
        .Paste
    End With
End Sub
```

Das zweite Beispiel zeigt, wie sämtliche Grafiken eines Tabellenblattes kopiert werden können. Die Grafiken sollen auf einem neuen Tabellenblatt eingefügt werden. Dazu fügen wir ein neues Tabellenblatt in die Mappe ein (Worksheets.Add.).

```
'=====
' Auf CD      Buchdaten\Beispiele\07_Objekte\
' Dateiname   07_01_Pictures.xls
' Tabelle     07_Objects_CopyPictures
' Modul       mdl_09_CopyPictures
'=====

Sub CopyAllPictures()
    ActiveSheet.Pictures.Copy
    Worksheets.Add
    ActiveSheet.Paste
End Sub
```

226 Grafik(en) löschen

In diesem Rezept erfahren Sie, wie Sie Grafiken mit Delete löschen können. In der ersten Prozedur wird nur ein einzelnes Bild gelöscht. Sie können das Bild dabei über den Index oder dessen Namen ansprechen. In der ersten Codezeile wird geprüft, ob überhaupt ein Bild vorhanden ist. Wenn nicht, wird die Prozedur verlassen.

Tipp

Das Ansprechen über Pictures (und nicht Shapes) hat den Vorteil, dass Elemente, die über die Symbolleiste FORMULAR eingefügt wurden, nicht entfernt werden. Wenn in Ihrem Tabellenblatt beispielsweise Befehlsschaltflächen aus der Symbolleiste FORMULAR enthalten sind, werden diese *nicht* gelöscht. Objekte, welche über die Symbolleiste STEUERELEMENT-TOOLBOX erzeugt wurden, werden alle entfernt.

```
'=====
' Auf CD      Buchdaten\Beispiele\07_Objekte\
' Dateiname   07_01_Pictures.xls
' Tabelle     07_Objects_DeletePictures
' Modul       mdl_10_DeletePictures
'=====

Sub DeleteOnePicture()
    If ActiveSheet.Pictures.Count < 1 Then Exit Sub
    ActiveSheet.Pictures(1).Delete
End Sub
```

In der zweiten Prozedur werden sämtliche Bilder in dem Tabellenblatt gelöscht. Das ist denkbar einfach, denn, wie Sie sehen können, besteht der Code nur aus einer einzigen Zeile.

Grundlagen

Allgemein

Datum und Zeit

Formeln

Steuerungselemente

Befehlsliste

Objekt

Diagramm

Ereignisse

User-Form

Web/Mail

Extern

Gemischtes

Spezial

```
'=====
' Auf CD      Buchdaten\Beispiele\07_Objekte\
' Dateiname   07_01_Pictures.xls
' Tabelle     07_Objects_DeletePictures
' Modul       mdl_10_DeletePictures
'=====

Sub DeleteAllPictures()
    ActiveSheet.Pictures.Delete
End Sub
```

Das dritte Beispiel zeigt, wie sämtliche Bilder einer Mappe entfernt werden können. Wir verwenden dazu eine For-Schleife, die jedes einzelne Tabellenblatt anspricht und alle darin enthaltenen Grafiken entfernt.

```
'=====
' Auf CD      Buchdaten\Beispiele\07_Objekte\
' Dateiname   07_01_Pictures.xls
' Tabelle     07_Objects_DeletePictures
' Modul       mdl_10_DeletePictures
'=====

Sub DeleteAllPicturesWB()
    Dim WS As Worksheet
    ' Alle Grafiken der Mappe löschen
    For Each WS In ActiveWorkbook.Worksheets
        WS.Pictures.Delete
    Next WS
End Sub
```

227 Grafik formatieren

Um Formatierungen manuell vorzunehmen, klicken Sie mit der rechten Maustaste auf die Grafik und wählen aus dem Kontextmenü den Eintrag GRAFIK FORMATIEREN. Das Dialogfenster, das sich nun öffnet, zeigt verschiedene Registerkarten. Alternativ können Sie dasselbe Dialogfenster öffnen, indem Sie doppelt auf die Grafik klicken. In diesem Beispiel wenden wir uns dem unteren Teil der Registerkarte BILD zu, nämlich der BILDSTEUERUNG.

Hinweis

Per VBA lassen sich auch alle anderen Einstellungen programmieren, die das Dialogfenster anzeigt. Verwenden Sie dazu am besten den Makro-Rekorder und zeichnen Sie die Schritte auf. Wechseln Sie dann in den VBA-Editor und bereinigen Sie den Code. Wahlweise können Sie einen solchen Code auch als Ereignis-Prozedur verwenden. Beispielsweise, indem Sie den Code mit einem Steuerelement verknüpfen. Wie Sie das umsetzen können, ist Bestandteil dieses Rezeptes.

In unserem Tabellenblatt befinden sich drei Optionsfelder und zwei Bildlaufleisten aus der Symbolleiste STEUERELEMENT-TOOLBOX. Um diese Elemente zu programmieren, verwenden wir für jedes eine eigene Ereignis-Prozedur. Die Ereignis-Prozeduren sind an das Tabellenblatt gebunden, in dem sich die Steuerelemente befinden.



Abbildung 176: Grafik formatieren

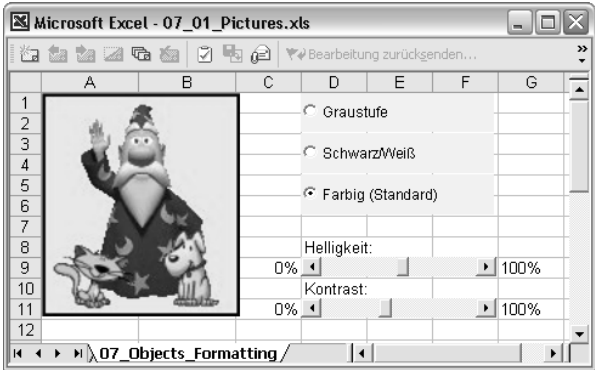


Abbildung 177: Grafik mittels Steuerelementen formatieren

Die drei Optionsfelder widmen sich der Farbe der Grafik. Wie Sie der Abbildung 177 entnehmen können, kann die Grafik in ein Graustufen- oder Schwarzweißbild umgewandelt werden. Die Option FARBIG setzt die Einstellungen zurück. Unser Bild wird somit wieder farbig dargestellt.

```
'=====
' Auf CD      Buchdaten\Beispiele\07_Objekte\
' Dateiname  07_01_Pictures.xls
' Tabelle    07_Objects_Formatting
' Ereignis   Tabelle15(07_Objects_Formatting)
'=====

Private Sub OptionButton1_Click()
    ' Graustufen
    ActiveSheet.Shapes(1).PictureFormat.ColorType _
        = msoPictureGrayscale
End Sub
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

```

Private Sub OptionButton2_Click()
    ' Schwarz/Weiß
    ActiveSheet.Shapes(1).PictureFormat.ColorType _
        = msoPictureBlackAndWhite
End Sub

Private Sub OptionButton3_Click()
    ' Farbig (Standard)
    ActiveSheet.Shapes(1).PictureFormat.ColorType _
        = msoPictureAutomatic
End Sub

```

Über die beiden Bildlaufleisten kann die Grafik in der Helligkeit `Brightness` sowie im Kontrast `Contrast` verändert werden. Es sind Werte im Bereich zwischen 0% und 100% möglich. In der Ereignisprozedur nehmen wir entsprechende Einschränkungen vor, indem wir einen Minimal- und einen Maximal-Wert festlegen (Min und Max). Da es sich dabei um Prozentwerte handelt, muss dieser Wert bei der Zuweisung an die Bildlaufleisten jeweils durch 100 dividiert werden.

```

' =====
' Auf CD      Buchdaten\Beispiele\07_Objekte\
' Dateiname  07_01_Pictures.xls
' Tabelle     07_Objects_Formatting
' Ereignis    Tabelle15(07_Objects_Formatting)
' =====

Private Sub ScrollBar1_Change()
    ' Helligkeit
    With ScrollBar1
        .Min = 0
        .Max = 100
    End With

    ActiveSheet.Shapes(1).PictureFormat.Brightness _
        = ScrollBar1 / 100
End Sub

' =====
' Auf CD      Buchdaten\Beispiele\07_Objekte\
' Dateiname  07_01_Pictures.xls
' Tabelle     07_Objects_Formatting
' Ereignis    Tabelle15(07_Objects_Formatting)
' =====

Private Sub ScrollBar2_Change()
    ' Kontrast
    With ScrollBar2
        .Min = 0
        .Max = 100
    End With

```

```
ActiveSheet.Shapes(1).PictureFormat.Contrast _  
    = ScrollBar2 / 100  
End Sub
```

228 Bedingtes Einfügen einer Grafik

Sie können Grafiken auch bedingt vom Zellinhalt einfügen. Dazu verwenden Sie am besten eine Ereignis-Prozedur, welche an das entsprechende Tabellenblatt gebunden ist. In unserem Beispiel soll je nach Inhalt der Zelle A1 eine bestimmte Grafik eingefügt werden. Wenn der Zellinhalt kleiner als 0 ist, soll eine rote Grafik eingefügt werden. Wenn der Zellinhalt dem Wert 0 entspricht, soll eine gelbe Grafik eingefügt werden. Wenn der Zellinhalt größer als 0 ist, soll eine grüne Grafik eingefügt werden.

Zu Beginn der Prozedur wird in der If-Entscheidung geprüft, ob bereits eine Grafik in dem Tabellenblatt enthalten ist. Wenn dies zutrifft, wird sie gelöscht. So wird bei jeder Veränderung des Wertes in der Zelle A1 zuerst eine bereits vorhandene Grafik gelöscht.

Im der zweiten If-Entscheidung wird geprüft, welcher Wert in die Zelle A1 eingegeben wurde. Je nachdem wird die Grafik der entsprechenden Farbe über der Zelle B1 ausgegeben.

Die Grafiken in den drei Farben Rot, Gelb und Grün finden Sie auf der CD im selben Verzeichnis wie die Beispieldatei.

```
'=====
' Auf CD      Buchdaten\Beispiele\07_Objekte\
' Dateiname   07_01_Pictures.xls
' Tabelle     07_Objects_Conditional
' Ereignis    Tabelle15(07_Objects_Conditional)
'=====
```

```
Private Sub Worksheet_Change(ByVal Target As Excel.Range)
    Dim shp As Shape
    Dim strPath As String

    strPath = ThisWorkbook.Path

    If ActiveSheet.Shapes.Count > 0 Then
        ActiveSheet.Shapes(1).Delete
    End If

    If Target.Address = "$A$1" Then
        With ActiveSheet
            If Target.Value < 0 Then
                .Pictures.Insert (strPath & "\Rot.jpg")
                With .Shapes(1)
                    .Left = Range("B1").Left
                    .Top = Range("B1").Top
                End With
            ElseIf Target.Value = 0 Then
                .Pictures.Insert (strPath & "\Gelb.jpg")
                With .Shapes(1)
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

```

        .Left = Range("B1").Left
        .Top = Range("B1").Top
    End With
ElseIf Target.Value > 0 Then
    .Pictures.Insert (strPath & "\Gruen.jpg")
    With .Shapes(1)
        .Left = Range("B1").Left
        .Top = Range("B1").Top
    End With
End If
End With
End If
End Sub

```

229 Eine animierte Grafik einbinden

Animierte Grafiken, also jene mit der Dateierweiterung *.gif oder *.png, können zwar in ein Tabellenblatt eingefügt werden, die Animation wird jedoch nicht ausgeführt. Es ist tatsächlich so, dass Excel nicht in der Lage ist, Animationen auszuführen. Beim Einfügen einer GIF- oder PNG-Grafik wird lediglich das erste Bild der animierten Bildfolge angezeigt. Ein Webbrowser hingegen ist in der Lage, Animationen auszuführen. Sie können ein Browser-Fenster in Ihr Tabellenblatt integrieren. Dabei wird auf ein ActiveX-Control zugegriffen.

Achtung

Das folgende Beispiel funktioniert nur, sofern der Windows Explorer ab Version 3.0 installiert ist. Dieser stellt die erforderliche ActiveX-Schnittstelle, ein Browser-Control und ein Internet-Explorer-Objekt zur Verfügung. Das Browser-Control stellt lediglich ein Browser-Fenster, ohne Schaltflächen und Eingabezeile, dar.

Vorgehensweise zum Einbinden des Controls:

1. Blenden Sie die Symbolleite **STEUERELEMENT-TOOLBOX** ein.
2. Klicken Sie auf die Schaltfläche **WEITERE STEUERELEMENTE**. Ein DropDown-Menü öffnet sich.
3. Bewegen Sie den Rollbalken so weit nach unten, bis Sie den Eintrag **MICROSOFT WEBBROWSER** sehen können, und klicken Sie auf diesen Menüpunkt. Das Menü verschwindet. Der Mauszeiger stellt nun ein Kreuz dar.
4. Ziehen Sie in Ihrem Tabellenblatt mit gedrückter linker Maustaste ein Viereck. Ein schwarzes Fenster mit dem Windows-Zeichen ist nun zu sehen. Das Browser-Fenster wurde eingebunden. Das weitere Vorgehen wird per VBA gesteuert.

Die vorangegangenen Schritte haben Ihnen gezeigt, wie Sie ein Browser-Fenster in Ihre Excel-Tabelle einbinden können. In diesem Fenster soll die animierte Grafik angezeigt werden. Dazu benötigen wir eine VBA-Prozedur.

Mittels `Webbrowser1` wird das erste Browser-Fenster angesprochen. Das heißt, dass die 1 hinter `Webbrowser` für den Index steht. Falls mehrere Webbrowser in Ihrem Tabellenblatt enthalten sind, werden diese aufsteigend nummeriert: `Webbrowser2`, `Webbrowser3` usw.

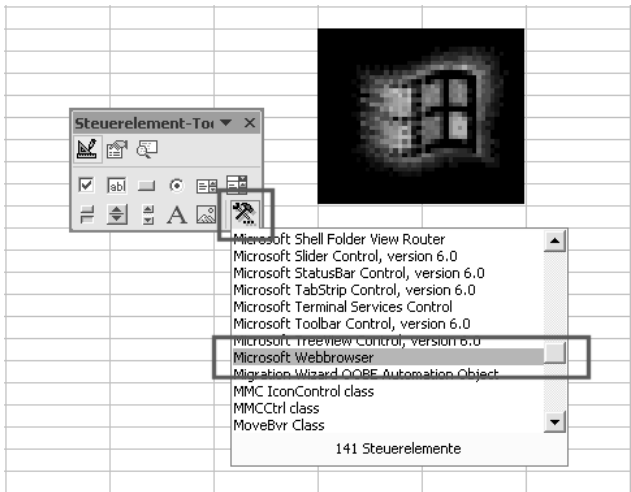


Abbildung 178: Webbrowser in Tabelle einbinden

```
'=====
' Auf CD      Buchdaten\Beispiele\07_Objekte\
' Dateiname  07_02_Webbrowser.xls
' Modul      mdl_01_WebbrowserLocal
'=====

Sub AnimatedGIFLocal()
    ActiveSheet.WebBrowser1.Navigate _
        ThisWorkbook.Path & "\Earth.gif"
End Sub
```

Der Methode `Navigate` wird der Pfad übergeben. In unserem Beispiel befindet sich die Grafik auf dem Laufwerk C:\. Sie können stattdessen auch eine Internet-Adresse (URL) angeben, die zu einer Grafik führt, es handelt sich bei dem `Control` ja um einen *Webbrowser*.

Achtung

Beachten Sie, dass das Verwenden des Webbrowsers nur funktioniert, wenn eine Verbindung zum Internet besteht.

```
'=====
' Auf CD      Buchdaten\Beispiele\07_Objekte\
' Dateiname  07_02_Webbrowser.xls
' Modul      mdl_02_WebbrowserInternet
'=====

Sub AnimatedGIFWeb()
    ActiveSheet.WebBrowser2.Navigate _
        "http://www.jumper.ch/Logos/Moni_Compi.gif"
End Sub
```

Grundlagen

Allgemein

Datum
Zeit

Funktionen

Steuerelemente

Befehle
leisten

Objekt

Diagramm

Ereignisse

User-Form

Web/
Mail

Extern

Gemischtes

Spezial

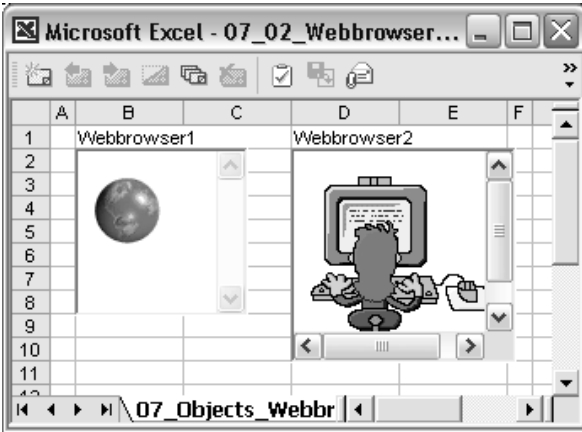


Abbildung 179: Zwei eingebundene Webbrowser-Fenster mit Animationen

230 Ein WordArt-Objekt erzeugen

WordArt sind die beliebten Schriftzüge, mit denen witzige Effekte erzeugt werden können. WordArt ist ein Shape-Objekt und muss entsprechend angesprochen werden. Ein Blick in ein Englisch-Deutsch-Wörterbuch verrät, dass der englische Ausdruck »Shape« übersetzt Form oder Gestalt bedeutet. Damit kehren wir an dieser Stelle also dem Objekt Picture, welches für Grafiken verwendet wird, den Rücken zu.

Ein WordArt-Objekt kann manuell per Klick auf das entsprechende Symbol in der Symbolleiste ZEICHNEN erstellt werden.

Um ein WordArt-Objekt per VBA einzufügen, ist es unerlässlich, die Syntax zu diesem Objekt zu kennen:

```
Ausdruck.AddTextEffect(PresetTextEffect, Text, FontName, FontSize, FontBold,
    FontItalic, Left, Top)
```

Die Methode AddTextEffect umfasst insgesamt acht Argumente. Jedes dieser Argumente ist erforderlich. Der Tabelle 48 können Sie die Argumente sowie deren Beschreibung entnehmen.

Argumente zu AddTextEffect	Beschreibung
PresetTextEffect	Effekt (siehe WordArt-Katalog)
Text	Text, den das WordArt darstellt
FontName	Name der Schriftart
FontSize	Schriftgröße
FontBold	Schriftdicke
FontItalic	Schriftneigung
Left	Abstand vom linken Tabellenrand
Top	Abstand vom oberen Tabellenrand

Tabelle 48: Argumente zur Methode AddTextEffect

Der nachfolgende Code erzeugt ein WordArt-Objekt:

```
'=====
' Auf CD      Buchdaten\Beispiele\07_Objekte\
' Dateiname  07_03_Shapes.xls
' Tabelle     07_Objects_WordArt
' Modul      mdl_01_CreateWordArt
'=====

Sub CreateWordArt()
    Dim shp As Shape

    Set shp = ActiveSheet.Shapes.AddTextEffect _
        (PresetTextEffect:=msoTextEffect21, _
        Text:="Codebook", _
        FontName:="Arial", _
        FontSize:=50, _
        FontBold:=msoTrue, _
        FontItalic:=msoFalse, _
        Left:=260, _
        Top:=260)
End Sub
```

231 Ein WordArt-Objekt formatieren

Das Thema »Formatierung eines WordArt-Objekts« ist schier unerschöpflich. Darauf möchten wir jedoch an dieser Stelle verzichten und lediglich anhand eines einzigen Beispiels zeigen, wie ein WordArt-Objekt korrekt referenziert und formatiert werden kann. Um mehr Formatierungen zu erreichen, raten wir Ihnen, mit dem Makro-Rekorder zu arbeiten. Zeichnen Sie die gewünschten Schritte auf und passen Sie den VBA-Code anschließend an. Auf diese Weise erfahren Sie am einfachsten die benötigten Objekte, Eigenschaften und Methoden.

Der Abbildung 180 können Sie die 30 verschiedenen Effekte entnehmen, die der WordArt-Katalog zur Verfügung stellt. Jedem dieser Effekte ist eine Konstante zugewiesen, die wahlweise im VBA-Code verwendet werden kann. Die Konstanten sind von 1 bis 30 durchnummeriert (msoTextEffect1, msoTextEffect2, msoTextEffect3 usw.). Das erste Bild zeigt den Effekt 1, das Bild rechts daneben zeigt den Effekt 2. Die Bilder wurden somit von links nach rechts und von oben nach unten nummeriert.

Neben dem WordArt-Katalog stehen Ihnen zusätzlich die WordArt-Formen zu Verfügung. Diese werden über die Eigenschaft PresetShape angesprochen. Es stehen insgesamt 40 solcher Formen zur Verfügung. Der Tabelle 49 können Sie die entsprechenden Konstanten entnehmen. Jede dieser Konstanten wird durch ein msoTextEffectShape angeführt. Für TextPlain lautet der ausgeschriebene Name zur Konstante demnach msoTextEffectShapeTextPlain.

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

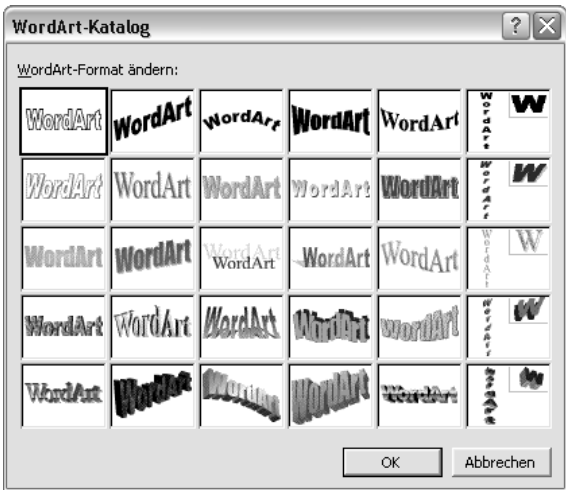


Abbildung 180: Der WordArt-Katalog

PlainText	Stop	TriangleUp	TriangleDown	ChevronUp
ChevronDown	RingInside	RingOutside	ArchUpCurve	ArchDownCurve
CircleCurve	ButtonCurve	ArchUpPour	ArchDownPour	CirclePour
ButtonPour	CurveUp	CurveDown	CanUp	CanDown
Wave1	Wave2	DoubleWave1	DoubleWave2	Inflate
Deflate	InflateBottom	DeflateBottom	InflateTop	DeflateTop

Tabelle 49: WordArt-Formen

				
DeflateInflate	DeflateInflate-Deflate	FadeRight	FadeLeft	FadeUp
				
FadeDown	SlantUp	SlantDown	CascadeUp	CascadeDown

Tabelle 49: WordArt-Formen (Forts.)

Achtung

Falls in Ihrem Tabellenblatt neben dem WordArt-Objekt noch andere Elemente (Shapes) enthalten sind, denken Sie beim Ansprechen über den Index daran, dass die richtige Nummer verwendet werden muss.

In unserem Beispiel sind in dem Tabellenblatt neben dem WortArt-Objekt noch zwei Befehlsschaltflächen enthalten. Auch diese können in VBA als Shapes-Objekt angesprochen werden. Da diese vor dem WordArt-Objekt eingefügt wurden, tragen sie den Index 1 und 2. Das WordArt-Objekt, das wir im Rezept zuvor eingefügt haben, trägt somit den Index 3. Alternativ zum Index können Sie das WordArt-Objekt auch über dessen Namen ansprechen. Um den Namen des WordArt-Objekts zu sehen, muss es aktiv sein. Im Namensfeld (links neben der Bearbeitungszeile) ist der Name abzulesen.

```
'=====
' Auf CD      Buchdaten\Beispiele\07_Objekte\
' Dateiname  07_03_Shapes.xls
' Tabelle     07_Objects_WordArt
' Modul      mdl_02_FormatWordArt
'=====

Sub FormatWordArt()
    Dim shp As Shapes

    Set shp = ActiveSheet.Shapes

    With shp(3)
        ' Effekte
        With .TextEffect
            .PresetTextEffect = msoTextEffect21
            .PresetShape = msoTextEffectShapeArchDownCurve
        End With

        ' Hintergrund
        With .Fill
            .Visible = msoTrue
            .Solid
            .ForeColor.SchemeColor = 15
            .Transparency = 0.9
        End With
    End With
End Sub
```

Grund-lagen

Allge-mein

Datu Zeit

tungen

Steuer elemen

Befehl leisten

Objekt

Dia-gramm

Ereig-nisse

User-Form

Web/Mail

Extern

Ge-misch tes

Specia

```

End With

' Linie
With .Line
    .Weight = 20
    .DashStyle = msoLineSolid
    .Style = msoLineSingle
    .Transparency = 0.5
    .Visible = msoTrue
    .ForeColor.RGB = RGB(255, 0, 0)
    .BackColor.RGB = RGB(0, 255, 0)
End With

' Drehung
.IncrementRotation 180

' Position und Größe
.Top = 200
.Left = 200
.Width = 300
.Height = 50
End With
End Sub

```

232 AutoFormen einfügen

Beim Thema »WordArt« haben wir erwähnt, dass das Thema schier unerschöpflich ist. Bei AutoFormen verhält es sich genauso. Es stehen insgesamt rund 130 AutoFormen zur Auswahl. Es können zudem viele Formatierungen vorgenommen werden, wie zum Beispiel Schatten und 3-D-Schatten. Dazu stehen sogar eigene Symbolleisten zur Verfügung. Linien und Fülleffekte können eingesetzt werden und vieles mehr.

Ein AutoForm kann mit der Methode `AddShape` erzeugt werden. Die Syntax dazu lautet:

```
Ausdruck.AddShape(Type, Left, Top, Width, Height)
```

Fünf Argumente sind erforderlich, um eine AutoForm zu erzeugen. Die Beschreibung können Sie der Tabelle 50 entnehmen.

Argumente zu AddShape	Beschreibung
Type	AutoForm-Type
Left	Abstand von links, gemessen am Tabellenblatt
Top	Abstand von oben, gemessen am Tabellenblatt
Width	Breite des Objektes in Punkten
Height	Höhe des Objektes in Punkten

Tabelle 50: Argumente zur Methode `AddShape`

Ein AutoForm erzeugen:

```
'=====
' Auf CD      Buchdaten\Beispiele\07_Objekte\
' Dateiname  07_03_Shapes.xls
' Tabelle     07_Objects_AutoShapes
' Modul      mdl_03_CreateAutoShape
'=====

Sub CreateAutoShape()
    Dim shp As Shape

    Set shp = ActiveSheet.Shapes.AddShape _
        (Type:=msoShapeCloudCallout, _
         Left:=230, _
         Top:=15, _
         Width:=200, _
         Height:=100)

    shp.TextFrame.Characters.Text = "Codebook"
End Sub
```

233 AutoFormen formatieren

Wie bereits im Rezept 232 erwähnt, sind die Formatierungsmöglichkeiten für ein AutoForm sehr umfangreich. Das nachfolgende Codebeispiel zeigt Ihnen verschiedene Möglichkeiten anhand eines einzelnen Beispiels. Die Beschreibung dazu finden Sie direkt im Code.

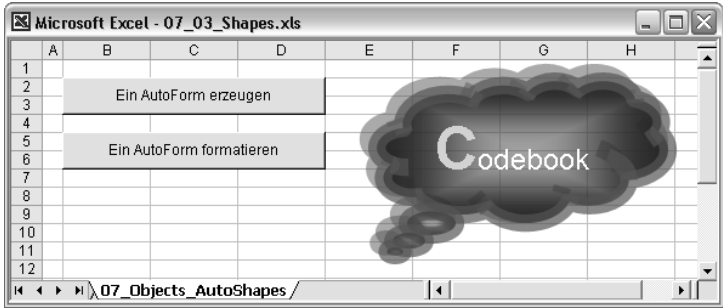


Abbildung 181: Formatiertes AutoForm

```
'=====
' Auf CD      Buchdaten\Beispiele\07_Objekte\
' Dateiname  07_03_Shapes.xls
' Tabelle     07_Objects_AutoShapes
' Modul      mdl_04_FormatAutoShape
'=====

Sub FormatAutoShape()
    Dim shp As Shapes
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

```
Set shp = ActiveSheet.Shapes
```

```
' Das WordArt-Objekt ist neben den beiden Schaltflächen das
' 3. Objekt in dem Tabellenblatt. Hier wird zuerst geprüft,
' ob das WordArt-Objekt zuvor eingefügt wurde.
```

```
If shp.Count < 3 Then Exit Sub
```

```
With shp(3)
```

```
    ' Ersten Buchstaben formatieren
```

```
    With .TextFrame.Characters(Start:=1, Length:=1).Font
```

```
        .Name = "Arial"
```

```
        .FontStyle = "Fett"
```

```
        .Size = 40
```

```
        .ColorIndex = 40
```

```
    End With
```

```
    ' Restliche Buchstaben formatieren
```

```
    With .TextFrame.Characters(Start:=2, Length:=7).Font
```

```
        .Name = "Arial"
```

```
        .Size = 20
```

```
        .ColorIndex = 2
```

```
    End With
```

```
    ' Text ausrichten
```

```
    With .TextFrame
```

```
        .HorizontalAlignment = xlCenter
```

```
        .VerticalAlignment = xlCenter
```

```
    End With
```

```
    ' Füllfarbe 90% transparent
```

```
    .Fill.Transparency = 0.9
```

```
    ' Linien formatieren
```

```
    With .Line
```

```
        .Weight = 12
```

```
        .Transparency = 0.3
```

```
        .ForeColor.RGB = RGB(255, 0, 0)
```

```
        .BackColor.RGB = RGB(255, 0, 0)
```

```
    End With
```

```
    ' Fülleffekt
```

```
    With .Fill
```

```
        .ForeColor.RGB = RGB(255, 0, 0)
```

```
        .OneColorGradient msoGradientFromCenter, 1, 0.23
```

```
    End With
```

```
    ' Schatten
```

```
    With .Shadow
```

```
        .Type = msoShadow1
```

```
        .ForeColor.RGB = RGB(255, 0, 0)
```

```
    End With
```

```
End With
```

```
End Sub
```

234 Objekte lokalisieren

Wenn Sie ermitteln möchten, in welchen Zellen sich die Objekte in Ihrem Tabellenblatt befinden, verwenden Sie die Eigenschaft `TopLeftCell` in Kombination mit der Eigenschaft `Address`. Die Eigenschaft `TopLeftCell` ermittelt die obere linke Adresse der Zelle, über der das Objekt liegt. In Kombination mit der Eigenschaft `Address(False, False)` wird der relative Zellbezug ausgegeben (z.B. A1). Wenn Sie die beiden `False` weglassen, wird der absolute Zellbezug (z.B. \$A\$1) verwendet.

In unserem Beispiel übergeben wir die Adresse und den Namen eines jeden Objektes im angegebenen Tabellenblatt an die Variable `strTemp`. Die Variable wird in der `For...Each`-Schleife nach und nach mit den Zelladressen gefüllt. Am Ende der Prozedur wird ein Nachrichtenfenster `MsgBox` angezeigt, das den Inhalt der Zelladressen und den Namen der Objekte anzeigt. Wenn keine Objekte im Tabellenblatt gefunden werden, gibt das Nachrichtenfenster ebenfalls eine entsprechende Meldung aus.

```
'=====
' Auf CD      Buchdaten\Beispiele\07_Objekte\
' Dateiname  07_03_Shapes.xls
' Tabelle     07_Objects_ShapesAddress
' Modul       mdl_05_ShapesAddress
'=====

Sub ShapesAddress()
    Dim shp As Shape
    Dim i As Integer
    Dim strTemp As String

    For Each shp In ActiveSheet.Shapes
        strTemp = strTemp & vbCrLf & _
            shp.TopLeftCell.Address(False, False) & _
            " - " & shp.Name
        i = i + 1
    Next shp

    If strTemp = "" Then
        MsgBox "Es wurden keine Objekte gefunden."
    Else
        MsgBox strTemp
    End If
End Sub
```

235 Ein ClipArt in der Farbe verändern

ClipArts sind Bilder vom Dateityp `*.wmf` (Windows Meta File). Viele ClipArts sind bereits im Office-Paket enthalten. Weitere ClipArts können aus dem Internet auf der Webseite von Microsoft bezogen werden.

Wenn Sie ein ClipArt in Ihr Tabellenblatt einfügen (EINFÜGEN | GRAFIK | CLIPART), können Sie genauso wie bei Bildern und AutoFormen verschiedene Formatierungen vornehmen. Klicken Sie dabei doppelt auf das ClipArt und nehmen Sie die gewünschten Änderungen vor.

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

Mit einem ClipArt kann noch viel mehr erreicht werden. Im ersten Moment scheint es ein einziges Element zu sein. Dieses besteht jedoch aus vielen Einzelteilen, die gruppiert sind. Diese Gruppierungen lassen sich aufheben. Klicken Sie bei aktiviertem ClipArt in der Symbolleiste ZEICHNEN auf den Eintrag GRUPPIERUNG AUFHEBEN. Dieser Vorgang muss zweimal ausgeführt werden. Dies ist erforderlich, da sich nicht die gesamte Grafik in einem Schritt komplett auflösen lässt. Jedes einzelne Element ist nun deutlich sichtbar, denn jedes ist selektiert. Um nur ein einzelnes Element zu selektieren, klicken Sie erst in eine beliebige Zelle, um die gruppierte Selektierung aufzuheben. Klicken Sie dann das gewünschte Element an. Das Element lässt sich nun im Einzelnen formatieren. Genau diese Schritte werden wir nun in einer VBA-Prozedur umsetzen.

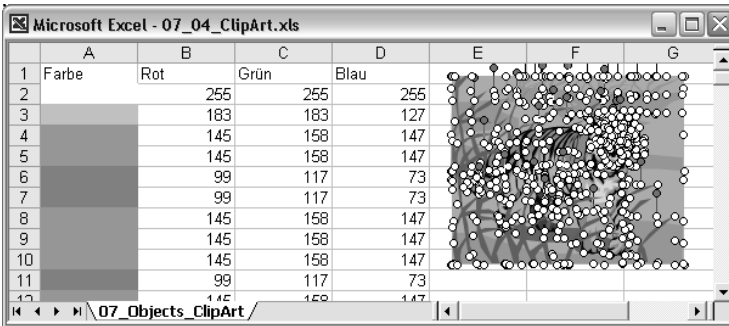


Abbildung 182: ClipArt mit aufgehobener Gruppierung

In unserer Prozedur sollen die RGB-Farben der einzelnen Elemente Shapes ermittelt und im Tabellenblatt ausgegeben werden. Wir gehen dabei noch einen Schritt weiter. Dem Bild sollen automatisch neue Farben zugewiesen werden. Sobald die einzelnen RGB-Werte bekannt sind, ist das sehr einfach. Es müssen bei der Zuweisung lediglich die Variablen vertauscht werden.

Sehen wir uns den Aufbau der Prozedur Schritt für Schritt an. Direkt nach dem Start der Prozedur soll ein Eingabefeld `InputBox` angezeigt werden. Darin kann zwischen zwei verschiedenen Bilddateien gewählt werden. In einer `If ... Then ... Else`-Entscheidung wird geprüft, welche Datei gewählt wurde und je nachdem der Dateiname an eine Variable übergeben. Danach wird das ClipArt referenziert.

Da nun der Name der gewünschten Datei bekannt ist, kann dieser mit dem Pfad verbunden werden. Da sich die Bilder und die Beispielmappe im selben Ordner befinden, können wir `ThisWorkbook.Path` verwenden. Die nächste Codezeile prüft, ob eine solche Datei in dem Pfad vorhanden ist. Falls nicht, wird die Prozedur verlassen.

In einem weiteren Schritt werden allfällig vorhandene Zeichnungsobjekte `DrawingObjects`, sofern vorhanden, entfernt. Zudem werden die Spalten A bis D von alten Einträgen bereinigt. Das ClipArt kann nun eingefügt werden (`Pictures.Insert`). Nachdem das ClipArt im Tabellenblatt vorhanden ist, kann die Gruppierung aufgehoben werden. Dies muss, genauso wie beim manuellen Vorgang, zweimal ausgeführt werden (`Ungroup.Ungroup`).

Nun wenden wir uns dem Kern unserer Prozedur zu. Die Füllfarbe eines Shapes kann über `Fill.ForeColor` ermittelt werden. Das Problem dabei ist, dass dabei nur ein einzelner Wert geliefert wird. Wir benötigen jedoch drei Werte für RGB (Rot, Grün und Blau). Um die einzelnen Werte zu ermitteln, werden entsprechende Berechnungen durchgeführt. Diese nach und nach ermittelten Werte werden an verschiedene Variablen übergeben. Für jeden RGB-Farbwert verwenden wir eine eigene Variable. In der Kategorie »Einstieg« finden Sie eine nähere Beschreibung zum Umgang mit RGB-Werten.

Um das Objekt umzufärben, werden wir die Farbwerte vertauschen. Die Reihenfolge der RGB-Farben ist: Rot, Grün und Blau. Wir verwenden die Reihenfolge: Grün, Rot, Blau. Sie können die Werte nach Belieben vertauschen (z.B. Blau, Grün, Rot).

Den Zellen in der Spalte A weisen wir die Originalfarben jedes einzelnen Elementes des ClipArts zu. In den Spalten B bis D werden die RGB-Werte in der korrekten Reihenfolge Rot, Grün und Blau ausgegeben.

Um die Prozedur zu beschleunigen und das Bildflackern zu unterdrücken, können Sie zu Beginn der Prozedur ein `Application.ScreenUpdating = False` verwenden. Am Ende der Prozedur muss dieses wieder auf `True` gesetzt werden, um das Bildschirmaktualisieren wieder zu aktivieren.

```
'=====
' Auf CD      Buchdaten\Beispiele\07_Objekte\
' Dateiname  07_04_ClipArt.xls
' Modul      mdl_01_ClipArt
'=====
```

```
Sub RGBColor()
    Dim bytQuestion As Byte
    Dim strFile As String
    Dim strPath As String
    Dim pic As Picture
    Dim shp As Shapes
    Dim dbl As Double
    Dim dblColor As Double
    Dim dblRed As Double
    Dim dblGreen As Double
    Dim dblBlue As Double

    ' Bild auswählen
    bytQuestion = Application.InputBox _
        ("Welches Bild soll eingefügt werden?" _
        & vbCrLf & vbCrLf & _
        "1 = Tiger" & vbCrLf & _
        "2 = Rose", _
        Title:="Windows Meta File einfügen", _
        Default:=1, _
        Type:=1)

    ' Je nach Bildauswahl den Dateinamen übergeben
    If bytQuestion = 1 Then
        strFile = "\Tiger.wmf"
    ElseIf bytQuestion = 2 Then
        strFile = "\Rose.wmf"
    Else
        MsgBox "Falsche Eingabe"
        Exit Sub
    End If

    Set shp = ActiveSheet.Shapes

    ' Pfad zusammenfügen und prüfen,
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

```

' ob eine solche Datei existiert
strPath = ThisWorkbook.Path & strFile
If Dir(strPath) = "" Then Exit Sub

' Vorhandene Elemente, Farben und Werte löschen
ActiveSheet.DrawingObjects.Delete
Range("A:D").Clear

' Bild einfügen
Set pic = ActiveSheet.Pictures.Range("F1").Insert(strPath)

' Bild-Gruppierungen aufheben
shp(1).Ungroup.Ungroup

For dbl = 1 To shp.Count
    ' RGB-Farben ermitteln
    dblColor = shp(dbl).Fill.ForeColor.RGB
    dblBlue = Int(dblColor / 65536)
    dblGreen = Int((dblColor - (dblBlue * 65536)) / 256)
    dblRed = dblColor Mod 256

    ' Neue Farben an Bildelemente übergeben
    shp(dbl).Fill.ForeColor.RGB = _
        RGB(dblGreen, dblRed, dblBlue)

    ' Spaltenüberschriften aufbereiten
    Cells(1, 1).Value = "Farbe"
    Cells(1, 2).Value = "Rot"
    Cells(1, 3).Value = "Grün"
    Cells(1, 4).Value = "Blau"

    ' Farben und RGB-Werte an Zellen übergeben
    Cells(dbl + 1, 1).Interior.Color = _
        RGB(dblRed, dblGreen, dblBlue)
    Cells(dbl + 1, 2).Value = dblRed
    Cells(dbl + 1, 3).Value = dblGreen
    Cells(dbl + 1, 4).Value = dblBlue
Next dbl

Set shp = Nothing
Set pic = Nothing
End Sub

```

236 Eine schematische Darstellung oder ein Organigramm erzeugen

Seit der Version XP (Version 2002) steht in Excel in der Symbolleiste ZEICHNEN ein neues Objekt zur Verfügung. Dieses Objekt nennt sich auf Deutsch »Schematische Darstellung oder Diagramm«. Sie finden es in der Symbolleiste ZEICHNEN oder im Menü EINFÜGEN | SCHEMATISCHE DARSTELLUNG. Sie können damit eine spezielle Sorte an Diagrammen erstellen, unter anderem auch Organisationsdiagramme.



Abbildung 183: Schematische Darstellung oder Organigramm

Sie werden in VBA mit verschiedenen neuen Objekten, Eigenschaften und Methoden konfrontiert, die ausschließlich für das Erstellen von schematischen Darstellungen zur Verfügung stehen. Die Methode, um ein schematisches Diagramm zu erstellen, nennt sich `AddDiagram`. Es wird dabei ein Objekt vom Typ `Shape` erzeugt. Die Syntax dazu lautet:

```
Ausdruck.AddDiagram(Type, Left, Top, Width, Height)
```

Für den Typ `Type` stehen insgesamt sieben verschiedene Konstanten zur Auswahl:

AddDiagram-Konstanten	Beschreibung
<code>msoDiagramCycle</code>	Ein Prozessdiagramm mit dem Diagrammtyp des fortlaufenden Zyklus
<code>msoDiagramMixed</code>	Ein gemischter Diagrammtyp
<code>msoDiagramOrgChart</code>	Ein Diagrammtyp mit einer hierarchischen Beziehung
<code>msoDiagramPyramid</code>	Ein Diagrammtyp mit einer Beziehung, die auf einer Gründung basiert
<code>msoDiagramRadial</code>	Ein Diagrammtyp, der die Beziehungen eines Kernelements anzeigt
<code>msoDiagramTarget</code>	Ein Diagrammtyp, der Schritte zu einem Ziel anzeigt
<code>msoDiagramVenn</code>	Ein Diagrammtyp, der überlappende Bereiche zwischen Elementen anzeigt

Tabelle 51: Konstanten zur Methode `AddDiagram`

Wenn Sie versuchen, das Erstellen eines schematischen Diagramms mit dem Makro-Rekorder aufzuzeichnen, werden Sie die erste Enttäuschung erleben, denn diesen Schritt lässt der Makro-Rekorder nicht zu. Die nächste Enttäuschung werden Sie erleben, wenn Sie versuchen, mit dem Makro-Rekorder ein bestehendes Diagramm zu verändern, denn auch hier tut sich nicht viel. Ein Blick in die Online-Hilfe von Excel VBA ist unerlässlich. Dabei ist es natürlich von Vorteil, wenn man die Methoden und Eigenschaften dazu kennt. Die nachfolgenden Rezepte werden Ihnen diese Hinweise liefern.

Grundlagen

Allgemein

Datum und Zeit

Formeln

Steuerelemente

Befehle

Objekte

Diagramme

Ereignisse

User-Formen

Web/Mail

Extern

Gemischtes

Spezial

Achtung

Wenn Sie mit der Excel-Version XP arbeiten, werden Sie sehr schnell feststellen, dass dieses neue Feature noch nicht ausgereift ist, denn es kann vorkommen, dass sich Excel hin und wieder komplett verabschiedet und neu gestartet werden muss.

Unsere Tests haben gezeigt, dass in der Excel-Version 2003 das Arbeiten mit Organisations-Diagrammen etwas stabiler ist. Es ist jedoch ganz offensichtlich immer noch nicht möglich, per VBA beispielsweise einen Text automatisch einem Shape zuzuweisen. Die Eingabe des Stichwortes »TextShape« in die Online-Hilfe präsentiert zwar ein sehr schönes Beispiel, wenn dieses jedoch in ein Modul kopiert und ausgeführt wird, antwortet der Debugger mit einer Fehlermeldung. In der Knowledge-Base Nr. 317293 auf der Microsoft-Webseite wird dies auch bestätigt.

Falls Sie trotz der absehbaren Probleme ein schematisches Diagramm per VBA erstellen möchten, raten wir Ihnen ganz dringend, vor dem Ausführen eines solchen Codes sämtliche Mappen zu speichern. Wenn Sie dies nicht tun und Excel nicht ordnungsgemäß beendet wird, dann ist ein Datenverlust nicht auszuschließen.

Beim Erstellen von Organisations-Diagrammen können die verschiedensten Variationen erzeugt werden. Dabei wird mit der Eigenschaft `Children` und der Methode `AddNode` gearbeitet. Die beiden Befehle erzeugen zusammen je einen Diagramm-Unterpunkt. Die Befehle können in beliebiger Konstellation zusammengesetzt werden. Um den einzelnen Punkten weitere Unterpunkte anzuhängen, arbeiten wir mit der Eigenschaft `Item`, gefolgt vom Index in runden Klammern. Wichtig dabei ist zu wissen, welches Item, also welches Feld welchen Index trägt (siehe Abbildung 184).

Beim Erstellen des Diagramms mittels einer Schleife sieht es zuerst aus, als wäre der Index wirr durcheinander (siehe dritte Ebene). Dies ist auf den zweiten Blick jedoch nicht der Fall. Beim ersten Durchlauf der Schleife wird jedem Feld der zweiten Ebene *ein* Element angehängt. Beim zweiten Durchlauf der Schleife wird jedem Feld der zweiten Ebene ebenfalls nur *ein* Element angehängt usw. Wenn das Diagramm später mit einem anderen Code verändert wird, wie zum Beispiel, indem ein Diagrammunterpunkt gelöscht wird, dann entspricht die Nummerierung nicht mehr der Abbildung, sondern ist fortlaufend der Reihe nach (von links nach rechts und von oben nach unten).

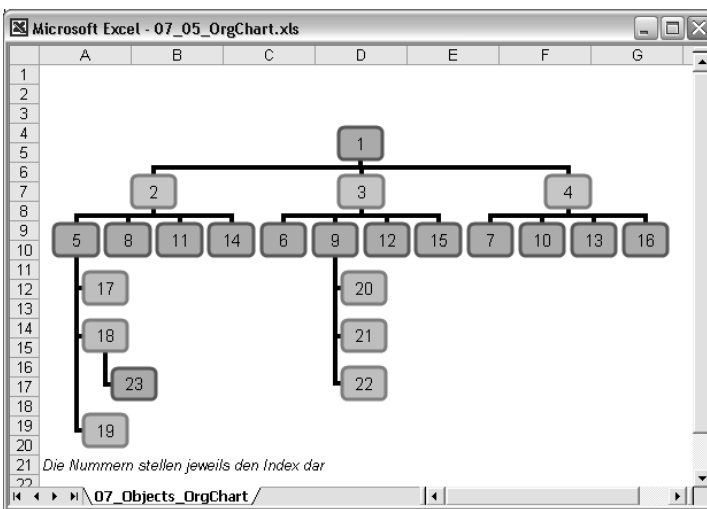


Abbildung 184: Ein Organisations-Diagramm mit Angabe der Index-Nummern

In unserem Beispiel werden wir ein Organisations-Diagramm einfügen. Nach der Variablen-Deklaration wird das Diagramm erstellt (`AddDiagram`). Dieser Befehl alleine erzeugt jedoch nur ein leeres Shape, also den Rahmen, in dem das Organisations-Diagramm aufgebaut wird. Wir müssen das Organisations-Diagramm nun nach und nach aufbauen.

Zuerst wird die oberste Ebene mit einem einzelnen Feld erzeugt (Index 1). Dann wird unter Zuhilfenahme einer Schleife die zweite Ebene erzeugt (Index 2 bis 4). Die dritte Ebene wird ebenfalls mit einer Schleife erzeugt. An jedes der Felder (`Item`) der zweiten Ebene werden vier weitere Felder angehängt (Index 5 bis 16). Im letzten Codeblock wird eine vierte (Index 17 bis 22) und eine fünfte Ebene (Index 23) erzeugt. Beachten Sie hier die Kombinationen von `Children` und `AddNode`, diese können verschieden zusammengestellt werden.

Hinweis

Wie Sie unschwer erkennen können, ist das Programmieren von schematischen Diagrammen noch sehr unausgereift und verwirrend. Da schematische Diagramme seit der Version XP verfügbar sind, wollten wir Ihnen trotz allem dieses eine Beispiel nicht vorenthalten.

```
'=====
' Auf CD      Buchdaten\Beispiele\07_Objekte\
' Dateiname   07_05_OrgChart.xls
' Tabelle     07_Objects_CreateOrgChart
' Modul       mdl_01_CreateOrgChart
'=====

Sub AddOrgChart()
    Dim shpChart As Shape
    Dim dgnNode As DiagramNode
    Dim shp As Shape
    Dim i As Integer

    ' Organisations-Chart einfügen (leeres Shape)
    Set shpChart = ActiveSheet.Shapes.AddDiagram _
        (Type:=msoDiagramOrgChart, _
        Left:=10, _
        Top:=0, _
        Width:=400, _
        Height:=250)

    ' den obersten Diagrammknoten einfügen
    Set dgnNode = shpChart.DiagramNode.Children.AddNode

    ' Drei Unterknoten erstellen
    For i = 1 To 3
        dgnNode.Children.AddNode
    Next i

    ' Vier weitere Knoten an jeden der drei Unterknoten anhängen
    For i = 1 To 4
        With ActiveSheet.Shapes(1).Diagram.Nodes
            .Item(2).Children.AddNode
            .Item(3).Children.AddNode
            .Item(4).Children.AddNode
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

```

        End With
    Next i

    ' Weitere Unterpunkte einfügen
    With ActiveSheet.Shapes(1).Diagram.Nodes
        .Item(5).Children.AddNode.AddNode.AddNode
        .Item(9).Children.AddNode.AddNode.AddNode
        .Item(18).Children.AddNode
    End With

    Set shpChart = Nothing
    Set dgnNode = Nothing
End Sub

```

237 Diagrammknoten löschen

Wie bereits erwähnt, ist ein schematisches Diagramm ein Shape. Der Begriff Shape bedeutet ins Deutsche übersetzt: Form oder Gestalt. Wenn Sie das gesamte Diagramm löschen möchten, verwenden Sie die Anweisung `Shapes.Delete`. Damit in Ihrem Tabellenblatt nur die schematischen Diagramme und nicht auch andere Shapes gelöscht werden, können Sie mittels der Eigenschaft `HasDiagram` prüfen, ob es sich bei dem Objekt um ein schematisches Diagramm handelt.

```

' =====
' Auf CD      Buchdaten\Beispiele\07_Objekte\
' Dateiname   07_05_OrgChart.xls
' Tabelle     07_Objects_DeleteOrgCharts
' Modul       mdl_03_DeleteOrgCharts
' =====

Sub DeleteOrgCharts()
    Dim shp As Shape
    For Each shp In ActiveSheet.Shapes
        If shp.HasDiagram Then
            shp.Delete
        End If
    Next shp
End Sub

```

Um einzelne Felder `Item` des Diagramms zu löschen, können diese über den Index gefolgt von der Methode `Delete` angesprochen werden. Denken Sie dabei daran, dass die Nummerierung des Index jetzt nicht mehr der Nummerierung gemäß Abbildung 184 entspricht, sondern fortlaufend ist.

```

' =====
' Auf CD      Buchdaten\Beispiele\07_Objekte\
' Dateiname   07_05_OrgChart.xls
' Tabelle     07_Objects_DeleteOrgCharts
' Modul       mdl_03_DeleteOrgCharts
' =====

Sub DeleteDiagramNodes()
    With ActiveSheet.Shapes(1).Diagram.Nodes

```

```
        .Item(7).Delete
        .Item(21).Delete
    End With
End Sub
```

238 Objekte löschen

Um nur ein einzelnes Objekt in Ihrem Tabellenblatt zu löschen (Delete), können Sie dieses über den Index oder dessen Namen ansprechen. Den Namen eines Objektes erfahren Sie, indem Sie das Objekt aktivieren. Dem Feld links neben der Bearbeitungszeile ist der Name zu entnehmen. Falls Sie nicht mit dem Index, sondern mit dem Namen arbeiten möchten, schreiben Sie diesen in das runde Klammernpaar nach Shapes. Ein Name muss immer zwischen Anführungs- und Schlusszeichen ("") geschrieben werden, zum Beispiel: Shapes("Bild 1").Delete.

```
'=====
' Auf CD      Buchdaten\Beispiele\07_Objekte\
' Dateiname   07_06_DeleteObjects.xls
' Modul       mdl_01_DeleteObjects
'=====

Sub DeleteOneShape()
    ActiveSheet.Shapes(1).Delete
End Sub
```

Um *sämtliche* eingefügten Objekte in einem Tabellenblatt zu löschen, verwenden Sie das Objekt Shape in Kombination mit der Methode Delete. Hier funktioniert das direkte Ansprechen nicht. Es muss mit einer Schleife gearbeitet werden.

```
'=====
' Auf CD      Buchdaten\Beispiele\07_Objekte\
' Dateiname   07_06_DeleteObjects.xls
' Modul       mdl_01_DeleteObjects
'=====

Sub DeleteAllShapes()
    Dim shp As Shape

    For Each shp In ActiveSheet.Shapes
        shp.Delete
    Next shp
End Sub
```

Wenn Sie ohne Schleife arbeiten möchten, dann können Sie alternativ auch das Objekt Drawing-Objects verwenden.

```
'=====
' Auf CD      Buchdaten\Beispiele\07_Objekte\
' Dateiname   07_06_DeleteObjects.xls
' Modul       mdl_01_DeleteObjects
'=====
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

```
Sub DeleteAllPictures()  
    ActiveSheet.DrawingObjects.Delete  
End Sub
```

239 Nur Objekte von einem bestimmten Typ löschen

Im Rezept 238 haben Sie drei Prozeduren gesehen, mit denen Sie Objekte löschen können. Wir wollen Ihnen hier zudem zeigen, wie Sie nur Objekte eines bestimmten Typs löschen können. Zum Beispiel nur WordArt-Objekte oder nur Objekte aus der Symbolleiste STEUERELEMENT-TOOLBOX oder Elemente aus der Symbolleiste FORMULAR.

In der folgenden Prozedur werden nur AutoFormen und Linien aus der Symbolleiste ZEICHNEN gelöscht. Wir geben dabei die Konstante des Objekts an: `Type = msoAutoShape` und `Type = msoLine`.

```
'=====
' Auf CD      Buchdaten\Beispiele\07_Objekte\
' Dateiname   07_06_DeleteObjects.xls
' Modul       mdl_01_DeleteObjects
'=====

Sub DeleteObjects()
    Dim shp As Shape

    ' Nur AutoFormen und Linien entfernen
    For Each shp In ActiveSheet.Shapes
        If shp.Type = msoAutoShape Or _
           shp.Type = msoLine Then
            shp.Delete
        End If
    Next shp
End Sub
```

Der Tabelle 52 können Sie die verfügbaren Konstanten zu den Objekten entnehmen:

Objekt	Index	Type-Konstanten
AutoFormen	1	msoAutoShape
Callout	2	msoCallout
Canvas (Nur Word)	20	msoCanvas
Diagramme	3	msoChart
Kommentare	4	msoComment
Eingebettete OLE-Objekte	7	msoEmbeddedOLEObject
Formular-Elemente	8	msoFormControl
Freihandformen	5	msoFreeform
Gruppen	6	msoGroup

Tabelle 52: Konstanten zu den Objekt-Typen

Objekt	Index	Type-Konstanten
Linien und Pfeile	9	msoLine
Verlinkte OLE-Objekte	10	msoLinkedOLEObject
Verlinkte Bilder	11	msoLinkedPicture
Media-Daten	16	msoMedia
Steuerelemente (Steuerelement-Toolbox)	12	msoOLEControlObject
Grafiken	13	msoPicture
Platzhalter	14	msoPlaceholder
Anker	18	msoScriptAnchor
Gemischte Typen	-2	msoShapeTypeMixed
Tabellen	19	msoTable
Textboxen	17	msoTextBox
Texteffekte (WordArt)	15	msoTextEffect

Tabelle 52: Konstanten zu den Objekt-Typen (Forts.)

240 Objekt-Typ ermitteln

Wenn Sie, wie im Rezept 239, Objekte eines bestimmten Typs löschen oder sonst ansprechen möchten, müssen Sie den Typ bzw. die zugehörige Konstante oder den Index kennen. Um diese zu ermitteln, muss jedes Objekt in dem Tabellenblatt angesprochen und dessen Typ abgefragt werden. Es ist unumgänglich, dabei jede Konstante einzeln abzufragen.

Hinweis

Die nachfolgende Prozedur zeigt, wie dies umgesetzt werden kann. Das Codebeispiel, das hier abgedruckt ist, wurde gekürzt. In der Beispieldatei auf der beiliegenden CD finden Sie den gesamten VBA-Code.

In einer `For`-Schleife werden sämtliche Objekte, die sich im aktiven Tabellenblatt befinden, angesprochen. Mittels `Select Case` wird der Typ und die Konstante abgefragt. Bei Zutreffen wird der Name der Konstanten in die Variable `strType` geschrieben. Vor dem Ende der `With`-Anweisung werden die Werte übergeben. Die Übergabe erfolgt an die Zelle, in der sich das Objekt befindet. Um den Wert zu sehen, müssen Sie gegebenenfalls das Objekt zur Seite ziehen.

```
'=====
' Auf CD      Buchdaten\Beispiele\07_Objekte\
' Dateiname   07_06_DeleteObjects.xls
' Modul       mdl_02_GetObjectType
'=====

Sub GetObjectType()
    Dim shp As Shapes
    Dim i As Integer
    Dim strType As String

    Set shp = ActiveSheet.Shapes
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

```

For i = 1 To shp.Count
  With ActiveSheet.Shapes(i)
    Select Case .Type
      Case msoAutoShape
        strType = "msoAutoShape"
      Case msoCallout
        strType = "msoCallout"
      Case msoCanvas
        strType = "msoCanvas"
      Case msoChart
        strType = "msoChart"
      Case msoComment
        strType = "msoComment"
      Case msoFreeform
        strType = "msoFreeform"
      Case msoGroup
        strType = "msoGroup"
      Case msoEmbeddedOLEObject
        strType = "msoEmbeddedOLEObject"
      Case msoFormControl
        strType = "msoFormControl"
      Case msoLine
        strType = "msoLine"

      ' [...]

      Case msoShapeTypeMixed
        strType = "msoShapeTypeMixed"
      Case Else
        strType = "Unbekannter Typ"
    End Select

    ' Die ermittelten Werte werden die Zelle, in der sich
    ' unter dem Objekt befindet, geschrieben
    .TopLeftCell = "Konstante: " & strType & _
                  " / Index: " & .Type

  End With
Next i
End Sub

```

241 Nur bestimmte Objekte gruppieren (Array)

Wie Sie im Rezept 240 erfahren haben, ist es relativ einfach, Objekte eines bestimmten Typs zu löschen. Objekte eines bestimmten Typs zu selektieren (Select) oder zu gruppieren (Group), ist schon etwas schwieriger, denn wir müssen mit einem Array arbeiten.

Wenn Sie das Selektieren von verschiedenen Objekten mit dem Makro-Rekorder aufzeichnen, können Sie sehen, dass dabei ebenfalls ein Array entsteht. Das Array könnte beispielsweise so aussehen:

```

ActiveSheet.Shapes.Range _
(Array("Line 13", "Oval 12", "Rectangle 11")).Select

```

Wenn jedoch viele solcher Shapes angesprochen werden müssen, wäre es mühsam, ein solches Array manuell zusammenzustellen. Insbesondere, da sich die Anzahl der Shapes verändern könnte. Wenn ein Shape hinzukommt oder wegfällt, müsste die Prozedur bereits wieder angepasst werden. In unserem Codebeispiel werden wir diesen Vorgang automatisieren.

Die Variable `varArr()` wird als Array von unbekannter Größe dimensioniert. Dies ist an den runden leeren Klammern zu erkennen. Die Größe des Arrays ist zu Beginn unbekannt, da wir die Anzahl an Shapes, die übergeben werden sollen, nicht kennen.

In einer `With`-Anweisung werden wir die Anzahl der insgesamt im Tabellenblatt vorhandenen Shapes zählen (`Count`). Diesen Wert übergeben wir an die Variable `strShp`. Wenn der Wert der Variablen größer als 1 (`> 1`) ist, wird die Prozedur fortgesetzt. Die Größe des Arrays kann nun festgelegt werden, denn die Anzahl der Shapes ist jetzt bekannt (`strShp`).

Die Größe des Arrays wird mit der Anweisung `ReDim` festgelegt. `ReDim` steht für `Re-Dimensionalisierung`, also für das Festlegen einer neuen Dimension. Damit ist die Größe des Arrays gemeint. Die Größe ist von 1 bis zu dem Wert, den die Variable `intShp` enthält. Das bedeutet, dass in jedem Array-Feld ein Shape aufgenommen werden kann. Wenn fünf Shapes gezählt wurden, dann sieht das Array wie folgt aus:

```
intArr(1 to 5)
```

Da wir nur Shapes vom Typ `AutoForm` (`msoAutoShape`) ermitteln möchten, werden wir innerhalb einer `For`-Schleife jedes Shape nach dessen Typ befragen. Wenn das Shape vom Typ `AutoForm` ist, wird der Name des Objektes an das Array übergeben (`Item(i).Name`).

Wenn in Ihrem Tabellenblatt nun beispielsweise fünf Shapes enthalten sind und nur zwei davon sind vom Typ `AutoForm`, könnte das Array wie folgt aussehen:

```
(Array("Line 13", "", "Oval 12", "", "Rectangle 11"))
```

Es sind also die Namen von drei Shapes im Array enthalten, die dem Typ `AutoForm` entsprechen. An der Stelle, wo im Tabellenblatt zwar ein Shape gefunden wurde, dieses jedoch nicht dem Typ `AutoForm` entspricht, werden hier symbolisch leere Felder (`""`) dargestellt. Innerhalb der Schleife wird zudem ein Zähler geführt (`intAutoShp`). Bei jedem Treffer wird der Zähler um eins erhöht. Diesen Zähler verwenden wir später, um das Array neu zu dimensionieren.

Nach der Schleife wird in einer `If`-Entscheidung geprüft, ob überhaupt ein Shape vom Typ `AutoForm` gefunden wurde (`> 1`). Wenn dies zutrifft, wird das Array neu dimensioniert. Dies geschieht über die Anweisung `ReDim Preserve`. `Preserve` heißt ins Deutsche übersetzt »behalten« oder »erhalten«. Das heißt, der gefüllte Inhalt des Arrays soll trotz Neudimensionierung erhalten bleiben. Wir benötigen nun jedoch nicht mehr alle Datenfelder, sondern nur noch jene, welche gefüllt sind. Wie viele das sind, haben wir ja mittels der Variablen `intAutoShp` ermittelt. Wenn wir von unserer obigen Codezeile ausgehen, sieht das Array nur noch wie folgt aus:

```
(Array("Line 13", "Oval 12", "Rectangle 11"))
```

Die `Re-Dimensionalisierung` hat somit bewirkt, dass die leeren Felder verschwunden sind. Das `Preserve` hat die nicht leeren Werte übernommen. Über die Anweisung `Range(varArr).Select` können nun die Shapes vom Typ `AutoForm`en selektiert werden. Die Anweisung würde somit wie folgt aussehen:

```
Range(Array("Line 13", "Oval 12", "Rectangle 11")).Select
```

Grundlagen
Allgemein
Datum und Zeit
Formeln
Diagramme
Steuerungselemente
Befehlszeile
Objektmodelle
Diagramme
Ereignisse
User-Formen
Web/Mail
Extern
Gemischtes
Spezial

Falls Sie die AutoFormen gruppieren möchten, dann wählen Sie an Stelle der Methode Select die Methode Group.

```
'=====
' Auf CD      Buchdaten\Beispiele\07_Objekte\
' Dateiname  07_06_DeleteObjects.xls
' Modul      mdl_03_GroupObjects
'=====

Sub GroupObjects()
    Dim intShp As Integer
    Dim intAutoShp As Integer
    Dim varArr() As Variant
    Dim i As Integer
    Dim rngShp As Range

    With ActiveSheet.Shapes
        ' Vorhandene Objekte zählen
        intShp = .Count
        If intShp > 1 Then
            intAutoShp = 0

            ' Die Größe des Arrays festlegen
            ReDim varArr(1 To intShp)

            For i = 1 To intShp
                ' Wenn ein Objekt vom Type AutoForm gefunden wird,
                ' dann den Namen des Objektes an das Array übergeben
                If .Item(i).Type = msoAutoShape Then
                    intAutoShp = intAutoShp + 1
                    varArr(intAutoShp) = .Item(i).Name
                End If
            Next

            If intAutoShp > 1 Then
                ' Dem Array eine neue Größe zuweisen
                ' Die Größe entspricht der Anzahl der gefundenen
                ' AutoFormen
                ReDim Preserve varArr(1 To intAutoShp)
                .Range(varArr).Select
            End If
        End With
    End Sub
```

242 Der Office-Assistent

Office-Assistenten sind die animierten Figuren, die Ihnen ihre Hilfe in Form von Sprechblasen anbieten. Keine Frage, die Assistenten sind umstritten. Die einen mögen sie, den anderen wiederum gehen sie auf die Nerven. Nichtsdestotrotz: Sie können den Assistenten programmieren, so dass er Ihnen den gewünschten Text anzeigt und vieles mehr, wie Sie noch erfahren werden.

Damit ein Assistent angezeigt werden kann, muss er installiert und aktiviert sein. Falls Sie bei der Installation Ihres Office-Paketes den Standard-Speicherort verwendet haben, finden Sie die Assistenten im Verzeichnis: C:\Programme\Microsoft Office\Office10. Falls nicht, finden Sie im Rezept 7.30 eine Beschreibung, wie Sie ermitteln können, wo sich Ihr Assistent befindet. Die Assistenten sind vom Dateityp *.acs (MS Agent Character File). Falls Sie in dem genannten Verzeichnis die Assistenten nicht finden sollten, suchen Sie am besten mittels der Windows-Suchmaschine nach Dateien mit der entsprechenden Endung.

Sie können per VBA prüfen, ob der Assistent installiert ist.

```
'=====
' Auf CD      Buchdaten\Beispiele\07_Objekte\
' Dateiname  07_07_Assistant.xls
' Modul      mdl_01_Check
'=====

Sub AssistantCheck()
  If objAssistant Is Nothing Then
    MsgBox "Keine oder fehlerhafte Installation."
  Else
    MsgBox "Der Assistent ist korrekt installiert."
  End If
End Sub
```

Im Office-Paket stehen insgesamt acht Assistenten zur Verfügung. Welche das sind und welchen Dateinamen sie tragen, können Sie der Tabelle 53 entnehmen. In der Version 2000 stand noch der Professor (*Genius.acs*) zur Verfügung. Dieser wurde jedoch in der Version XP durch den Magier (*Merlin.acs*) ersetzt.

Assistent, Name des Assistenten und Dateiname			
			
Karl Klammer Clippit.acs	Hüpfer Dot.acs	F1 F1.acs	Office Logo Logo.acs
			
Merlin Merlin.acs	Mutter Natur Mnature.acs	Minki Offcat.acs	Rocky Rocky.acs

Tabelle 53: Die Office-Assistenten

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

Um manuell einen anderen Assistenten auszuwählen, müssen Sie zuerst sicherstellen, dass dieser eingeblendet ist. Wenn der Assistent ausgeblendet ist, blenden Sie ihn ein, indem Sie in der Menüleiste auf das Fragezeichen klicken und den Eintrag OFFICE-ASSISTENTEN ANZEIGEN wählen. Wenn der Office-Assistent eingeblendet ist, klicken Sie ihn mit der rechten Maustaste an, um das Kontextmenü zu öffnen. Klicken Sie im Kontextmenü auf den Eintrag ASSISTENTEN WÄHLEN. Auf der Registerkarte KATALOG sind die Schaltflächen <ZURÜCK und WEITER> zu finden. Mit Hilfe dieser Schaltflächen können Sie zwischen den verfügbaren Assistenten wechseln.

Um den gewünschten Assistenten per VBA auszuwählen und anzuzeigen, verwenden Sie die Eigenschaft `Assistant`, gefolgt von den Eigenschaften `Filename` für den Dateinamen und `Visible`, um sicherzustellen, dass er eingeblendet wird. Eine Pfadangabe ist nur dann erforderlich, wenn sich der Assistent nicht im Standardordner der Installation befindet.

```
' =====
' Auf CD      Buchdaten\Beispiele\07_Objekte\
' Dateiname   07_07_Assistant.xls
' Modul       mdl_02_Show
' =====

Sub AssistantShow()
    With Assistant
        .Filename = "Merlin.acs"
        .Visible = True
    End With
End Sub
```

Die nachfolgende Liste ist eine Zusammenstellung über das, was der Assistent anzeigen kann. In den nachfolgenden Rezepten werden wir näher auf diese Punkte eingehen.

- ▶ Überschrift und Text anzeigen
- ▶ Verschiedene Schaltflächen anzeigen
- ▶ Aufzählungen anzeigen
- ▶ Nummerierungen anzeigen
- ▶ Kontrollkästchen zur Auswahl anbieten
- ▶ Bilder anzeigen
- ▶ Einen Hyperlink enthalten, der zu einer Webseite führt
- ▶ Animationen ausführen

243 Einen eigenen Titel und Text in der Sprechblase ausgeben

In diesem Beispiel wollen wir einen eigenen Titel und einen eigenen Text in der Sprechblase des Assistenten anzeigen lassen. Die Schaltfläche OK wird automatisch angezeigt. Es können auch andere Schaltflächen angezeigt werden. Mehr zum Thema »Schaltflächen« erfahren Sie im Rezept 7.31.

Zu Beginn jeder Prozedur, in Bezug auf den Assistenten, stellen wir sicher, dass der Assistent angezeigt wird mit `Visible = True`. Dies, um eine Fehlermeldung auszuschließen. Um eine neue Sprechblase zu erzeugen, verwenden wir die Eigenschaft `NewBalloon`. Innerhalb der `With`-Anweisung weisen wir der Überschrift `Header` einen beliebigen Text zu. Danach erzeugen wir den Text `Text` und schließlich zeigen wir die Sprechblase an (`Show`).



Abbildung 185: Dem Assistenten Überschrift und Text zuweisen

```
'=====
' Auf CD      Buchdaten\Beispiele\07_Objekte\
' Dateiname  07_07_Assistant.xls
' Modul      mdl_03_HeadText
'=====

Sub HeaderAndText()
    ' Den Assistenten aktivieren
    Assistant.Visible = True

    ' Eine neue Sprechblase erzeugen
    With Assistant.NewBalloon
        ' Überschrift
        .Heading = "Hallo Besucher"
        ' Text
        .Text = "Ich bin der Office-Assistent"
        ' Sprechblase anzeigen
        .Show
    End With
End Sub
```

244 Den Pfad des Assistenten ausgeben

Falls Sie nicht sicher sind, wo auf Ihrer Festplatte sich die Assistenten befinden, können Sie dies durch den Assistenten selbst anzeigen lassen. In unserem Beispiel zeigen wir in der Sprechblase die Pfadangabe an.



Abbildung 186: Sprechblase mit Fragezeichen-Symbol und Pfadangabe

Neu in dem Beispiel ist, dass zudem ein Fragezeichen-Symbol `Icon` eingeblendet wird. Dazu verwenden wir die Konstante `msoIconAlertQuery`. Neben dieser Konstante stehen noch weitere zur

Grundlagen
Allgemein
Datum und Zeit
Diagramme
Steuerelemente
Befehlsliste
Objekt
Diagramm
Ereignisse
User-Form
Web/Mail
Extern
Gemischtes
Spezial

Verfügung. Welche das sind, können Sie der Tabelle 54 entnehmen. Am Ende der Prozedur wird der `Visible`-Wert auf `False` gesetzt. Damit wird der Assistent ausgeblendet, sobald die Schaltfläche OK in der Sprechblase angeklickt wird. Die Schaltfläche OK wird automatisch angezeigt. Dazu ist keine eigene Codezeile erforderlich.

```
'=====
' Auf CD      Buchdaten\Beispiele\07_Objekte\
' Dateiname  07_07_Assistant.xls
' Modul      mdl_04_Path
'=====

Sub AssistantPath()
    With Assistant
        .Visible = True

        With .NewBalloon
            .Heading = "Wo komme ich her?"
            .Text = Assistant.Filename
            ' Ein Fragezeichen-Symbol anzeigen
            .Icon = msoIconAlertQuery
            .Show
        End With

        ' Den Assistenten wieder ausblenden
        .Visible = False
    End With
End Sub
```

Hinweis

Sie können alternativ zu einer Konstanten auch den Index verwenden. Der Vorteil beim Einsetzen von Konstanten besteht darin, dass in Textform zu sehen ist, um welche Schaltflächen es sich handelt. Beim Index muss man die Nummer kennen. Ein Index ist somit nicht so aussagekräftig wie eine Konstante.

Symbol	Icon-Konstanten	Index
	<code>msoIconAlert</code>	2
	<code>msoIconAlertCritical</code>	7
	<code>msoIconAlertInfo</code>	4
	<code>msoIconAlertQuery</code>	6

Tabelle 54: Die verfügbaren Icons

Symbol	Icon-Konstanten	Index
	msoIconAlertWarning	5
Kein Symbol	msoIconNone	0
	msoIconTip	3

Tabelle 54: Die verfügbaren Icons (Forts.)

245 Schaltflächen in einer Sprechblase

Sie haben die Möglichkeit, verschiedene Kombinationen an Schaltflächen anzuzeigen. Die Eigenschaft für Schaltflächen nennt sich `Button`, gefolgt von der gewünschten Konstanten. Wir verwenden für unsere Prozedur die Konstante `msoButtonSetOkCancel`. Damit werden die Schaltflächen OK und ABBRECHEN angezeigt.



Abbildung 187: Eine Sprechblase mit den gewünschten Schaltflächen

Um die Methode `Show` herum bauen wir eine `If`-Entscheidung auf. Die Entscheidung prüft, welche Schaltfläche angeklickt wurde, um dann entsprechend darauf zu reagieren. In unserem Beispiel wird bei Klick auf die Schaltfläche OK den selektierten Zellen eine rote Hintergrundfarbe zugewiesen. Bei Klick auf die Schaltfläche ABBRECHEN wird die Prozedur ohne etwas auszuführen verlassen. Die Schaltfläche ABBRECHEN muss nicht programmiert werden, so wie das zum Beispiel bei `MsgBox` der Fall ist.

Bei Eingabe der Methode `Show`, gefolgt von einem Gleichheitszeichen (`=`), wird per IntelliSense automatisch die verfügbare Liste der Konstanten angezeigt.

```
'=====
' Auf CD      Buchdaten\Beispiele\07_Objekte\
' Dateiname  07_07_Assistant.xls
' Modul      mdl_05_Button
'=====

Sub AssistantButton()
    With Assistant
        .Visible = True

        With .NewBalloon
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

```
.Heading = "Schaltflächen"
.Text = "Möchten Sie die markierten Zellen rot einfärben?"
.Icon = msoIconAlertQuery

' Schaltflächen "OK" und "Abbrechen" festlegen
.Button = msoButtonSetOkCancel

' Wenn "OK" geklickt wird, dann die Zellen rot einfärben,
' Wenn "Abbrechen" geklickt wird, dann die Prozedur
' verlassen
If .Show = msoBalloonButtonOK Then
    Selection.Interior.ColorIndex = 3
Else
    Exit Sub
End If
End With

End With
End Sub
```

Der nachfolgenden Tabelle können Sie die in der Sprechblase verfügbaren Schaltflächen oder Schaltflächenkombinationen entnehmen.

Schaltflächen	Button-Konstanten	Index
Abbrechen, Wiederholen, Ignorieren	msoButtonSetAbortRetryIgnore	10
Zurück, Schließen	msoButtonSetBackClose	6
Zurück, Weiter, Schließen	msoButtonSetBackNextClose	8
Zurück, Weiter, Erneut erinnern	msoButtonSetBackNextSnooze	12
Abbrechen	msoButtonSetCancel	2
Weiter, Schließen	msoButtonSetNextClose	7
Keine Schaltfläche	msoButtonSetNone	0
OK	msoButtonSetOK	1
OK, Abbrechen	msoButtonSetOkCancel	3
Wiederholen, Abbrechen	msoButtonSetRetryCancel	9
Suchen, Schließen	msoButtonSetSearchClose	11
Tipps, Optionen, Schließen	msoButtonSetTipsOptionsClose	13
Ja, Ja alle, Nein, Abbrechen	msoButtonSetYesAllNoCancel	14
Ja, Nein	msoButtonSetYesNo	4
Ja, Nein, Abbrechen	msoButtonSetYesNoCancel	5

Tabelle 55: Schaltflächen für die Sprechblase

246 Eine Sprechblase mit Aufzählungspunkten

Sie können den Assistenten mit Aufzählungspunkten versehen. Den einzelnen Aufzählungspunkten können zudem verschiedene Prozeduren hinterlegt werden.

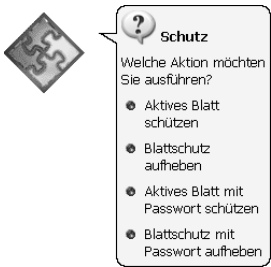


Abbildung 188: Eine Sprechblase mit Aufzählungszeichen

Achtung Es sind maximal fünf Aufzählungspunkte in einer Sprechblase zugelassen. Falls Sie mehr als fünf einfügen, wird der Debugger gestartet.

In unserem Beispiel arbeiten wir mit zwei Prozeduren. In der ersten Prozedur werden die Aufzählungspunkte `Label` festgelegt. Für jeden Punkt verwenden wir ein `Label` mit dem entsprechenden Index. Jedem der `Labels` weisen wir einen Text zu.

Wir verwenden in unserem Code zudem die Eigenschaft `Mode`, gefolgt von der Konstanten `msoModeless`. Damit wird bezweckt, dass trotz geöffneter Sprechblase an dem Tabellenblatt weitergearbeitet werden kann.

Die Eigenschaft `Callback` wird verwendet, um die zweite Prozedur aufzurufen, welche die Anweisungen für die einzelnen Aufzählungspunkte enthält.

```
'=====
' Auf CD      Buchdaten\Beispiele\07_Objekte\
' Dateiname  07_07_Assistant.xls
' Modul      mdl_06_Enumeration
'=====

Sub Enumeration()
    With Assistant
        .Visible = True

        With .NewBalloon
            .Heading = "Schutz"
            .Text = "Welche Aktion möchten Sie ausführen?"
            .Icon = msoIconAlertQuery

            ' Keine Schaltfläche anzeigen
            .Button = msoButtonSetNone

            ' Es kann trotz geöffneter Sprechblase gearbeitet werden
            .Mode = msoModeModeless

            ' Vier Aufzählungspunkte
            .Labels(1).Text = "Aktives Blatt schützen"
            .Labels(2).Text = "Blattschutz aufheben"
            .Labels(3).Text = "Aktives Blatt mit Passwort schützen"
```

Grundlagen
Allgemein
Datum und Zeit
Diagramme
Steuerelemente
Befehlsliste
Objekt
Diagramm
Ereignisse
User-Form
Web/Mail
Extern
Gemischtes
Spezial

```

.Labels(4).Text = "Blattschutz mit Passwort aufheben"

' Prozedur "MyColors" aufrufen
.Callback = "MyColors"
.Show
End With

End With
End Sub

```

Die Prozedur `MyColors` wird durch die Prozedur `Enumeration` aufgerufen. Hier wird jedem der Aufzählungspunkte eine Anweisung übergeben, so dass bei Klick auf einen der Punkte in der Sprechblase die gewünschte Aktion ausgeführt wird.

Am Ende der Prozedur wird die Methode `Close` ausgeführt. Sie bewirkt, dass die Sprechblase nach dem Ausführen der Prozedur geschlossen wird. Wenn Sie diese Methode weglassen, dann wird die Sprechblase so lange angezeigt, bis die Excel-Anwendung geschlossen wird.

```

Sub MyColors(bln As Boolean, lngChoice As Long, lngPriv As Long)
    Select Case lngChoice
        Case 1
            ActiveSheet.Protect
        Case 2
            ActiveSheet.Unprotect
        Case 3
            ActiveSheet.Protect Password:="Codebook"
        Case 4
            ActiveSheet.Unprotect Password:="Codebook"
    End Select
    ' Sprechblase wird nach Auswahl geschlossen
    ' Wenn dies nicht geschieht, dann bleibt die Sprechblase
    ' immer mit diesen Angaben geöffnet
    bln.Close
End Sub

```

247 Eine Sprechblase mit Nummerierung

Sie können nebst Aufzählungszeichen auch Nummerierungen verwenden. Diese werden beispielsweise dann eingesetzt, wenn Sie den Anwendern gewisse Schritte Ihrer Prozedur erläutern möchten.

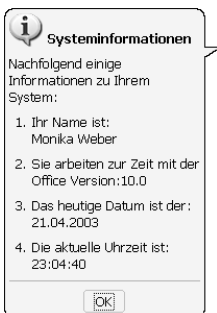


Abbildung 189: Eine Sprechblase mit Aufzählungszeichen

Achtung

Es können maximal fünf Nummerierungen in einer Sprechblase angezeigt werden. Falls Sie mehr einfügen, wird der Debugger gestartet.

In unserem Beispiel geben wir einige Systeminformationen in der Sprechblase aus. Wir verwenden insgesamt vier Aufzählungszeichen. Wie bereits im vorangegangenen Rezept, setzen wir dabei Labels ein, denen wir den gewünschten Text zuweisen. Noch bevor wir die Bezeichnungsfelder Label aufbereiten, stellen wir den BalloonType auf Nummern msoBalloonTypeNumbers um.

```
'=====
' Auf CD      Buchdaten\Beispiele\07_Objekte\
' Dateiname  07_07_Assistant.xls
' Modul      mdl_07_Numbering
'=====

Sub Numbering()
    With Assistant
        .Visible = True

        With .NewBalloon
            .Heading = "Systeminformationen"
            .Text = "Nachfolgend einige Informationen zu Ihrem System:"
            .Icon = msoIconAlertInfo

            ' Auf Nummerierung umstellen
            .BalloonType = msoBalloonTypeNumbers

            ' Vier Nummerierungspunkte
            .Labels(1).Text = "Ihr Name ist:" _
                            & vbCrLf & Application.UserName
            .Labels(2).Text = "Sie arbeiten zur Zeit mit der" _
                            & vbCrLf & "Office Version:" _
                            & Application.Version
            .Labels(3).Text = "Das heutige Datum ist der:" _
                            & vbCrLf & Date
            .Labels(4).Text = "Die aktuelle Uhrzeit ist:" _
                            & vbCrLf & Time

            .Show
        End With
    End With
End Sub
```

248 Eine Sprechblase mit Kontrollkästchen

Eine Prozedur mit Kontrollkästchen gestaltet sich etwas schwieriger, als eine Prozedur, die lediglich Aufzählungspunkte oder Nummerierungen aufweist, denn bei Kontrollkästchen kann eine Mehrfachauswahl getroffen werden. Das bedeutet, dass entweder keines, eines, mehrere oder alle Kontrollkästchen aktiviert werden können. Je nachdem welche Kontrollkästchen aktiviert wurden, muss der Code entsprechend darauf reagieren. Ein Kontrollkästchen ist vom Typ Checkboxes.

Grundlagen

Allgemein

Datum
Zeit

ungen

Steuer
elemente

Befehl
leisten

Objekt

Diagramm

Ereignisse

User-
Form

Web/
Mail

Extern

Gemischtes

Spezial

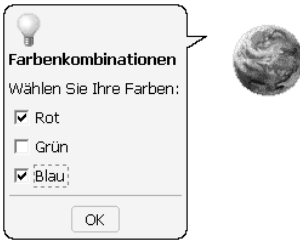


Abbildung 190: Sprechblase mit Kontrollkästchen

Achtung

Es können maximal fünf Kontrollkästchen in einer Sprechblase angezeigt werden. Falls Sie mehr einfügen, wird der Debugger gestartet.

Bei der Überprüfung, welche Kontrollkästchen aktiviert wurden, verwenden wir eine If...Then...Else-Entscheidung. In dieser Entscheidung müssen *alle* möglichen Einstellungen überprüft werden. Je nachdem, welche Auswahl getroffen wurde, wird beim Klick auf den OK-Button in der Sprechblase eine andere Meldung ausgegeben.

```
' =====
' Auf CD      Buchdaten\Beispiele\07_Objekte\
' Dateiname  07_07_Assistant.xls
' Modul      mdl_08_Checkbox
' =====
```

```
Sub MyCheckboxes()
    Dim strMsg As String

    With Assistant
        .Visible = True

        With .NewBalloon
            .Heading = "Farbenkombinationen"
            .Text = "Wählen Sie Ihre Farben:"
            .Icon = msoIconTip

            ' Drei Checkboxes werden erzeugt
            .Checkboxes(1).Text = "Rot"
            .Checkboxes(2).Text = "Grün"
            .Checkboxes(3).Text = "Blau"

            ' Prüfen, welche Checkboxes aktiviert wurden
            If .Show = msoBalloonButtonOK Then
                If .Checkboxes(1).Checked Then _
                    strMsg = .Checkboxes(1)
                If .Checkboxes(2).Checked Then _
                    strMsg = strMsg & " und " & .Checkboxes(2)
                If .Checkboxes(3).Checked Then _
                    strMsg = strMsg & " und " & .Checkboxes(3)
```



```

        If strMsg <> "" Then
            MsgBox "Sie haben " & strMsg & " gewählt"
        Else
            MsgBox "Sie haben keine Wahl getroffen"
        End If
    End If

End With
End With
End Sub
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

249 Den Text in einer Sprechblase formatieren

In einer Sprechblase können beschränkt Textformatierungen vorgenommen werden. Sie können den Text in verschiedenen Farben ausgeben. Zudem haben Sie die Möglichkeit, den Text unterstrichen darzustellen. Um solche Formatierungen vorzunehmen, wird mit Systemwerten gearbeitet. Diese werden in geschweifte Klammern {} geschrieben. Der Tabelle 56 können Sie die verschiedenen Formatierungsmöglichkeiten entnehmen.



Abbildung 191: Sprechblase mit formatiertem Text

Die einzelnen Formatierungen werden mit dem Text zusammen in Anführungs- und Schlusszeichen (" ") geschrieben. Die einzelnen Formatierungsmöglichkeiten können kombiniert werden.

```

' =====
' Auf CD      Buchdaten\Beispiele\07_Objekte\
' Dateiname   07_07_Assistant.xls
' Modul       mdl_09_Formatting
' =====

Sub Formatting()
    Assistant.Visible = True

    With Assistant.NewBalloon
        .Heading = "{cf 4}{ul 1}Hallo" & _
                  "{cf 5}{ul 0} Besucher"
        .Text = "{cf 249}{ul 0}Ich bin der" & _
               "{cf 253}{ul 1} Office-Assistent"
        .Show
    End With
End Sub
```

Systemwert	Farbe	Systemwert	Farbe
{cf 0}	Schwarz	{cf 248}	Mittelgrau
{cf 1}	Dunkelrot	{cf 249}	Rot
{cf 2}	Dunkelgrün	{cf 250}	Grün
{cf 3}	Dunkelgelb	{cf 251}	Gelb
{cf 4}	Dunkelblau	{cf 252}	Blau
{cf 5}	Dunkelmagenta	{cf 253}	Magenta
{cf 6}	Dunkelzyan	{cf 254}	Zyan
{cf 7}	Hellgrau	{cf 255}	Weiß
Systemwert	Bilddatei	Systemwert	Formatierung
{bmp Pfad/Datei}	Bitmap	{ul 0}	Nicht unterstrichen
{wmf Pfad/Datei}	Metadatei	{ul 1}	Unterstrichen

Tabelle 56: Formatierungsmöglichkeiten für Sprechblasentext

250 Grafiken in einer Sprechblase anzeigen

Wie Sie der Tabelle 56 entnehmen können, besteht die Möglichkeit, Bilder in einer Sprechblase anzuzeigen.



Abbildung 192: Grafiken in einer Sprechblase anzeigen

Was Sie in Bezug auf Grafiken in einer Sprechblase wissen müssen:

- ▶ In einer Sprechblase können nur Bilder vom Format Bitmap (*.bmp) und Metadateien (*.wmf) angezeigt werden sowie PICT-Dateien (*.pict) für den Macintosh.
- ▶ Die Grafiken können im Assistenten nicht skaliert werden. Das heißt, die Größe der Grafik kann nicht verändert werden.
- ▶ Die Grafiken werden eingebunden und müssen deshalb in einer separaten Datei vorliegen.
- ▶ Die Angabe der Bilder erfolgt in geschweiften Klammern {}. In diesen Klammern können keine Variablen verwendet werden. Sie müssen in den Klammern den genauen Pfad der Grafik angeben. Auf der Buch-CD befinden sich die entsprechenden Dateien im selben Pfad wie das Beispiel. Um das Beispiel ausführen zu können, kopieren Sie die Grafiken ins Laufwerk C: oder ändern Sie die Pfadangabe.
- ▶ Falls Sie die Sprechblase auf einem anderen PC verwenden möchten, denken Sie daran, dass auch dort das Bild vorhanden sein muss. Passen Sie gegebenenfalls den Pfad an.
- ▶ Es ist nicht möglich, Text um eine Grafik fließen zu lassen.

In unserem Beispiel werden zwei Bilder verschiedenen Typs eingefügt, eine Bitmap-Datei und eine Meta-Datei. Der Text im Titel wird in der Farbe Magenta und unterstrichen formatiert. Vor dem zweiten Bild wird zuerst ein Text eingefügt, danach folgt das Bild und anschließend wiederum ein Text. Dabei ist deutlich zu erkennen (siehe Abbildung 192), dass der Text im Textbereich bündig an der unteren Linie des Bildes ausgerichtet ist. Wenn der Text länger ist, wird er unterhalb des Bildes am Anfang der Zeile weitergeführt.

```
'=====
' Auf CD      Buchdaten\Beispiele\07_Objekte\
' Dateiname  07_07_Assistant.xls
' Modul      mdl_10_Pictures
'=====

Sub Pictures()
  With Assistant
    .Visible = True

    With .NewBalloon
      .Heading = " {cf 5}{u1 1}{bmp C:\Idylle.bmp}" & _
        " ... Das Bild mit dem Namen "Idylle"."
      .Text = "Ein Text vor dem Tiger ... " & _
        "{wmf C:\Tiger.wmf}" & _
        " ... Ein Text nach dem Tiger"

      .Show
    End With
  End With
End Sub
```

251 Eine Sprechblase mit Hyperlinks

Sie können eine Sprechblase mit Hyperlinks versehen. Der Text, der den Hyperlink darstellt, wird nicht automatisch in blauer Farbe und unterstrichen dargestellt. Dies sind Formatierungen, die wir explizit vornehmen müssen. Die Formatierungsmöglichkeiten können Sie der Tabelle 56 entnehmen.



Abbildung 193: Sprechblase mit Hyperlinks

Wir arbeiten in diesem Beispiel mit zwei Prozeduren. In der ersten Prozedur wird der Hyperlink aufbereitet bzw. formatiert. Dieser wird an je ein Label übergeben. Um den Hyperlink an die zweite Prozedur übergeben zu können, verwenden wir die Variable `intAssist`. Damit können wir prüfen, ob in der Sprechblase das erste Label oder das zweite Label angeklickt wurde. Je nachdem welche Auswahl getroffen wurde, wird in der zweiten Prozedur dem entsprechenden Hyperlink gefolgt.

Grundlagen

Allgemein

Datum
Zeit

Benutzungen

Steuerelemente

Befehlsliste

Objekt

Diagramm

Ereignisse

User-Form

Web/Mail

Extern

Gemischtes

Spezial

```

' =====
' Auf CD      Buchdaten\Beispiele\07_Objekte\
' Dateiname   07_07_Assistant.xls
' Modul       mdl_11_Hyperlink
' =====

Sub Hyperlink()
    Dim intAssist As Integer

    With Assistant
        .Visible = True

        With .NewBalloon
            .Heading = "Internet"
            .Text = "Welche Webseite möchten Sie besuchen?"
            .Icon = msoIconAlertQuery

            ' Zwei Label mit blauer Farbe und unterstichenem
            ' Hyperlinktext
            .Labels(1).Text = "Office-Help-Desk:" & vbCrLf & _
                "{cf 252}{ul 1}" & _
                "www.Jumper.ch"
            .Labels(2).Text = "Buchverlag:" & vbCrLf & _
                "{cf 252}{ul 1}" & _
                "www.Addison-Wesley.de"

            .Button = msoButtonSetCancel

            ' Die Auswahl des Labels an eine Variable übergeben (Index)
            intAssist = .Show
        End With
    End With

    ' Hyperlink an zweite Prozedur übergeben
    Select Case intAssist
        Case 1
            Call FollowHyperlink _
                ("http://www.jumper.ch")
        Case 2
            Call FollowHyperlink _
                ("http://www.addison-wesley.de")
    End Select
End Sub

```

In der nachfolgenden Prozedur wird der eigentliche Weg zur Webseite aufbereitet. Es wird ein Objekt vom Typ Internet Explorer erzeugt. Das bedeutet, dass vorausgesetzt wird, dass der Internet Explorer installiert ist. In der With-Anweisung wird die Methode `Navigate` verwendet, um dem Hyperlink zu folgen. Der Hyperlink wurde von der ersten Prozedur an die zweite übergeben. Der Wert ist in der Variablen `strHyperlink` enthalten. Je nach Auswahl in der Sprechblase ist dies der erste oder der zweite Hyperlink. Die Methode `Visible = True` bewirkt, dass das Browser-Fenster angezeigt wird. Die Methode `Offline = False` stellt sicher, dass die Webseite online aus dem Internet abgefragt wird.

```
Private Sub FollowHyperlink(strHyperlink As String)
    Dim objInternetExplorer As Object

    Set objInternetExplorer = CreateObject _
        ("InternetExplorer.Application")

    ' Dem Hyperlink folgen
    With objInternetExplorer
        .Navigate strHyperlink
        .Visible = True
        .Offline = False
    End With
End Sub
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

252 Den Assistenten animieren

Zu guter Letzt in dieser Kategorie wenden wir uns etwas Spielerischem zu. Der Assistent führt ja hin und wieder Animationen aus. Auch diese können Sie selbst per VBA steuern. Es gibt Animationen, die nur einmalig ausgeführt werden, und es gibt Animationen, die permanent ausgeführt werden. Welche das sind, können Sie den Tabellen 57 und 58 entnehmen.

Animationen können auch manuell ausgelöst werden. Klicken Sie dazu mit der rechten Maustaste auf den Assistenten und wählen Sie aus dem Kontextmenü den Eintrag ANIMATION!.



Abbildung 194: Den Assistenten animieren

In unserer Prozedur erzeugen wir fünf Labels. Jedem Label weisen wir eine andere Animation zu (Eigenschaft Animation). In der Select Case-Anweisung werden die Animationen aufbereitet. Je nachdem, welches Label in der Sprechblase angeklickt wurde, wird die entsprechende Animation ausgeführt.

```
'=====
' Auf CD      Buchdaten\Beispiele\07_Objekte\
' Dateiname   07_07_Assistant.xls
' Modul       mdl_12_Animation
'=====

Sub Animation()
    Dim lngChoice As Long

    With Assistant
        .Visible = True
```

```

With .NewBalloon
    .Heading = "Farben"
    .Text = "Was soll ich tun?"
    .Icon = msoIconAlertQuery

    ' Keine Schaltfläche anzeigen
    .Button = msoButtonSetNone

    ' Fünf Aufzählungspunkte
    .Labels(1).Text = "Erscheinen"
    .Labels(2).Text = "Verschwinden"
    .Labels(3).Text = "Erfolg zeigen"
    .Labels(4).Text = "Nachdenklich"
    .Labels(5).Text = "Aufmerksamkeit"

    lngChoice = .Show
End With

' Je nach Auswahl eine Animation ausführen
Select Case lngChoice
    Case 1
        .Animation = msoAnimationAppear
    Case 2
        .Animation = msoAnimationDisappear
    Case 3
        .Animation = msoAnimationCharacterSuccessMajor
    Case 4
        .Animation = msoAnimationThinking
    Case 5
        .Animation = msoAnimationGetAttentionMajor
End Select

End With
End Sub

```

Hinweis

Nicht jede der Konstanten zum Objekt `Animation` löst immer einen Effekt aus. Die Animationen hängen zum Teil vom eingesetzten Assistenten ab. Es würde zu weit führen, diese alle hier aufzuführen.

Konstanten zu einmaliger Animation

<code>msoAnimationAppear</code>	<code>msoAnimationIdle</code>
<code>msoAnimationBeginSpeaking</code>	<code>msoAnimationListensToComputer</code>
<code>msoAnimationCharacterSuccessMajor</code>	<code>msoAnimationLookDown</code>
<code>msoAnimationDisappear</code>	<code>msoAnimationLookDownLeft</code>
<code>msoAnimationEmptyTrash</code>	<code>msoAnimationLookDownRight</code>
<code>msoAnimationGestureDown</code>	<code>msoAnimationLookLeft</code>

Tabelle 57: Einmalige Animationen

Konstanten zu einmaliger Animation	
msoAnimationGestureLeft	msoAnimationLookRight
msoAnimationGestureRight	msoAnimationLookUp
msoAnimationGestureUp	msoAnimationLookUpLeft
msoAnimationGetArtsy	msoAnimationLookUpRight
msoAnimationGetAttentionMajor	msoAnimationPrinting
msoAnimationGetAttentionMinor	msoAnimationRestPose
msoAnimationGetWizardy	msoAnimationSaving
msoAnimationGoodbye	msoAnimationSearching
msoAnimationGreeting	msoAnimationSendingMail

Tabelle 57: Einmalige Animationen (Forts.)

Konstanten zu permanenter Animation	
msoAnimationCheckingSomething	msoAnimationWorkingAtSomething
msoAnimationThinking	msoAnimationSearching
msoAnimationWritingNothingSomething	msoAnimationGetTechy

Tabelle 58: Permanente Animationen

Grundlagen

Allgemein

Datum
Zeit

Diagramme

Steuerelemente

Befehle
ausführen

Objekte

Diagramme

Ereignisse

User-Formen

Web/
Mail

Extern

Gemischtes

Spezial

Diagramme

Diagramme sind ein beliebtes Instrument in Excel. Mittels Diagrammen lassen sich Daten als Grafik farbig und übersichtlich gestalten. Ein Diagramm besteht aus verschiedenen Elementen, wie z.B. den Säulen, Trendlinien, Legenden und vielem mehr. Um Diagramme per VBA zu erstellen oder zu verändern, steht Ihnen deshalb eine große Anzahl an Objekten, Eigenschaften und Methoden zur Verfügung.

Auf den nachfolgenden Seiten finden Sie eine umfangreiche Sammlung an Beispielen, die alle das Erstellen und Bearbeiten von Diagrammen und deren Elementen behandeln. Die ersten Schritte im Umgang mit Diagrammen per Programmierung sind in der Tat etwas gewöhnungsbedürftig. Das Ansprechen der Objekte ohne Selektierung und Aktivierung erweist sich hin und wieder als etwas schwierig.

Achtung

Wichtig und unerlässlich beim Arbeiten mit Diagrammen ist eine korrekte Deklaration der Variablen und Referenzierung der Objekte und Bezüge. Dies erhöht zum einen die Geschwindigkeit der Prozedur und dient zum anderen einer übersichtlicheren Darstellung und einer Vermeidung von Fehlern.

Um ein Diagramm darzustellen, bestehen zwei Möglichkeiten: Entweder Sie verwenden ein eigenes Excel-Diagrammblatt, wo Sie die Grafik blattfüllend darstellen können, oder Sie fügen das Diagramm innerhalb des Tabellenblattes ein. Der Vorteil bei Letzterem ist, dass Sie mit nur einem Blatt auskommen und nicht ständig zwischen den Blättern hin- und herklicken müssen. Ein weiterer Vorteil besteht darin, dass man die Daten und das Diagramm im Blickfeld hat. Welche der beiden Möglichkeiten Sie wählen, hängt wohl unter anderem vom Umfang der Daten ab, die Sie darstellen möchten. Natürlich lässt sich die Einstellung auch im Nachhinein noch verändern. So kann ein Diagramm, das sich in einem Tabellenblatt befindet, in ein Diagrammblatt umgewandelt werden und umgekehrt. Bei aktiviertem Diagramm ist in der Menüleiste der Eintrag **DIAGRAMM** zu finden. Wählen Sie darin den Eintrag **SPEICHERORT** aus und treffen Sie die gewünschte Auswahl.

Ein Diagramm wird in der Regel über den Diagramm-Assistenten erstellt. Diesen erreichen Sie, indem Sie in der Symbolleiste **STANDARD** die Schaltfläche für den Diagramm-Assistenten anklicken. Die Schaltfläche ist leicht zu erkennen, denn sie stellt ein kleines Diagramm dar. Alternativ können Sie den Assistenten auch über den Menüpunkt **EINFÜGEN | DIAGRAMM** starten. Markieren Sie vor dem Starten des Assistenten die Daten, die grafisch dargestellt werden sollen. Sie können die Quelle auch erst angeben, wenn Sie den Assistenten bereits gestartet haben. Die Daten lassen sich bequem auch bei geöffnetem Diagramm-Assistenten (2. Schritt) markieren.

Um Ihnen einen ersten Überblick über die wichtigsten Objekte zu verschaffen, haben wir nachfolgend die Objekt-Hierarchie zu den Diagrammen zusammengefasst (siehe Abbildung 195). Eine noch ausführlichere Auflistung können Sie der Online-Hilfe von Excel-VBA entnehmen, indem Sie den Begriff »Chart« eingeben.

Um die Namen der Objekte zu erfahren, führt der einfachste Weg über den Makro-Rekorder. Zeichnen Sie die gewünschten Schritte auf oder klicken Sie bei einem bestehenden Diagramm auf die Elemente, von welchen Sie die VBA-Namen erfahren möchten. Sie werden im Rezept 280

Grundlagen

Allgemein

Datum
Zeit

Formeln

Steuerelemente

Befehle
leisten

Objekte

Diagramm

Ereignisse

User-Form

Web/
Mail

Extern

Gemischtes

Spezial

erfahren, wie die Namen der Objekte in einem Nachrichtenfenster angezeigt werden können. Wechseln Sie nach Abschluss der Aufzeichnung in den VBA-Editor und sehen Sie sich den Code an. In der Regel muss ein solcher Code komplett umgeschrieben oder angepasst werden, um eine saubere Programmierung zu erhalten, denn der Makro-Rekorder erzeugt keine Variablen (Dim) und kennt auch keine Referenzierung (Set). Der Code ist dennoch nützlich, denn er erspart die oft etwas schwierige und zeitraubende Suche in der Online-Hilfe nach Objekten, Eigenschaften und Methoden.

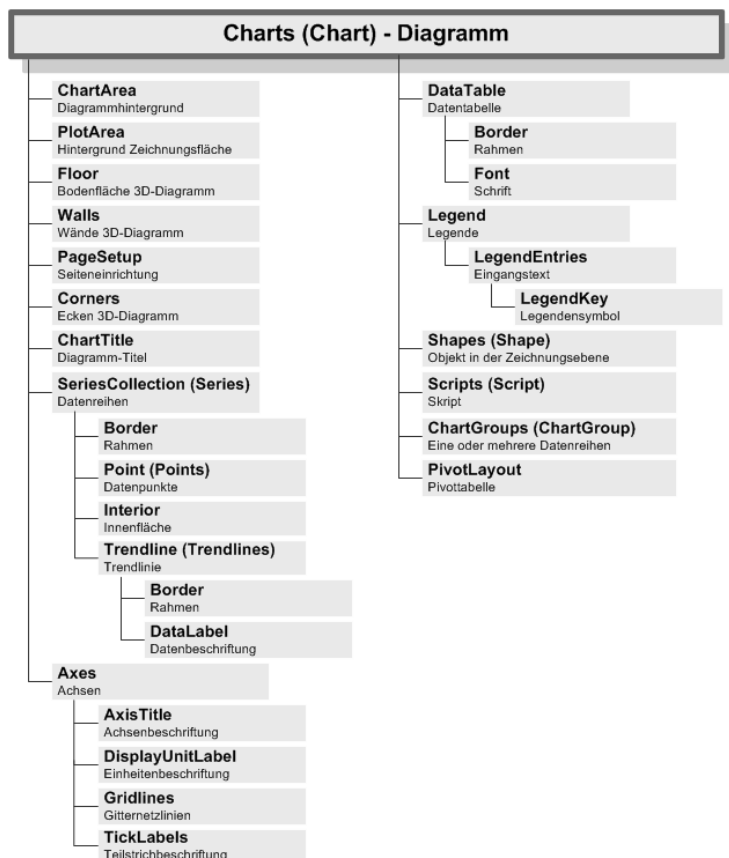


Abbildung 195: Hierarchie der Diagramm-Objekte

Hinweis

Bei vielen der nachfolgenden Beispiele haben wir zugunsten der übersichtlichen Darstellung auf eine Fehlerbehandlung verzichtet. Ein Fehler wird jedoch unweigerlich ausgelöst, wenn einer der Codes z.B. ohne bestehendes Diagramm ausgeführt wird.

Eine (nicht sehr elegante) Möglichkeit, um sämtliche Fehler zu umgehen, ist:

```
On Error Resume Next
```

Eine bessere Lösung wäre beispielsweise:

```
If ActiveChart Is Nothing Then
    Exit Sub
End If
```

Einige der VBA-Befehle gelten nur für 3-D-Diagramme. Wenn ein 2-D-Diagramm zu Grunde liegt und ein Code auf ein 3-D-Diagramm zugeschnitten ist, kann es ebenfalls zu einem Fehler führen, der im Debugger endet. In der Praxis ist es somit unerlässlich, die Codes auf mögliche Fehler zu testen und gegebenenfalls entsprechende Routinen einzubauen, die solche Fehler abfangen.

253 Diagrammtypen und -index

Als Sie das erste Mal den Diagramm-Assistenten gestartet haben, haben Sie bestimmt auch vor der Frage gestanden: »Welcher Diagrammtyp eignet sich wohl am besten, um meine Daten aussagekräftig darzustellen?« Und sicherlich haben Sie erfreut festgestellt, dass eine große Vielzahl an Diagrammtypen zur Verfügung steht. Die Entscheidung, welchen Typen Sie verwenden sollen, können wir Ihnen nicht abnehmen, denn das hängt von zu vielen verschiedenen Faktoren ab. Einerseits von der Form der Tabelle und andererseits auch davon, was das Diagramm aussagen soll.

Es stehen 14 Diagrammtypen zur Verfügung. Innerhalb dieser Typen sind Untertypen zu finden. Alles in allem finden Sie in Excel insgesamt 73 Diagramm(unter)typen. Der Tabelle 59 können Sie eine Zusammenfassung der Diagrammhaupttypen entnehmen. Es würde zu weit führen, sämtliche Diagrammtypen hier darzustellen. Wir möchten Ihnen diese jedoch nicht vorenthalten und haben Sie auf der CD in unserem Beispiel *08_01_ChartTypIndex.xls* zusammengefasst. Mit diesem Beispiel werden wir uns im Anschluss an die Tabelle noch weiter befassen.

Diagrammtyp	Abbildung
Säule	
Balken	
Linie	
Kreis	
Punkt (XY)	
Fläche	

Tabelle 59: Diagrammtypen

Grundlagen

Allgemein

Datum
Zeit

tungen

Steuer
elemente

Befehl
leisten

Objekt

Diagramm

Ereignisse

User-
Form

Web/
Mail

Extern

Gemischtes

Spezial

Diagrammtyp	Abbildung
Ring	
Netz	
Oberfläche	
Blase	
Kurs	
Zylinder	
Kegel	
Pyramide	

Tabelle 59: Diagrammtypen (Forts.)

Wie gesagt, es ist manchmal nicht einfach, den richtigen Diagrammtypen auszuwählen. Oft wäre es wünschenswert, wenn auf einfache Weise die verschiedenen Diagrammtypen nacheinander angesehen werden könnten. Der Weg über den Diagramm-Assistenten ist dabei etwas umständlich. Eine Vorschau im Diagramm-Assistenten ist zwar möglich, jedoch nicht in Originalgröße. Viel bequemer wäre es deshalb, wenn über ein Drehfeld im Tabellenblatt der Diagrammtyp gewechselt werden könnte. In VBA gibt es für jeden der 73 Diagrammtypen einen Namen bzw. eine Konstante. Die Namen der Diagramm-Konstanten oder zumindest deren Index müssen bekannt sein, wenn Sie damit arbeiten wollen. Eine Zusammenfassung der Konstanten können Sie der Bibliothek von Excel entnehmen. Diese erreichen Sie, indem Sie im VBA-Editor die Taste **[F2]** drücken. Da ein Auslesen dieser Konstanten per VBA nicht möglich ist, haben wir sie in unserer Übungsdatei zusammengefasst.

Das Beispiel (siehe Abbildung 196), das wir uns nun ansehen wollen, umfasst in der Spalte A den Index der verschiedenen Diagrammtypen. In der Spalte B sind die VBA-Konstanten zu finden

und in Spalte C (im Beispiel ausgeblendet) deren Bedeutung in deutscher Sprache. Das Beispiel umfasst zudem den Bereich E2:I6, der die Daten enthält, und ein zugehöriges Diagramm.

Die rote Zelle L3 zeigt den Index des Diagramms an. In den Zellen L6 und L7 werden die VBA-Konstanten und die deutsche Beschreibung dazu ausgegeben. Diese beiden Werte werden durch einen SVERWEIS ermittelt, welcher sich auf die Matrix der Spalten A bis C bezieht. Eine WENN-Bedingung umschließt den SVERWEIS. Es handelt sich dabei um eine Fehlerabfrage ISTFEHLER, die prüft, ob in der Zelle L3 ein gültiger Index vorhanden ist. Falls der Index ungültig ist, wird in den Zellen L6 und L7 der Text UNGÜLTIGER INDEX ausgegeben, ansonsten der Index selbst.

Formel in Zelle L6:

```
=WENN(ISTFEHLER(SVERWEIS($L$3;$A$2:$C$74;2;0));»UNGÜLTIGER INDEX«;SVERWEIS($L$3;$A$2:$C$74;2;0))
```

Formel in Zelle L7:

```
=WENN(ISTFEHLER(SVERWEIS($L$3;$A$2:$C$74;3;0));»UNGÜLTIGER INDEX«;SVERWEIS($L$3;$A$2:$C$74;3;0))
```

Über die Zellen L6 und L7 wurde eine bedingte Formatierung gelegt, die den Text UNGÜLTIGER INDEX in roter Farbe, fett und kursiv darstellt, sofern ein falscher Index im Feld L3 auftaucht.

Der letzte und wohl wichtigste Teil ist ein Drehfeld `SpinButton`. Das Drehfeld stammt aus der Symbolleiste `STEUERELEMENT-TOOLBOX`. Ein Drehfeld ist bekanntlich an ein Tabellenblatt gebunden und die Prozedur, die dahinter steckt, ist eine Ereignisprozedur. Die Prozedur wird ausgeführt, sobald einer der Pfeile des Drehfeldes angeklickt wird. Bei eingeschaltetem Entwurfsmodus (erste Schaltfläche in der Symbolleiste `STEUERELEMENT-TOOLBOX`) gelangen Sie, per Doppelklick auf das Drehfeld, in die VBA-Umgebung dieses Tabellenblattes. Sie befinden sich nun an der richtigen Stelle, um Ihren Code einzugeben. Das Codegerüst, das bereits vorhanden ist, wird in unserem Beispiel durch ein anderes ersetzt.

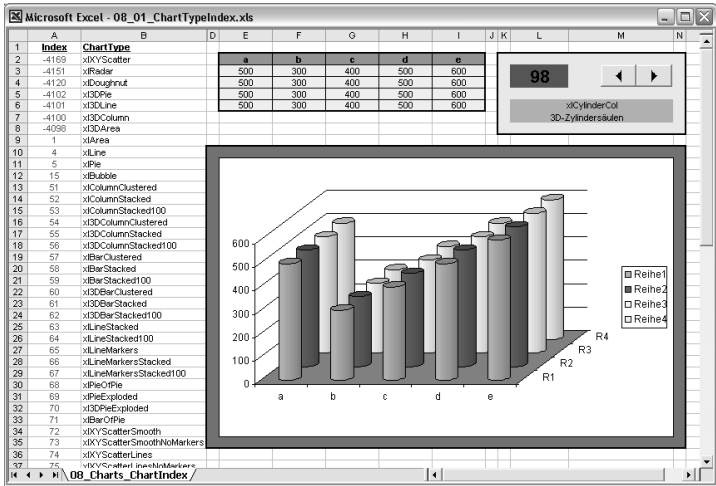


Abbildung 196: Diagrammtypen und Index ermitteln

Für das Drehfeld setzen wir zwei verschiedene Ereignisprozeduren ein. Eine für den Pfeil nach links, `SpinButton_Down`, und eine für den Pfeil nach rechts, `SpinButton_Up`. Es hängt von der Höhe und Breite des Drehfeldes ab, ob die Pfeile nach oben und unten, oder nach links und rechts zei-

Grund-lagen

Allge-mein

Datu Zeit

tungen

Steuer elemen

Befehl leisten

Objekt

Dia-gramm

Ereig-nisse

User-Form

Web/Mail

Extern

Ge-misch tes

Specia

gen. Wenn die Breite des Feldes größer ist als die Höhe, so wie in unserem Beispiel, dann erhalten Sie Pfeile, die nach links und rechts zeigen. Wenn die Höhe des Feldes größer ist als die Breite, erhalten Sie Pfeile, die nach oben und unten zeigen.

Sehen wir uns die beiden Codes an, die dem Drehfeld hinterlegt sind. Um das Diagramm anzusprechen, verwenden wir die Variable `chtIndex`, welche wir als `Chart` deklarieren. In der nächsten Codezeile weisen wir dieser Variablen das Diagramm zu. `ChartObjects(1).Chart` steht dabei für das erste Diagramm. Theoretisch wäre es ja möglich, dass sich auf dem Tabellenblatt mehrere Diagramme befinden. Die Diagramme können über deren Index oder Namen angesprochen werden. Wenn Sie ein Diagramm erstellen, trägt dieses automatisch den Namen `DIAGRAMM 1`. Weitere Diagramme erhalten aufsteigende Nummern. Anstelle eines Index können Sie eben diesen Namen ansprechen (`ChartObjects("Diagramm 1")`). Sie können den Namen eines Diagramms auch ändern. Das Vorgehen ist dasselbe wie beim Benennen einer Zelle oder eines Bereiches. Der einfachste Weg führt über das Feld links der `BEARBEITUNGSLEISTE`. Aktivieren Sie erst das Diagramm und geben Sie dann in dem Feld den gewünschten Namen ein. Es ist oft nicht ganz einfach, das Diagramm selbst zu aktivieren, ohne dabei die Diagrammfläche oder ein anderes Element zu erwischen. Sie können alternativ das Diagramm direkt per VBA umbenennen:

```
ActiveSheet.ChartObjects(1).Name = "Mein Diagramm"
```

Nach der Referenzierung verwenden wir eine Fehlerbehandlung. Dies für den Fall, dass ein ungültiger Index angewählt wird und ein entsprechendes Diagramm nicht existiert. Erstaunlicherweise entbehrt die Liste der Index-Einträge jeglicher nachvollziehbarer Logik. So sind einige Werte in einem 4000er Minusbereich zu finden und andere in einem Plusbereich zwischen 1 und 112, dies jedoch nicht lückenlos. Beim Einsatz eines Drehfeldes, das, wie in unserem Beispiel, jeweils eine Zahl addiert oder subtrahiert, gelangt man so zwangsläufig früher oder später zu einem ungültigen oder besser gesagt nicht vorhandenen Index.

Nach der Fehlerbehandlung wird dem Diagramm `chtIndex` der Typ `ChartType` zugewiesen. Der Wert wird aus der Zelle L3 gezogen, wobei beim Klick auf den linken Pfeil des Drehfeldes mit dem Wert -1 und beim Klick auf den rechten Pfeil mit dem Wert +1 reagiert werden muss. Der Zelle L3 wird dann der neue Index des Diagrammtyps zugewiesen.

```
' =====
' Auf CD      Buchdaten\Beispiele\08_Diagramme\
' Dateiname   08_01_ChartTypeIndex.xls
' Ereignis     Tabelle1(08_Charts_ChartIndex)
' =====
```

```
' Drehfeld nach links
```

```
Private Sub SpinButton1_SpinDown()
    Dim chtIndex As Chart

    Set chtIndex = ActiveSheet.ChartObjects(1).Chart

    On Error GoTo ErrorHandler

    chtIndex.ChartType = Range("L3") - 1
    Range("L3") = chtIndex.ChartType

    Set chtIndex = Nothing
```

```
Exit Sub
ErrorHandler:
MsgBox "Ungültiger Index." & vbCrLf & _
"- oder -" & vbCrLf & _
"Diagramm-Typ kann bei aktueller Tabelle " & _
"nicht angezeigt werden." & vbCrLf & vbCrLf & _
"Geben Sie einen gültigen Wert in das rote Feld ein.", _
Title:="Diagramm-Index: Fehlermeldung"
End Sub

'=====
' Auf CD      Buchdaten\Beispiele\08_Diagramme\
' Dateiname   08_01_ChartTypeIndex.xls
' Ereignis    Tabelle1(08_Charts_ChartIndex)
'=====

' Drehfeld nach rechts
Private Sub SpinButton1_SpinUp()
Dim chtIndex As Chart

Set chtIndex = ActiveSheet.ChartObjects(1).Chart

On Error GoTo ErrorHandler

chtIndex.ChartType = Range("L3") + 1
Range("L3") = chtIndex.ChartType

Set chtIndex = Nothing

Exit Sub
ErrorHandler:
MsgBox "Ungültiger Index." & vbCrLf & _
"- oder -" & vbCrLf & _
"Diagramm-Typ kann bei aktueller Tabelle " & _
"nicht angezeigt werden." & vbCrLf & vbCrLf & _
"Geben Sie einen gültigen Wert in das rote Feld ein.", _
Title:="Diagramm-Index: Fehlermeldung"
End Sub
```

In unserem Beispiel soll zudem die Möglichkeit bestehen, in das rote Feld, Zelle L3, einen Index von Hand einzugeben. Verwenden Sie dazu eine der roten Zahlen aus der Spalte A der Beispieldatei. Damit das Diagramm direkt auf die Eingabe reagiert und den Typ entsprechend dem Wert in Zelle L3 ändert, ist eine weitere Ereignisprozedur erforderlich. Wir verwenden dazu das Ereignis `Worksheet_Change`. Die Vorgehensweise entspricht im Grunde genommen den Prozeduren des Drehfeldes. Der Unterschied besteht darin, dass hier auf ein Subtrahieren oder Addieren des Wertes 1 verzichtet werden kann, denn dem `ChartType` wird direkt der neue Inhalt von Zeile L3 zugewiesen. Damit die Prozedur nur bei einer Veränderung der Zelle L3 ausgeführt wird, muss die Zieladresse im Code abgefragt werden:

```
If Target.Address <> "$L$3" Then Exit Sub
```

Grundlagen
Allgemein
Datum
Zeit
tungen
Steuerelemente
Befehle
Objekt
Diagramm
Ereignisse
User-Form
Web/Mail
Extern
Gemischtes
Spezial

Die obige Codezeile bedeutet, dass die Prozedur verlassen wird, wenn es sich bei der Zieladresse Target.Address nicht um die Zelle L3 handelt. Mehr zum Thema Ereignisprozeduren erfahren Sie in der Kategorie »Ereignisse«.

```
'=====
' Auf CD      Buchdaten\Beispiele\08_Diagramme\
' Dateiname   08_01_ChartTypeIndex.xls
' Ereignis     Tabelle1(08_Charts_ChartIndex)
'=====

Private Sub Worksheet_Change(ByVal Target As Range)
    Dim chtIndex As Chart

    Set chtIndex = ActiveSheet.ChartObjects(1).Chart

    On Error GoTo ErrorHandler

    If Target.Address <> "$L$3" Then Exit Sub
    chtIndex.ChartType = Range("L3")

    Set chtIndex = Nothing

Exit Sub
ErrorHandler:
    MsgBox "Ungültiger Index." & vbCrLf & _
        "- oder -" & vbCrLf & _
        "Diagramm-Typ kann bei aktueller Tabelle " & _
        "nicht angezeigt werden." & vbCrLf & vbCrLf & _
        "Geben Sie einen gültigen Wert in das rote Feld ein.", _
        Title:="Diagramm-Index: Fehlermeldung"
End Sub
```

254 Hintergrund Diagramm- und Zeichnungsfläche

Wenden wir uns zwei weiteren Diagrammelementen zu, nämlich dem Diagrammhintergrund ChartArea und dem Hintergrund der Zeichnungsfläche PlotArea. Der Diagrammhintergrund ist die äußere große Fläche, die das Diagramm sowie alle anderen Elemente beinhaltet. Die Zeichnungsfläche ist der innere Bereich, der direkt unter dem Diagramm, jedoch über dem Diagrammhintergrund liegt (siehe Abbildung 197).

Wenn Sie die Zeichnungs- oder Diagrammfläche ohne VBA formatieren möchten, klicken Sie mit der rechten Maustaste auf die gewünschte Fläche. Je nachdem, welche der beiden Flächen Sie mit der rechten Maustaste anklicken, zeigt das Kontextmenü einen anderen Inhalt. Achten Sie auf den ersten Menüpunkt im Kontextmenü. Er zeigt entweder den Text ZEICHNUNGSFLÄCHE FORMATIEREN oder DIAGRAMMFLÄCHE FORMATIEREN an. Wenn Sie einen dieser Menüpunkte anklicken, öffnet sich ein entsprechendes Dialogfenster. Darin können Sie die gewünschten Gestaltungsmöglichkeiten vornehmen. Bei aktivem Diagramm können Sie alternativ zum Kontextmenü die Auswahl auch über den Menüpunkt DIAGRAMM treffen. Wie sieht der Vorgang per VBA aus? Sehen wir uns das Ganze an einem Beispiel an.

In unserem Beispiel soll für jede der beiden Flächen das Hintergrundmuster Pattern, die Hintergrundfarbe Interior.ColorIndex sowie die Musterfarbe Interior.PatternColorIndex über den Index verändert werden können. Das Hintergrundmuster kann entweder durch die gewünschte Konstante oder deren Index festgelegt werden. Die gesamte Liste der Hintergrundmuster ist der

Beispielmappe zu entnehmen (Spalte A und B). Die Erläuterungen in Deutsch stehen in der Spalte C, welche ausgeblendet ist.

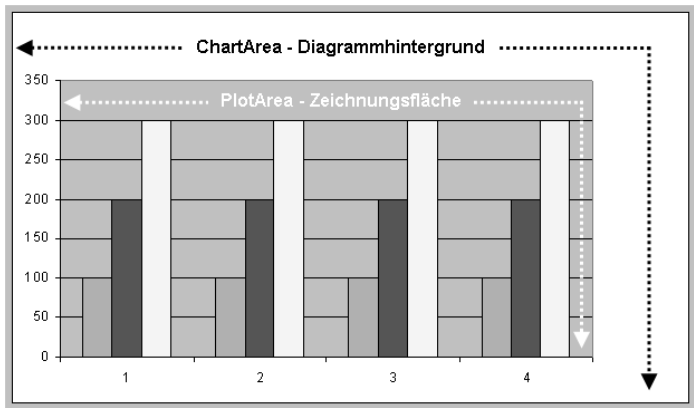


Abbildung 197: Chart- und PlotArea

Das Tabellenblatt (siehe Abbildung 198) umfasst zudem den Datenbereich E2:G6 sowie das zugehörige Diagramm. In der Spalte J kann der gewünschte Index für jedes der sechs Elemente eingetragen werden. In der Mappe ist zudem (hier auf der Abbildung nicht sichtbar) eine Farbtabelle enthalten, um das Arbeiten mit dem Farbindex zu erleichtern.

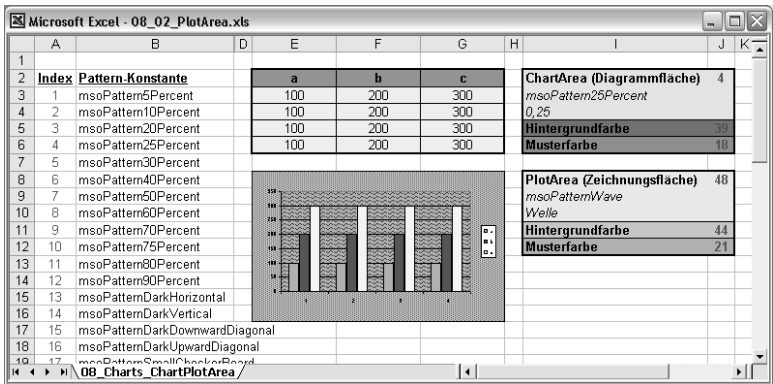


Abbildung 198: Diagramm- und Zeichnungsfläche verändern

Damit sich das Diagramm automatisch bei der Eingabe eines Index verändert, wurde dem Tabellenblatt ein Worksheet_Change-Ereignis hinterlegt. Im Code wurden zu Beginn die erforderlichen Variablen hinterlegt, für jedes anzusprechende Diagrammelement eine. Um das Arbeiten im Code zu verkürzen und um diesen übersichtlicher zu gestalten, wurden entsprechende Referenzierungen (Set) vorgenommen. Eine Fehlerbehandlung wurde eingebaut, um eine Meldung auszugeben, sofern ein falscher Index eingegeben wurde.

Damit das Ereignis nicht bei jeder Veränderung in einer beliebigen Zelle ausgeführt wird, wurden die sechs Zellen, die einen Index beinhalten, hinterlegt. Danach wird in einer With-Anweisung der Diagrammhintergrund bearbeitet. Die zweite With-Anweisung bezieht sich auf die Zeichnungsfläche. Am Ende der Prozedur werden die Verweise und somit auch der Speicherplatz wieder freigegeben. Dieser Vorgang ist etwas umstritten. Manche Programmierer verwenden das Zurücksetzen

Grundlagen

Allgemein

Datum
Zeit

ungen

Steuerelemente

Befehle
leisten

Objekt

Diagramm

Ereignisse

User-Form

Web/
Mail

Extern

Gemischtes

Spezial

konsequent, andere wiederum sind der Meinung, dass der Vorgang überflüssig ist, da die Objekte nach dem Schließen der Mappe ohnehin freigegeben werden. Es ist in jedem Fall sicherlich nicht verkehrt, die Objekte wieder freizugeben, auch wenn dies nicht immer erforderlich ist.

```
'=====
' Auf CD      Buchdaten\Beispiele\08_Diagramme\
' Dateiname  08_02_PlotArea.xls
' Ereignis   Tabelle1(08_Charts_ChartPlotArea)
'=====
```

```
Private Sub Worksheet_Change(ByVal Target As Range)
    Dim cht As Chart
    Dim chtArea As ChartArea
    Dim chtPlot As PlotArea

    Set cht = Worksheets(1).ChartObjects(1).Chart
    Set chtArea = cht.ChartArea
    Set chtPlot = cht.PlotArea

    On Error GoTo ErrorHandler

    If Target.Address <> "$J$2" And _
        Target.Address <> "$J$5" And _
        Target.Address <> "$J$6" And _
        Target.Address <> "$J$8" And _
        Target.Address <> "$J$11" And _
        Target.Address <> "$J$12" Then
        Exit Sub
    End If

    ' Diagrammhintergrund
    With chtArea
        .Fill.Patterned Pattern:=Range("J2") ' DIAGRAMMHINTERGRUND:
        .Interior.ColorIndex = Range("J5") ' Hintergrundmuster
        .Interior.PatternColorIndex = Range("J6") ' Hintergrundfarbe
        .Interior.PatternColorIndex = Range("J6") ' Musterfarbe
    End With

    ' Hintergrund Zeichnungsfläche
    With chtPlot
        .Fill.Patterned Pattern:=Range("J8") ' ZEICHNUNGSFLÄCHE:
        .Interior.ColorIndex = Range("J11") ' Hintergrundmuster
        .Interior.PatternColorIndex = Range("J12") ' Hintergrundfarbe
        .Interior.PatternColorIndex = Range("J12") ' Musterfarbe
    End With

    Set cht = Nothing
    Set chtArea = Nothing
    Set chtPlot = Nothing

    Exit Sub
ErrorHandler:
    MsgBox "Ungültiger Index", vbCritical, _
        Title:="Fehlermeldung"
End Sub
```

255 Wände und Böden von 3-D-Diagrammen

Als Nächstes wenden wir uns den Eigenschaften zu, die nur für 3-D-Diagramme Gültigkeit haben. Im Gegensatz zum vorangegangenen Beispiel, einem 2-D-Diagramm, arbeiten wir bei 3-D-Diagrammen nicht mit einer Zeichnungsfläche, sondern mit Wänden Walls und Böden Floor. Erst diese machen einen Teil des 3-D-Effektes aus. Auch hier haben wir die Möglichkeit, diese Objekte unabhängig voneinander zu gestalten.

Erst möchten wir Ihnen jedoch erläutern, wie Wände und Böden eines 3-D-Diagramms ohne VBA gestaltet werden können. Die Vorgehensweise entspricht dem vorangegangenen Rezept. Klicken Sie mit der rechten Maustaste auf die Wand oder den Boden. Je nach Auswahl zeigt das Kontextmenü den Eintrag WÄNDE FORMATIEREN oder BODEN FORMATIEREN an. Wählen Sie den entsprechenden Menüpunkt aus, um das Dialogfenster zu öffnen. Darin können Sie nun die gewünschten Formatierungen vornehmen. Im nächsten Schritt erfahren Sie, wie Wände und Böden per VBA verändert werden können.

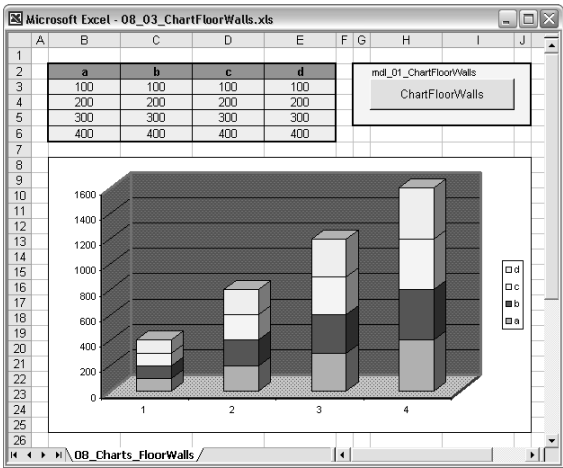


Abbildung 199: Diagramm-Wände und -Böden formatieren

Eine korrekte Dimensionierung und Referenzierung ist entscheidend. Der Code wird damit bedeutend überschaubarer.

Hinweis

Bei den Farben arbeiten wir diesmal nicht mit dem Farbindex von Excel, sondern mit RGB-Farbwerten. Mehr zum Thema RGB-Farben erfahren Sie in der Kategorie »Einstieg«.

```
'=====
' Auf CD      Buchdaten\Beispiele\08_Diagramme\
' Dateiname   08_03_FloorWalls.xls
' Modul       mdl_01_ChartFloorWalls
'=====

Sub ChartWallsCorners()
    Dim cht As Chart ' Diagramm
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

```

Dim chtFloor As Floor      ' Böden
Dim chtWalls As Walls      ' Wände

Set cht = Worksheets(1).ChartObjects(1).Chart
Set chtFloor = cht.Floor
Set chtWalls = cht.Walls

' Bodenfläche Hintergrund
With chtFloor
    With .Interior          ' HINTERGRUND BODEN:
        .Color = RGB(255, 255, 0)    ' Farbe
        .Pattern = xlPatternGray16   ' Muster
        .PatternColor = RGB(255, 0, 0) ' Musterfarbe
    End With

    ' Bodenfläche Linien
    With .Border            ' RAHMEN BODEN:
        .Color = RGB(0, 0, 255)      ' Farbe
        .LineStyle = xlContinuous    ' Linienart
        .Weight = xlThick            ' Liniendicke
    End With
End With

' Wandfläche Hintergrund
With chtWalls
    With .Interior          ' HINTERGRUND WAND:
        .Color = RGB(0, 255, 0)      ' Farbe
        .Pattern = xlPatternSemiGray75 ' Muster
        .PatternColor = RGB(255, 0, 0) ' Musterfarbe
    End With

    ' Wandfläche Hintergrund
    With .Border            ' RAHMEN WAND:
        .Color = RGB(0, 255, 0)      ' Farbe
        .LineStyle = xlContinuous    ' Linienart
        .Weight = xlThick            ' Liniendicke
    End With
End With

Set cht = Nothing
Set chtFloor = Nothing
Set chtWalls = Nothing
End Sub

```

Für die Eigenschaften wie Muster `Pattern`, Linienart `LineStyle` und Liniendicke `Weight` stehen Ihnen verschiedene Konstanten zur Verfügung. Diese können Sie den nachfolgenden Tabellen entnehmen:

Interior.Pattern	Index	Beschrei- bung	Interior.Pattern	Index	Beschrei- bung
xlPatternAutomatic	-4105	Automa- tisch	xlPatternHorizontal	-4128	Horizontal
xlPatternChecker	9	Schach- felder	xlPatternLightDown	13	Hell unten
xlPatternCrissCross	16	Zickzack	xlPatternLightHorizontal	11	Hell horizont.
xlPatternDown	-4121	Nach unten	xlPatternLightUp	14	Hell oben
xlPatternGray16	17	Grau 16%	xlPatternLightVertical	12	Hell vertikal
xlPatternGray25	-4124	Grau 25%	xlPatternNone	-4142	Keine
xlPatternGray50	-4125	Grau 50%	xlPatternSemiGray75	10	Halb Gr. 75%
xlPatternGray75	-4146	Grau 75%	xlPatternSolid	1	Ausgefüllt
xlPatternGray8	18	Grau 8%	xlPatternUp	-4162	Nach oben
xlPatternGrid	15	Gitternetz	xlPatternVertical	-4166	Vertikal

Tabelle 60: Konstante und Index zu Hintergrundmuster

Border.LineStyle	Index	Beschrei- bung	Border.LineStyle	Index	Beschrei- bung
xlContinuous	1	Durch- gezogen	xlDot	-4118	Gepunktet
xlDash	-4115	Gestrichelt	xlDouble	-4119	Doppelt
xlDashDot	4	Strich- Punkt	xlLineStyleNone	-4142	Keine
xlDashDotDot	5	Strich- Punkt- Punkt	xlSlantDashDot	13	Schräg gepunktet

Tabelle 61: Konstante und Index zu Linienart

Border.Weight	Index	Beschrei- bung	Border.Weight	Index	Beschrei- bung
xlHairline	1	Sehr dünn	xlThick	4	Dick
xlMedium	-4138	Mittel	xlThin	2	Dünn

Tabelle 62: Konstante und Index zu Liniendicke

256 Diagrammtitel hinzufügen

Recht einfach ist es, einen Diagrammtitel hinzuzufügen, oder einen bestehenden zu verändern. Dabei spielt es keine Rolle, ob bereits ein Titel besteht, dessen Text verändert werden soll, oder ob

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

erst ein Titel hinzugefügt werden muss, für beide Varianten können Sie ein und dieselbe Prozedur verwenden.

Wenn Sie einen Diagrammtitel ohne VBA zuweisen möchten, klicken Sie mit der rechten Maustaste auf die Diagrammfläche und wählen aus dem Kontextmenü den Eintrag **DIAGRAMM-OPTIONEN**, oder wählen Sie alternativ aus der Menüleiste den Menüpunkt **DIAGRAMM | DIAGRAMMOPTIONEN** aus. Voraussetzung, dass der Menüpunkt **DIAGRAMM** angezeigt wird, ist, dass das Diagramm zuvor aktiviert wurde. Wechseln Sie im Dialogfenster in die Registerkarte **TITEL** und geben Sie in der ersten Eingabezeile den Titel ein. Wenn Sie sich ein paar Sekunden gedulden, können Sie sehen, wie im Vorschaufenster des Dialogfensters der Titel erscheint. Den Achsenbeschriftungen werden wir uns später widmen.

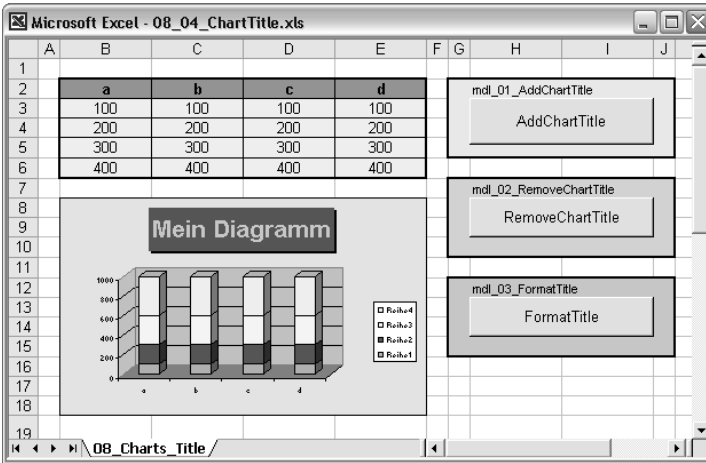


Abbildung 200: Diagrammtitel

Im nächsten Schritt erfahren Sie, wie ein Diagrammtitel per VBA in Ihr Diagramm eingefügt werden kann. In unserem Beispiel wird in einer With-Anweisung zuerst sichergestellt, dass der Titel sichtbar ist (`HasTitle = True`). Danach wird der Text hinzugefügt bzw. überschrieben (`ChartTitle.Characters.Text = "xxx"`).

```
' =====
' Auf CD      Buchdaten\Beispiele\08_Diagramme\
' Dateiname   08_04_ChartTitle.xls
' Modul       mdl_01_AddChartTitle
' =====
```

```
Sub AddChartTitle()
    With Worksheets(1).ChartObjects(1).Chart
        .HasTitle = True
        .ChartTitle.Characters.Text = "Mein Diagramm"
    End With
End Sub
```

Um einen Diagrammtitel zu entfernen, verwenden Sie die Eigenschaft `HasTitle` gefolgt vom Wert `False`.

<pre>'===== ' Auf CD Buchdaten\Beispiele\08_Diagramme\ ' Dateiname 08_04_ChartTitle.xls ' Modul mdl_02_RemoveChartTitle '===== Sub RemoveChartTitle() Worksheets(1).ChartObjects(1).Chart.HasTitle = False End Sub</pre>		Grund- lagen
		Allge- mein
		Datu Zeit

Die Titelschrift und der Hintergrund des Textfeldes kann nach Belieben gestaltet werden. Die Beschreibung der einzelnen Codezeilen finden Sie diesmal, zugunsten der Übersichtlichkeit, direkt im Code.

<pre>'===== ' Auf CD Buchdaten\Beispiele\08_Diagramme\ ' Dateiname 08_04_ChartTitle.xls ' Modul mdl_03_FormatTitle '===== Sub FormatTitle() Dim cht As Chart Dim chtTitle As ChartTitle Set cht = Worksheets(1).ChartObjects(1).Chart Set chtTitle = cht.ChartTitle ' Überprüfen, ob ein Titel vorhanden ist ' Wenn nicht wird er eingefügt If cht.HasTitle Then With cht .HasTitle = True .ChartTitle.Characters.Text = "Mein Diagramm" End With End If With chtTitle ' TITEL: .Shadow = True ' Schatten .AutoScaleFont = False ' Automatische Schriftskalierung With .Interior ' HINTERGRUND: .ColorIndex = 3 ' Rote Hintergrundfarbe .PatternColorIndex = 3 ' Musterfarbe .Pattern = xlSolid ' Muster End With With .Font ' SCHRIFT: .Name = "Arial" ' Schriftart .FontStyle = "Fett" ' Schriftstil .Size = 16 ' Schriftgröße .Strikethrough = False ' Durchgestrichen .Superscript = False ' Hochgestellt .Subscript = False ' Tiefgestellt</pre>		tungen
		Steuer elemen
		Befehl leisten
		Objekt
		Dia- gramm
		Ereig- nisse
		User- Form
		Web/ Mail
		Extern
		Ge- misch tes
		Specia

```
        .Underline = xlUnderlineStyleNone    ' Unterstrichen
        .ColorIndex = 6                      ' Farbe
    End With
End With

Set cht = Nothing
Set chtTitle = Nothing
End Sub
```

Für die Unterstreichung stehen Ihnen drei verschiedene Konstanten zur Verfügung:

Underline-Konstante	Index	Beschreibung
xlUnderlineStyleDouble	-4119	Doppelt unterstrichen
xlUnderlineStyleNone	-4142	Nicht unterstrichen
xlUnderlineStyleSingle	2	Einfach unterstrichen

Tabelle 63: Konstanten und Index für die Unterstreichung

257 Datenreihen formatieren

Datenreihen sind der Kern eines jeden Diagramms, denn sie stellen die Daten dar, welche in einer Excel-Tabelle zu Grunde liegen. Datenreihen können verschieden dargestellt werden, beispielsweise in Form von Säulen, Balken, Ringen, Blasen, Linien usw. Angesprochen werden alle über das Auflistungsobjekt: `SeriesCollection`.

Sie können Datenreihen auch ohne Zuhilfenahme von VBA formatieren. Klicken Sie dazu mit der rechten Maustaste auf die Datenreihe, die Sie formatieren möchten. Wählen Sie aus dem Kontextmenü den ersten Eintrag DATENREIHEN FORMATIEREN aus und nehmen Sie, in dem sich öffnenden Dialogfenster, die gewünschten Formatierungen vor. Sie können aus einer Datenreihe auch nur einen einzelnen Datenpunkt formatieren. Aktivieren Sie dazu zuerst die gewünschte Reihe. Danach klicken Sie innerhalb der Reihe auf den gewünschten Datenpunkt. Dass der Punkt aktiviert ist, können Sie daran erkennen, dass er nun durch entsprechende Eckmarken gekennzeichnet ist. Sobald der Datenpunkt aktiv ist, klicken Sie mit der rechten Maustaste auf diesen Punkt und wählen aus dem Kontextmenü den Eintrag DATENPUNKT FORMATIEREN. Wie gewohnt öffnet sich ein entsprechendes Dialogfenster. Sie können hier Ihre Einstellungen vornehmen.

In unserem nächsten Beispiel wollen wir uns den Umgang mit dem Objekt `SeriesCollection` anschauen. Mit einer `For...Next`-Schleife erreichen wir, dass alle vorhandenen Datenreihen gezählt werden (`Count`). Diesen Zähler verwenden wir, um den einzelnen Datenreihen verschiedene Farben zuzuweisen. Der Zähler `i` wird verwendet, um die Säulen anzusprechen und wird zudem für den Farbindex benutzt. Da der Farbindex 1 und 2 jedoch für Schwarz und Weiß stehen, unsere Säulen jedoch bunt werden sollen, addieren wir jeweils den Wert 2 zum Farbindex hinzu. Neben dem Farbindex für den Hintergrund sollen auch die Eigenschaften der Rahmen, welche die Datenreihen umgeben, verändert werden.

```
' =====
' Auf CD      Buchdaten\Beispiele\08_Diagramme\
' Dateiname   08_05_SeriesCollection.xls
' Modul       mdl_01_SeriesCollection
' =====
```



```
Sub ChartSeriesCollection()  
    Dim chtSC As SeriesCollection  
    Dim i As Integer  
  
    Set chtSC = Worksheets(1).ChartObjects(1).Chart.SeriesCollection  
  
    For i = 1 To chtSC.Count  
        With chtSC(i)  
            .Interior.ColorIndex = i + 2 ' DATENREIHE:  
                                         ' Hintergrundfarbe  
            With .Border  
                .ColorIndex = i + 3 ' RAHMEN:  
                .LineStyle = xlContinuous ' Farbe  
                .Weight = xlThin ' Linienart  
            End With  
            .Weight = 4 ' Liniendicke  
        End With  
    End With  
  
    Next i  
  
    Set chtSC = Nothing  
End Sub
```

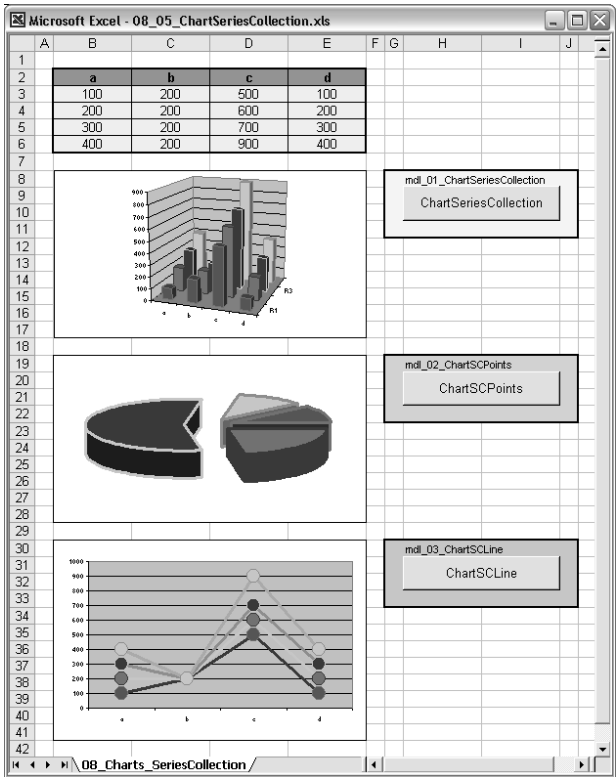


Abbildung 201: Datenreihen formatieren

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Errei-
gnisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

258 Datenpunkte formatieren

Bei Kreisdiagrammen reicht es nicht aus, die Datenreihen anzusprechen. Die einzelnen »Kuchenstücke« sind ein Objekt innerhalb der Datenreihe. Es müssen daher explizit die Datenpunkte *Points* angesprochen werden. Der nachfolgende Code berücksichtigt dies. Wenn Sie diesen Code z.B. auf ein Säulendiagramm anwenden, werden nur die einzelnen Säulen der *ersten* Datenreihe entsprechend formatiert. Das Beispiel basiert auf der Abbildung 201.

```
'=====
' Auf CD      Buchdaten\Beispiele\08_Diagramme\
' Dateiname   08_05_SeriesCollection.xls
' Modul       mdl_02_SeriesPoints
'=====

Sub ChartSCPoints()
    Dim chtSC As SeriesCollection
    Dim i As Integer

    Set chtSC = Worksheets(1).ChartObjects(2).Chart.SeriesCollection

    For i = 1 To chtSC(1).Points.Count
        With chtSC(1).Points(i)      ' DATENPUNKTE:
            .Interior.ColorIndex = i + 2 ' Hintergrundfarbe
            With .Border              ' RAHMEN:
                .ColorIndex = i + 3    ' Farbe
                .LineStyle = xlContinuous ' Linienart
                .Weight = xlThick      ' Liniendicke
            End With
        End With
    Next i

    Set chtSC = Nothing
End Sub
```

259 Markierer und Linien formatieren

So genannte Markierer sind beispielsweise in Liniendiagrammen zu finden. Sie stellen die Datenpunkte dar und können eigens formatiert werden. Bei Punkt- und Liniendiagrammen ist die Vorgehensweise zwar gleich wie im Rezept 257, es kommen jedoch noch weitere Eigenschaften hinzu. Eine Hintergrundfarbe entfällt, da hier keine Fläche vorhanden ist. Stattdessen können die Markierer und die Linien selbst in verschiedenen Farben dargestellt werden. Das Beispiel basiert auf der Abbildung 201.

```
'=====
' Auf CD      Buchdaten\Beispiele\08_Diagramme\
' Dateiname   08_05_SeriesCollection.xls
' Modul       mdl_03_SeriesLines
'=====

Sub ChartSCLine()
    Dim chtSC As SeriesCollection
```

```
Dim i As Integer

Set chtSC = Worksheets(1).ChartObjects(3).Chart.SeriesCollection

For i = 1 To chtSC.Count
    With chtSC(i)
        .MarkerBackgroundColorIndex = i + 2 ' DATENREIHE Markierer:
        .MarkerForegroundColorIndex = i + 3 ' Vordergrund
        .MarkerStyle = xlMarkerStyleCircle ' Hintergrund
        .Smooth = False ' Stil
        .MarkerSize = 10 ' Kurvenglättung
        .Shadow = False ' Größe
        With .Border ' Schatten
            .ColorIndex = i + 4 ' RAHMEN:
            .LineStyle = xlContinuous ' Farbe
            .Weight = xlThick ' Linienart
        End With ' Liniendicke
    End With
End With
Next i

Set chtSC = Nothing
End Sub
```

Im obigen Code ist noch eine neue Konstante `MarkerStyle` hinzugekommen. Auch hier möchten wir Ihnen die VBA-Werte nicht vorenthalten:

MarkerStyle Konstante	Index	Beschreibung
xlMarkerStyleAutomatic	-4105	Automatisch
xlMarkerStyleCircle	8	Kreis
xlMarkerStyleDash	-4115	Langer Strich
xlMarkerStyleDiamond	2	Viereck hochgestellt
xlMarkerStyleDot	-4118	Kurzer Strich
xlMarkerStyleNone	-4142	Kein Marker
xlMarkerStylePlus	9	Pluszeichen
xlMarkerStyleSquare	1	Viereck
xlMarkerStyleStar	5	Stern
xlMarkerStyleTriangle	3	Dreieck
xlMarkerStyleX	-4168	Ein X

Tabelle 64: MarkerStyle-Konstanten

260 Trendlinien hinzufügen

Trendlinien zeigen Ihnen, ob das Ergebnis, das das Diagramm darstellt, zu einem Gewinn oder zu einem Verlust tendiert. Je nach Diagramm ist das ohne Trendlinie nur schwer zu erkennen. Trendlinien sind Linien, die zusätzlich zu Ihrem Diagramm hinzugefügt werden können. Am besten lässt sich das an einem Beispiel und einem Bild erläutern. In Abbildung 202 ist eine Trendlinie zu

Grundlagen

Allgemein

Datum
Zeit

Diagramme

Steuerelemente

Befehle
leisten

Objekte

Diagramme

Ereignisse

User-Form

Web/
Mail

Extern

Gemischtes

Spezial

sehen. Es handelt sich dabei um die schwarze Linie, die über den Säulen liegt. Ihr ist zu entnehmen, dass die Daten des Diagramms eine leichte Steigung der Werte anzeigen. Ohne Trendlinie wäre das nur sehr schwer zu erkennen.

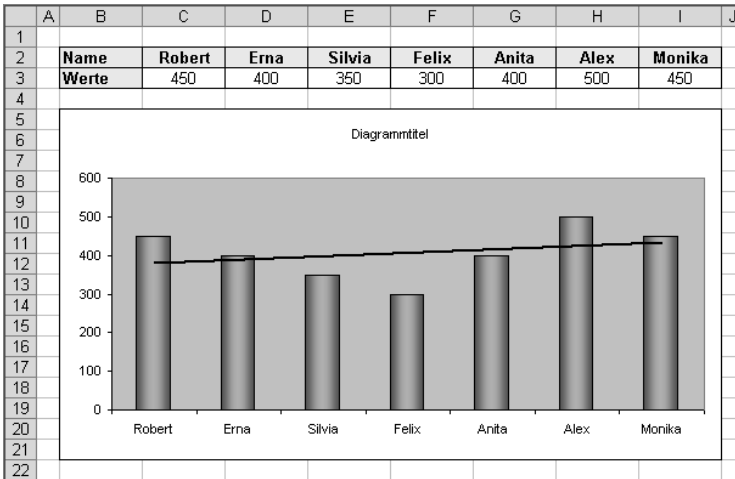


Abbildung 202: Säulendiagramm mit linearer Trendlinie

Das Bild stellt eine lineare Trendlinie dar. Es gibt verschiedene Trendlinientypen. Die einzelnen Typen, die in Excel zu Verfügung stehen, können Sie dem Dialogfenster entnehmen, welches sich öffnet, wenn Sie manuell eine Trendlinie zu Ihrem Diagramm hinzufügen. Klicken Sie dazu mit der rechten Maustaste auf die Datenreihe und wählen Sie aus dem Kontextmenü den Eintrag TRENDLINIE HINZUFÜGEN. Dem Dialogfenster, das sich nun öffnet, können Sie auf der Registerkarte TYP die sechs verschiedenen Typen entnehmen (siehe Abbildung 203). Auf die verschiedenen Diagrammtypen werden wir im Einzelnen noch zu sprechen kommen.

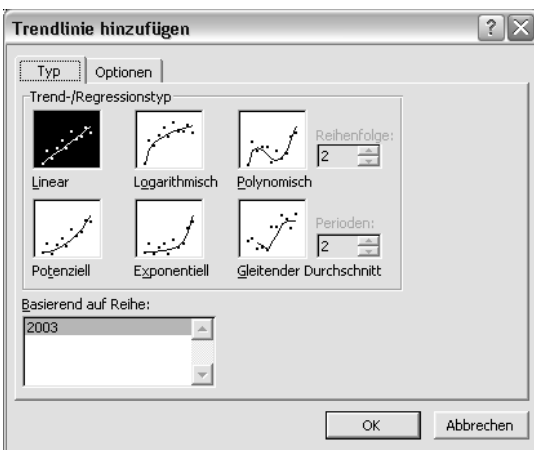


Abbildung 203: Verschiedene Trendlinientypen

Wenn Ihr Diagramm mehrere Datenreihen aufweist, kann für jede eine Trendlinie eingefügt werden.

Unser nächstes Beispiel weist drei Datenreihen auf, je eine für die Jahre 2001 bis 2003. Jeder Datenreihe wurde eine bestimmte Farbe zugewiesen. Die Datenreihe für das Jahr 2001 ist rot, die Reihe für 2002 ist türkis (cyan) und die Reihe für 2003 ist gelb. Entsprechend dieser Farben sollen auch die Trendlinien farbig dargestellt werden. Damit ist klar zu erkennen, welche Trendlinie zu welcher Datenreihe gehört. Das Objekt Trendlinie heißt in VBA *Trendline*.

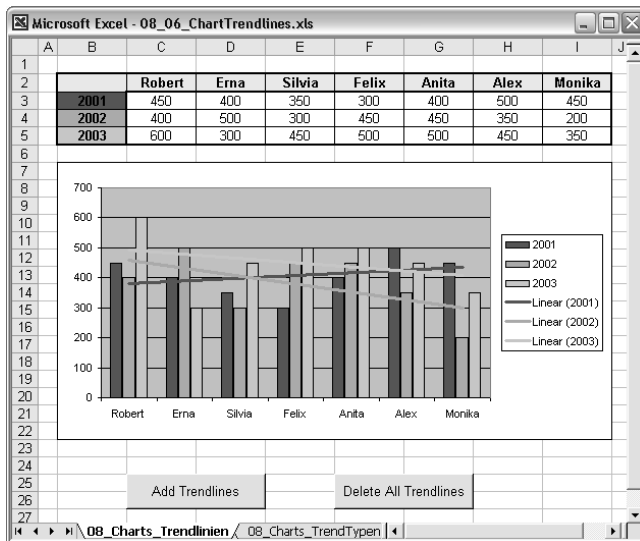


Abbildung 204: Diagramm mit drei Datenreihen und Trendlinien

Wie gesagt werden in unserem Code drei Trendlinien hinzugefügt, für jedes Jahr eine. Da wir auch noch jeder Trendlinie eine eigene Farbe zuweisen möchten, ist es am einfachsten, jede Datenreihe separat anzusprechen. Diagramm und Datenreihe werden zu Beginn der Prozedur sauber dimensioniert und referenziert. Das erleichtert späteres Ansprechen.

Jede Datenreihe wird angesprochen, indem gleich eine Trendlinie hinzugefügt wird (*Trendline.Add*). In je einer *With*-Anweisung wird die gewünschte Farbe zugewiesen. Damit die Trendlinien deutlich zu erkennen sind, stellen wir sie etwas dicker dar (*Weight = xlThick*).

```
' =====
' Auf CD      Buchdaten\Beispiele\08_Diagramme\
' Dateiname   08_06_Trendlines.xls
' Tabelle     08_Charts_Trendlinien
' Modul       mdl_01_Add_Trendlines
' =====
```

```
Sub AddTrendlines()
    Dim chtSC As SeriesCollection

    Set chtSC = ActiveSheet.ChartObjects(1).Chart.SeriesCollection

    ' Rote Trendlinie für 2001
    chtSC(1).Trendlines.Add
    With chtSC(1).Trendlines(1).Border
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

```
        .ColorIndex = 3
        .Weight = xlThick
    End With

    ' Cyan Trendlinie für 2002
    chtSC(2).Trendlines.Add
    With chtSC(2).Trendlines(1).Border
        .ColorIndex = 8
        .Weight = xlThick
    End With

    ' Gelbe Trendlinie für 2003
    chtSC(3).Trendlines.Add
    With chtSC(3).Trendlines(1).Border
        .ColorIndex = 6
        .Weight = xlThick
    End With

    Set chtSC = Nothing
End Sub
```

Zu erwähnen ist, dass nicht alle Diagrammtypen Trendlinien unterstützen. Der Tabelle 65 können Sie entnehmen, welche Diagramme Trendlinien unterstützen. Wenn Sie zum Beispiel den Trend in einem 3-D-Diagramm ermitteln möchten, ist es unerlässlich, vorher den Diagrammtyp umzustellen.

Unterstützte Typen	Nicht unterstützte Typen
Punktdiagramm (XP)	Alle 3-D-Diagramme
Flächendiagramm	Gestapelte Diagramme
Säulendiagramm	Netzdigramm
Balkendiagramm	Kreisdiagramm
Liniendiagramm	Ringdiagramm
Kursdiagramm	
Blasendiagramm	

Tabelle 65: Unterstützung von Trendlinien

261 Trendlinien entfernen

Wenn Sie eine Trendlinie manuell aus dem Diagramm entfernen möchten, klicken Sie mit der rechten Maustaste auf die Trendlinie und wählen aus dem Kontextmenü den Eintrag LÖSCHEN.

Wenn mehrere Trendlinien aus einem oder mehreren Diagrammen entfernt werden müssen, wäre es sehr umständlich, jede einzelne rechts anzuklicken und unter Zuhilfenahme des Kontextmenüs zu entfernen. Eine VBA-Prozedur kann einem das Leben dabei schon erheblich erleichtern.

Um sämtliche Trendlinien in einem Diagramm zu entfernen, muss mit zwei Schleifen gearbeitet werden. In der äußeren Schleife werden die Datenreihen angesprochen und in der inneren die Trendlinien selbst, die sich »auf« den Datenreihen befinden.

Um näher darauf einzugehen: `SeriesCollection` ist der Oberbegriff für sämtliche Datenreihen. Das Objekt, das diesem untergeordnet ist, nennt sich `Series`. In der äußeren Schleife wird nun jedes `Each` Serienelement der Datenreihen angesprochen. Alternativ wäre auch eine Schleife mit einem Zähler denkbar. In der inneren Schleife wird ebenfalls mit zwei Objekten gearbeitet, wobei `Trendlines` für alle Trendlinien und `Trendline` (also ohne »s« am Ende) für eine einzelne Trendlinie steht. Das Prinzip ist das gleiche wie bei der äußeren Schleife. Es wird jede Trendlinie innerhalb von allen Trendlinien angesprochen. Die einzelnen Trendlinien werden schließlich gelöscht (`Delete`).

```
'=====
' Auf CD      Buchdaten\Beispiele\08_Diagramme\
' Dateiname  08_06_Trendlines.xls
' Tabelle     08_Charts_Trendlinien
' Modul      mdl_02_Delete_Trendlines
'=====

Sub DeleteAllTrendlines()
    Dim chtS As Series
    Dim chtSC As SeriesCollection
    Dim chtTrend As Trendline

    Set chtSC = ActiveSheet.ChartObjects(1).Chart.SeriesCollection

    For Each chtS In chtSC
        For Each chtTrend In chtS.Trendlines
            chtTrend.Delete
        Next chtTrend
    Next chtS

    Set chtSC = Nothing
End Sub
```

262 Trendlinien-Typen

Unser nächstes Beispiel basiert auf einer einfachen kleinen Tabelle. Aus dieser Tabelle wurde ein Liniendiagramm erstellt. Dem Liniendiagramm wurde eine lineare Trendlinie hinzugefügt. Wie bereits erwähnt, gibt es sechs verschiedene Trendlinien-Typen. In unserem Tabellenblatt sind entsprechend diesen Typen sechs Optionsfelder enthalten. Auf diese Weise können Sie per Klick die Trendlinie auf einen anderen Typen umstellen. Die Optionsfelder stammen aus der Symbolleiste `STEUERELEMENT-TOOLBOX`.

Der Tabelle 66 können Sie entnehmen, welche Trendlinien-Typen in Excel zur Verfügung gestellt werden:

Trendlinien-Typ	VBA-Konstante
Linear	xlLinear
Logarithmisch	xlLogarithmic
Polynomisch	xlPolynomial

Tabelle 66: Die sechs verfügbaren Trendlinien-Typen

Grundlagen

Allgemein

Datum
Zeit

Funktionen

Steuerelemente

Befehle
leisten

Objekte

Diagramme

Ereignisse

User-Formen

Web/
Mail

Extern

Gemischtes

Spezial

Trendlinien-Typ	VBA-Konstante
Potenziell	xlPower
Exponentiell	xlExponential
Gleitender Durchschnitt	xlMovingAvg

Tabelle 66: Die sechs verfügbaren Trendlinien-Typen (Forts.)

Den einzelnen Optionsfeldern werden verschiedene Ereignisprozeduren hinterlegt. Die Ereignisprozeduren sind hinter dem zweiten Tabellenblatt zu finden.

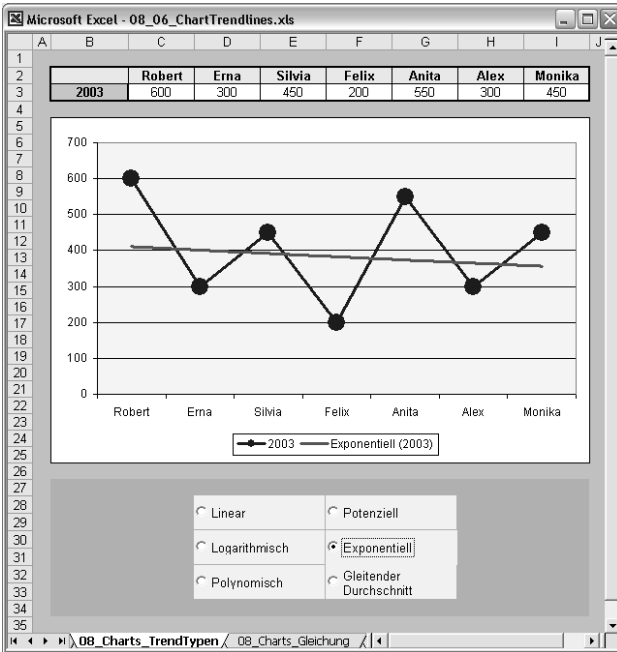


Abbildung 205: Trendlinien-Typ per Options-Schaltfläche umstellen

Jede der Ereignisprozeduren ist einem Optionsfeld zugewiesen, so dass beim Anklicken eines anderen Feldes der Typ der Trendlinie wechselt (`Trendlines(x).Type = ...`). Die erste der nachfolgenden Prozeduren ist die Hauptprozedur. Ihr wird, je nachdem welches Optionsfeld ausgewählt wurde, die entsprechende Konstante übergeben.

```
'=====
' Auf CD      Buchdaten\Beispiele\08_Diagramme\
' Dateiname  08_06_Trendlines.xls
' Tabelle    08_Charts_TrendTypen
' Ereignis   Tabelle2(08_Charts_TrendTypen)
'=====

' Hauptprozedur
Sub ChangeTrendline(varType As Variant)
    Dim chtSC As SeriesCollection
```



```
Set chtSC = ActiveSheet.ChartObjects(1).Chart.SeriesCollection
chtSC(1).Trendlines(1).Type = varType
End Sub

' Linear
Private Sub OptionButton1_Click()
    ChangeTrendline xlLinear
End Sub

' Logarithmisch
Private Sub OptionButton2_Click()
    ChangeTrendline xlLogarithmic
End Sub

' Polynomisch
Private Sub OptionButton3_Click()
    ChangeTrendline xlPolynomial
End Sub

' Potenziell
Private Sub OptionButton4_Click()
    ChangeTrendline xlPower
End Sub

' Exponentiell
Private Sub OptionButton5_Click()
    ChangeTrendline xlExponential
End Sub

' Gleitender Durchschnitt
Private Sub OptionButton6_Click()
    ChangeTrendline xlMovingAvg
End Sub
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

263 Gleichungen und Bestimmtheitsmaß ein- und ausblenden

In unserem nächsten Beispiel wollen wir über zwei Checkboxes die Möglichkeit bieten, sowohl die Gleichungen als auch das Bestimmtheitsmaß nach Wahl ein- oder auszublenden. Diese beiden Checkboxes stammen, genau wie im vorangegangenen Beispiel, ebenfalls aus der Symbolleiste STEUERELEMENT-TOOLBOX. Demzufolge benötigen wir zwei Ereignisprozeduren.

Die Eigenschaft `DisplayEquation` steht für Gleichung und die Eigenschaft `DisplayRSquared` steht für das Bestimmtheitsmaß. Wenn diese Angaben eingeblendet werden, wird zudem auch die Schrift weiß eingefärbt. Für unser Beispiel ist das insofern von Vorteil, als dass sie sich dann besser vom dunklen Hintergrund abhebt.

```
'=====
' Auf CD      Buchdaten\Beispiele\08_Diagramme\
' Dateiname   08_06_Trendlines.xls
' Tabelle     08_Charts_Gleichung
' Ereignis    Tabelle3(08_Charts_Gleichung)
```

```

' =====

' Gleichung
Private Sub CheckBox1_Click()
    Dim chtSC As SeriesCollection

    Set chtSC = ActiveSheet.ChartObjects(1).Chart.SeriesCollection

    ' Gleichung einblenden
    If CheckBox1 = True Then
        With chtSC(1).Trendlines(1)
            .DisplayEquation = True
            .DataLabel.Font.ColorIndex = 2
        End With
    Else
        chtSC(1).Trendlines(1).DisplayEquation = False
    End If

    Set chtSC = Nothing
End Sub

' Bestimmtheitsmaß
Private Sub CheckBox2_Click()
    Dim chtSC As SeriesCollection

    Set chtSC = ActiveSheet.ChartObjects(1).Chart.SeriesCollection

    ' Bestimmtheitsmaß einblenden
    If CheckBox2 = True Then
        With chtSC(1).Trendlines(1)
            .DisplayRSquared = True
            .DataLabel.Font.ColorIndex = 2
        End With
    Else
        chtSC(1).Trendlines(1).DisplayRSquared = False
    End If

    Set chtSC = Nothing
End Sub

```

264 Trendperiode vor- oder rückwärts

Mittels Diagrammdaten kann vorausgesagt werden, wie bei gleichem Trend das Ergebnis beispielsweise in einem Monat aussehen könnte. Das Gleiche gilt auch rückwärts. Das heißt, Sie können aufgrund der bestehenden Daten ermitteln, wie das Ergebnis vermutlich noch einige Monate zuvor ausgesehen hätte.

In unserem Beispiel gehen wir davon aus, dass ein Diagramm mit einer bestehenden linearen Trendlinie vorhanden ist. Um den Trend manuell einzustellen, klicken Sie mit der rechten Maustaste auf die Trendlinie und wählen aus dem Kontextmenü den Eintrag **TRENDLINIE FORMATIEREN**. In dem Dialogfenster, das sich nun öffnet, aktivieren Sie die Registerkarte **OPTIONEN**. In der Mitte des Dialogfensters lässt sich nun der Vorwärts- oder Rückwärtstrend in Einheiten bestimmen.

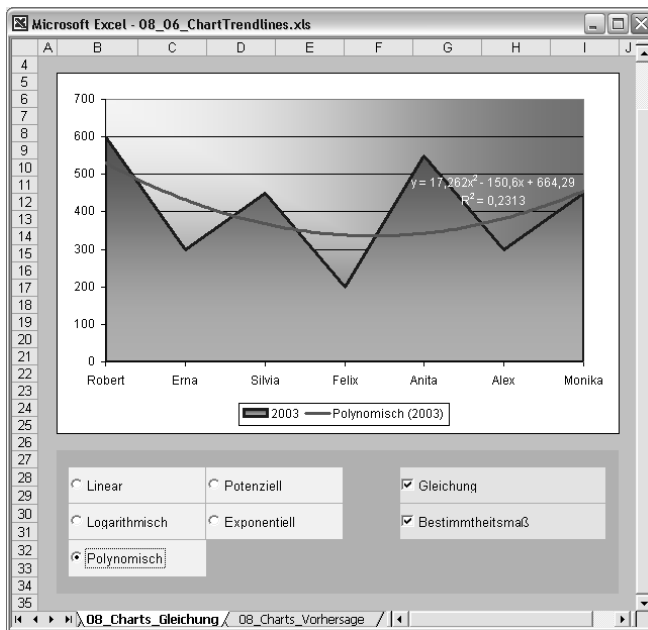


Abbildung 206: Gleichungen und Bestimmtheitsmaß ein- oder ausblenden

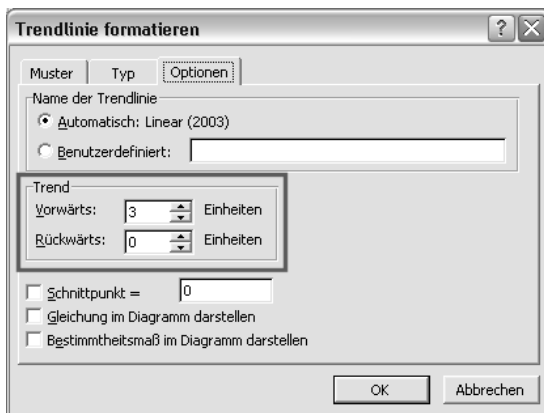


Abbildung 207: Vorwärts- oder Rückwärtstrend bestimmen

Das Ganze lässt sich per Programmierung etwas komfortabler gestalten. Wir fügen in unser Tabellenblatt zwei Rollbalken `ScrollBar` aus der Symbolleiste `STEUERELEMENT-TOOLBOX` ein; eine für den Vorwärtstrend und eine für den Rückwärtstrend (siehe Abbildung 207). Beiden Rollbalken hinterlegen wir eine Ereignisprozedur. Diese befindet sich hinter dem entsprechenden Tabellenblatt.

In jeder der Ereignisprozeduren legen wir einen minimalen Wert von 0 und einen maximalen Wert von 10 fest. Das bedeutet, dass der Rollbalken auf diese Werte begrenzt ist und somit nur innerhalb dieser Werte gescrollt werden kann. Für den Vorwärtstrend wird die Eigenschaft `Forward` verwendet und für den Rückwärtstrend `Backward`. Damit auf dem Tabellenblatt die aktuell eingestellte Einheit

Grundlagen

Allgemein

Datum
Zeit

tungen

Steuer
elementeBefehl
leisten

Objekt

Dia-
gramm

Ereignisse

User-
FormWeb/
Mail

Extern

Ge-
misch-
tes

Specia

zu sehen ist, fügen wir oberhalb jedes Rollbalkens einen entsprechenden Text ein. Die If...Then...Else-Entscheidung ist lediglich dazu da, den Text EINHEIT oder EINHEITEN korrekt anzuzeigen. Wenn der Wert auf 1 eingestellt ist, soll EINHEIT angezeigt werden, sonst EINHEITEN.

```
'=====
' Auf CD      Buchdaten\Beispiele\08_Diagramme\
' Dateiname   08_06_Trendlines.xls
' Tabelle     08_Charts_Vorhersage
' Ereignis    Tabelle4(08_Charts_Vorhersage)
'=====

' Vorwärtstrend
Private Sub ScrollBar1_Change()
    Dim chtSC As SeriesCollection

    Set chtSC = ActiveSheet.ChartObjects(1).Chart.SeriesCollection

    With ScrollBar1
        .Min = 0
        .Max = 10
    End With

    ' Vorwärtstrend
    chtSC(1).Trendlines(1).Forward = ScrollBar1

    ' Anzeige der aktuellen Einheit in Zelle D27
    If ScrollBar1 = 1 Then
        Range("D27") = ScrollBar1 & " Einheit"
    Else
        Range("D27") = ScrollBar1 & " Einheiten"
    End If

    Set chtSC = Nothing
End Sub

' Rückwärtstrend
Private Sub ScrollBar2_Change()
    Dim chtSC As SeriesCollection

    Set chtSC = ActiveSheet.ChartObjects(1).Chart.SeriesCollection

    With ScrollBar2
        .Min = 0
        .Max = 10
    End With

    ' Rückwärtstrend
    chtSC(1).Trendlines(1).Backward = ScrollBar2

    ' Anzeige der aktuellen Einheit in Zelle D30
    If ScrollBar2 = 1 Then
        Range("D30") = ScrollBar2 & " Einheit"
```

```
Else
    Range("D30") = ScrollBar2 & " Einheiten"
End If

Set chtSC = Nothing
End Sub
```

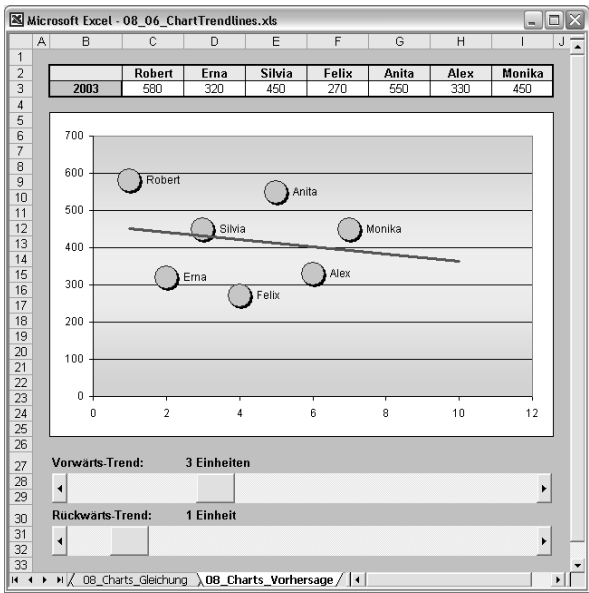


Abbildung 208: Vorwärts- und Rückwärtstrend per Rollbalken einstellen

265 Achsen formatieren

Dieses Rezept zeigt auf, wie die Achsen Y oder X formatiert werden können. Wir wenden uns in diesem Beispiel hauptsächlich der Skalierung und Ausrichtung zu. Zusätzlich soll die Achsenbeschriftung erfasst werden können. Es sind noch viel mehr Einstellungen möglich, vor allem im Bereich der Formatierung. Die Hauptsache an diesem Beispiel ist jedoch, dass Sie sehen können, wie das Objekt Achsen *Axes* dimensioniert und referenziert werden kann. Den Rest der Einstellungen können Sie selbst ermitteln, indem Sie den Makro-Rekorder benutzen und die gewünschten Schritte aufzeichnen.

Um Achsen manuell zu formatieren, klicken Sie mit der rechten Maustaste auf die X- oder Y-Achse und wählen aus dem Kontextmenü den Eintrag **ACHSE FORMATIEREN**. Beachten Sie, dass das Dialogfenster fünf Registerkarten enthält. In der Registerkarte **MUSTER** können Formatierungen vorgenommen werden.

Um manuell die Achsenbeschriftung vorzunehmen, klicken Sie mit der rechten Maustaste auf die Diagrammfläche und wählen aus dem Kontextmenü den Eintrag **DIAGRAMMOPTIONEN**. Aktivieren Sie die Registerkarte **TITEL** und geben Sie wahlweise einen Text bei **RUBRIKENACHSE (X)** oder **GRÖSSENACHSE (Y)** ein. Wenn Sie einige Sekunden warten, können Sie im Dialogfenster sehen, dass die Beschriftung in der Vorschau angezeigt wird.

Grundlagen

Allgemein

Datum und Zeit

Formeln

Statistik

Diagramme

Objekte

Diagramm

Ereignisse

User-Form

Web/Mail

Extern

Gemischtes

Spezial

Da wir in Bezug auf Trendlinien relativ viele Einstellungen vornehmen, werden wir mit zwei Tabellenblättern arbeiten. Auf dem ersten Tabellenblatt befinden sich die Datenquelle sowie das Diagramm (siehe Abbildung 209).

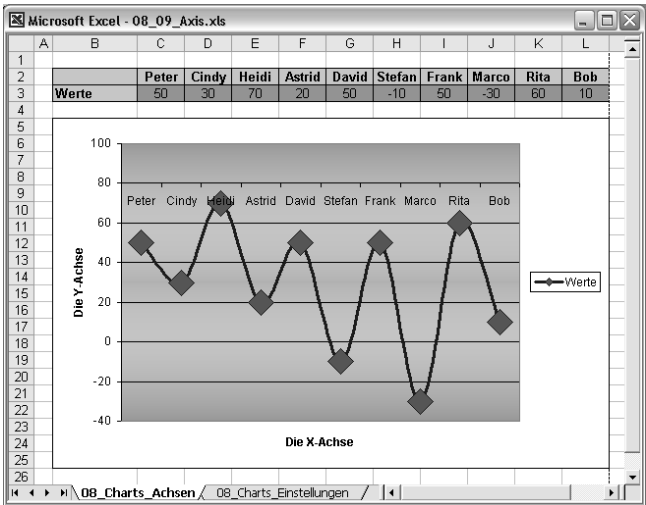


Abbildung 209: Datenquelle und Diagramm auf dem ersten Tabellenblatt

Auf dem zweiten Tabellenblatt können die Einstellungen eingetragen werden. Per Klick auf eine Schaltfläche im zweiten Tabellenblatt werden die eingegebenen Werte an das Diagramm übergeben (siehe Abbildung 210). Wir werden zudem mittels DATEN | GÜLTIGKEIT einige Einschränkungen vornehmen. Diese sind entsprechend gekennzeichnet. Wir können uns damit bereits im Vorfeld einen Teil an Programmieraufwand ersparen.

Für den Schnittpunkt der Rubrikenachse `Crosses` gibt es verschiedene Einstellungen. Sie können diese der nachfolgenden Tabelle entnehmen. Wenn Sie mit der Einstellung `BENUTZERDEFINIERT` arbeiten, muss zusätzlich der gewünschte Wert mitgeliefert werden. Verwenden Sie dazu die Eigenschaft `CrossesAt`.

Crosses-Konstante	Index	Beschreibung
xlAutomatic	-4105	Automatisch
xlMinimum	-4114	Minimum
xlMaximum	2	Maximum
xlCustom	4	Benutzerdefiniert (erfordert ergänzend die Eigenschaft CrossesAt)

Tabelle 67: Schnittpunkt der Rubrikenachse

Sie können die logarithmische Skalierung `ScaleType` ein- oder ausschalten. Bei ausgeschalteter logarithmischer Skalierung gilt `Linear`. Das heißt, dass Sie in VBA entweder die Konstanten `xlLogarithmic` oder `xlLinear` verwenden können.

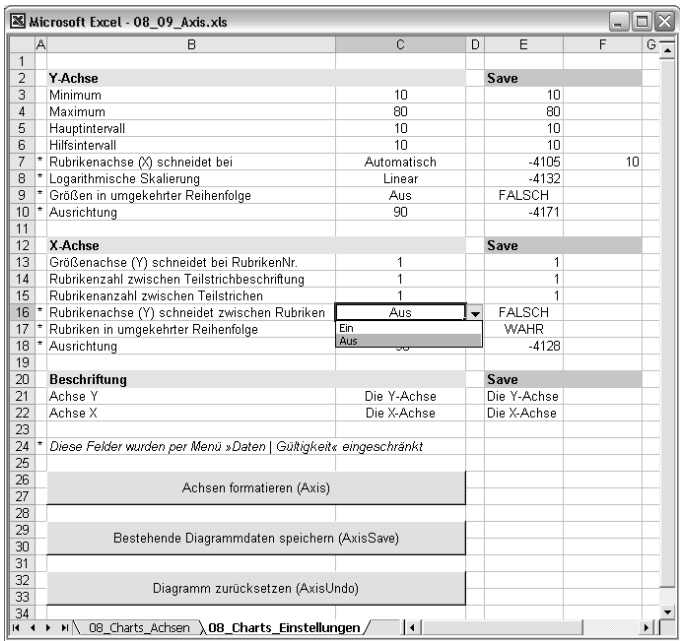


Abbildung 210: Datenquelle und Diagramm auf dem ersten Tabellenblatt

Achtung Wenn Sie die logarithmische Einstellung wählen, müssen Sie bedenken, dass Negativ- oder Nullwerte nicht korrekt angezeigt werden können. Excel wird Sie bei einem Fehler entsprechend mit einer Meldung darauf hinweisen.

ScaleType-Konstante	Index	Beschreibung
xlLinear	-4132	Linear
xlLogarithmic	-4133	Logarithmisch

Tabelle 68: Logarithmische Skalierung

Die Beschreibung der weiteren Werte entnehmen Sie den Kommentaren in der Prozedur.

```
'=====
' Auf CD      Buchdaten\Beispiele\08_Diagramme\
' Dateiname  08_07_Axis.xls
' Tabelle    08_Charts_Achsen
' Tabelle    08_Charts_Einstellungen
' Modul      mdl_01_Axis
'=====

Sub Axis()
    Dim cht As Chart
    Dim chtAX As Axes
    Dim WS2 As Worksheet
```

```

Dim strInput As Integer

Set cht = Worksheets(1).ChartObjects(1).Chart
Set chtAX = cht.Axes
Set WS2 = Worksheets(2)

' *** Y-ACHSE ***
With chtAX(xlValue)
    .MinimumScale = WS2.Range("C3")      ' Minimum
    .MaximumScale = WS2.Range("C4")      ' Maximum
    .MajorUnit = WS2.Range("C5")         ' Hauptintervall
    .MinorUnit = WS2.Range("C6")         ' Hilfsintervall

    ' Rubrikenachse schneidet bei
    If WS2.Range("C7") = "Automatisch" Then
        .Crosses = xlAutomatic
    ElseIf WS2.Range("C7") = "Minimum" Then
        .Crosses = xlMinimum
    ElseIf WS2.Range("C7") = "Maximum" Then
        .Crosses = xlMaximum
    ElseIf WS2.Range("C7") = "Benutzerdefiniert" Then
        .Crosses = xlCustom
        strInput = Application.InputBox _
            ("Geben Sie den gewünschten Werte ein", _
            Title:="X-Achse schneidet bei:", Type:=1)
        .CrossesAt = strInput
    End If

    ' Logarithmische Reihenfolge
    If WS2.Range("C8") = "Logarithmisch" Then
        .ScaleType = xlLogarithmic
    ElseIf WS2.Range("C8") = "Linear" Then
        .ScaleType = xlLinear
    End If

    ' Größen in umgekehrter Reihenfolge
    If WS2.Range("C9") = "Ein" Then
        .ReversePlotOrder = True
    Else
        .ReversePlotOrder = False
    End If

    ' Ausrichtung
    .TickLabels.Orientation = WS2.Range("C10")
End With

' *** X-ACHSE ***
With chtAX(xlCategory)
    ' Größenachse (Y) schneidet bei RubrikenNr.
    .CrossesAt = WS2.Range("C13")

```

```
' Rubrikenzahl zwischen Teilstrichbeschriftung
.TickLabelSpacing = WS2.Range("C14")

' Rubrikenanzahl zwischen Teilstrichen
.TickMarkSpacing = WS2.Range("C15")

' Rubrikenachse (Y) schneidet zwischen Rubriken
If WS2.Range("C16") = "Ein" Then
    .AxisBetweenCategories = True
Else
    .AxisBetweenCategories = False
End If

' Rubriken in umgekehrter Reihenfolge
If WS2.Range("C17") = "Ein" Then
    .ReversePlotOrder = True
Else
    .ReversePlotOrder = False
End If

' Ausrichtung
.TickLabels.Orientation = WS2.Range("C19")
End With

' *** ACHSEN-BESCHRIFTUNG ***
With cht
    .Axes(xlCategory, xlPrimary).AxisTitle.Characters.Text _
        = WS2.Range("C21")
    .Axes(xlValue, xlPrimary).AxisTitle.Characters.Text _
        = WS2.Range("C22")
End With

Set cht = Nothing
Set chtAX = Nothing
Set WS2 = Nothing
End Sub
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

266 Achseinstellungen speichern

Dieses Rezept baut auf dem vorherigen auf. Die Idee, wie sie hier zu sehen ist, kann auch auf andere Diagrammkomponenten abgeleitet werden.

Wie Sie der Abbildung 209 entnehmen können, sind neben der Formatierung von Achsen noch zwei weitere Prozeduren enthalten (entsprechend den Schaltflächen). Sie sind dazu da, die bestehenden Diagrammoptionen zu speichern und auf Wunsch wiederherzustellen.

Wie Sie wissen, können Sie einmal ausgeführte VBA-Prozeduren nicht rückgängig machen, außer Sie erstellen eigens dazu eine Prozedur. Bei Diagrammen ist es oft so, dass es später schwierig ist, die ursprünglichen Werte wieder herzustellen, ohne dass das Diagramm neu erstellt werden muss. Wir möchten Ihnen deshalb an dieser Stelle anhand der Achsen einen Weg zeigen, wie Sie solche Einstellungen sichern können. Die Daten werden in dem zweiten Tabellenblatt in die Spalte E geschrieben. Von da können sie später zur Wiederherstellung wieder bezogen werden (siehe nachfolgendes Rezept).

```
'=====
' Auf CD      Buchdaten\Beispiele\08_Diagramme\
' Dateiname   08_07_Axis.xls
' Tabelle     08_Charts_Achsen
' Tabelle     08_Charts_Einstellungen
' Modul       mdl_02_AxisSave
'=====
```

```
Sub AxisSave()
    Dim cht As Chart
    Dim chtAX As Axes
    Dim WS2 As Worksheet

    Set cht = Worksheets(1).ChartObjects(1).Chart
    Set chtAX = cht.Axes
    Set WS2 = Worksheets(2)

    ' Bestehende Diagramm-Einstellungen sichern
    With WS2
        ' *** Y-ACHSE ***
        .Range("E3") = chtAX(xlValue).MinimumScale
        .Range("E4") = chtAX(xlValue).MaximumScale
        .Range("E5") = chtAX(xlValue).MajorUnit
        .Range("E6") = chtAX(xlValue).MinorUnit
        .Range("E7") = chtAX(xlValue).Crosses
        .Range("F7") = chtAX(xlValue).CrossesAt
        .Range("E8") = chtAX(xlValue).ScaleType
        .Range("E9") = chtAX(xlValue).ReversePlotOrder
        .Range("E10") = chtAX(xlValue).TickLabels.Orientation

        ' *** X-ACHSE ***
        .Range("E13") = chtAX(xlCategory).CrossesAt
        .Range("E14") = chtAX(xlCategory).TickLabelSpacing
        .Range("E15") = chtAX(xlCategory).TickMarkSpacing
        .Range("E16") = chtAX(xlCategory).AxisBetweenCategories
        .Range("E17") = chtAX(xlCategory).ReversePlotOrder
        .Range("E18") = chtAX(xlCategory).TickLabels.Orientation

        ' *** AXSEN-BESCHRIFTUNG ***
        .Range("E21") = cht.Axes(xlCategory, xlPrimary). _
            AxisTitle.Characters.Text
        .Range("E22") = cht.Axes(xlValue, xlPrimary). _
            AxisTitle.Characters.Text
    End With

    Set cht = Nothing
    Set chtAX = Nothing
    Set WS2 = Nothing
End Sub
```

267 Achseneinstellungen rückgängig machen

In diesem Rezept, das auf den vorangegangenen beiden basiert, werden dem Diagramm die gesicherten Werte wieder zugewiesen. Die Werte wurden im vorangegangenen Rezept gespeichert, indem sie in das zweite Tabellenblatt in die Spalte E geschrieben wurden.

Den Tabellen 67 und 68 können Sie entnehmen, dass `ScaleType` und `Crosses` mit Konstanten arbeiten. Beim Schreiben der Sicherheitswerte in die Tabelle `08_CHARTS_EINSTELLUNGEN` ist Ihnen vielleicht aufgefallen, dass Excel nicht die Namen der Konstanten, sondern deren Index verwendet. Bei der Wiederherstellung eines Diagramms muss dies berücksichtigt werden. Mittels `If...Then...Else`-Entscheidungen lässt sich das umsetzen.

Bei der Ausrichtung von Achsen sind Werte zwischen -90 und 90 möglich. Bei den Werten -90, 0 und 90 wird dabei ein Index verwendet. Sämtliche anderen Werte werden entsprechend der Eingabe gespeichert. Das bedeutet, wenn Sie bei der Ausrichtung einen Wert von 20 verwenden, wird bei der Sicherung auch 20 ins Tabellenblatt geschrieben. Wenn Sie jedoch den Wert 90 einstellen, wird nicht der Wert 90, sondern der Index -4171 ins Tabellenblatt geschrieben. Die drei Werte, die mit einem Index arbeiten, können Sie der Tabelle 69 entnehmen.

Konstante	Index für die Ausrichtung	Beschreibung
xlUpward	-4171	90
xlHorizontal	-4128	0
xlDownward	-4170	-90

Tabelle 69: Index für die Ausrichtung

```
'=====
' Auf CD      Buchdaten\Beispiele\08_Diagramme\
' Dateiname  08_07_Axis.xls
' Tabelle     08_Charts_Axis
' Tabelle     08_Charts_Settings
' Modul       mdl_03_AxisUndo
'=====

Sub AxisUndo()
    Dim cht As Chart
    Dim chtAX As Axes
    Dim WS2 As Worksheet

    Set cht = Worksheets(1).ChartObjects(1).Chart
    Set chtAX = cht.Axes
    Set WS2 = Worksheets(2)

    ' *** Y-ACHSE ***
    With chtAX(xlValue)
        .MinimumScale = WS2.Range("E3") ' Minimum
        .MaximumScale = WS2.Range("E4") ' Maximum
        .MajorUnit = WS2.Range("E5") ' Hauptintervall
        .MinorUnit = WS2.Range("E6") ' Hilfsintervall

        ' Rubrikenachse schneidet bei
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

```

If WS2.Range("E7") = -4105 Then
    .Crosses = xlAutomatic
ElseIf WS2.Range("E7") = -4114 Then
    .Crosses = xlMinimum
ElseIf WS2.Range("E7") = 2 Then
    .Crosses = xlMaximum
ElseIf WS2.Range("E7") = 4 Then
    .Crosses = xlCustom
End If
.CrossesAt = WS2.Range("F7")

' Logarithmische Reihenfolge
If WS2.Range("E8") = -4133 Then
    .ScaleType = xlLogarithmic
ElseIf WS2.Range("E8") = -4132 Then
    .ScaleType = xlLinear
End If

' Größen in umgekehrter Reihenfolge
.ReversePlotOrder = WS2.Range("E9")

' Ausrichtung
If WS2.Range("E10") = -4171 Then
    chtAX(xlValue).TickLabels.Orientation = "90"
ElseIf WS2.Range("E10") = -4128 Then
    chtAX(xlValue).TickLabels.Orientation = "0"
ElseIf WS2.Range("E10") = -4170 Then
    chtAX(xlValue).TickLabels.Orientation = "-90"
Else
    chtAX(xlValue).TickLabels.Orientation = WS2.Range("E10")
End If
End With

' *** X-ACHSE ***
With chtAX(xlCategory)
    ' Größenachse (Y) schneidet bei RubrikenNr.
    .CrossesAt = WS2.Range("C13")

    ' Rubrikenzahl zwischen Teilstrichbeschriftung
    .TickLabelSpacing = WS2.Range("C14")

    ' Rubrikenanzahl zwischen Teilstrichen
    .TickMarkSpacing = WS2.Range("C14")

    ' Rubrikenachse (Y) schneidet zwischen Rubriken
    If WS2.Range("C16") = "Ein" Then
        .AxisBetweenCategories = True
    Else
        .AxisBetweenCategories = False
    End If

    ' Rubriken in umgekehrter Reihenfolge
    If WS2.Range("C17") = "Ein" Then

```

```
        .ReversePlotOrder = True
    Else
        .ReversePlotOrder = False
    End If

    ' Ausrichtung
    If WS2.Range("E18") = -4171 Then
        chtAX(xlCategory).TickLabels.Orientation = "90"
    ElseIf WS2.Range("E18") = -4128 Then
        chtAX(xlCategory).TickLabels.Orientation = "0"
    ElseIf WS2.Range("E18") = -4170 Then
        chtAX(xlCategory).TickLabels.Orientation = "-90"
    Else
        chtAX(xlCategory).TickLabels.Orientation = WS2.Range("E18")
    End If
End With

' *** AXSEN-BESCHRIFTUNG ***
cht.Axes(xlCategory, xlPrimary).AxisTitle. _
    Characters.Text = WS2.Range("E21")
cht.Axes(xlValue, xlPrimary).AxisTitle. _
    Characters.Text = WS2.Range("E22")

Set cht = Nothing
Set chtAX = Nothing
Set WS2 = Nothing
End Sub
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

268 Primär- oder Sekundärachse

Bei Diagrammen mit mehr als zwei Säulen haben Sie die Möglichkeit, eine zweite Achse einzu-
blenden. Die zweite Achse nennt sich »Sekundärachse« und wird meistens über die zweite Säule
gelegt. Eine Sekundärachse lässt sich nicht für jeden Diagrammtyp festlegen. Der Tabelle 70 kön-
nen Sie entnehmen, welche Typen sich eignen bzw. sich nicht eignen.

Sekundärachse lässt sich einblenden	Sekundärachse lässt sich nicht einblenden
Säulen	Netz
Balken	Oberfläche
Linie	Blase
Kreis	Kurs
Punkt	Zylinder (3-D)
Fläche	Kegel (3-D)
Ring	Pyramide (3-D)

Tabelle 70: Übersicht, auf welche Diagrammtypen die Sekundärachse angezeigt werden kann

Um diese Einstellung ohne VBA vorzunehmen, klicken Sie mit der rechten Maustaste auf die
gewünschte Datenreihe im Diagramm. Wählen Sie aus dem Kontextmenü den Eintrag DATEN-
REIHEN FORMATIEREN. Aktivieren Sie die Registerkarte AXSEN. In diesem Dialogfenster können
Sie nun zwischen Primär- oder Sekundärachse auswählen.

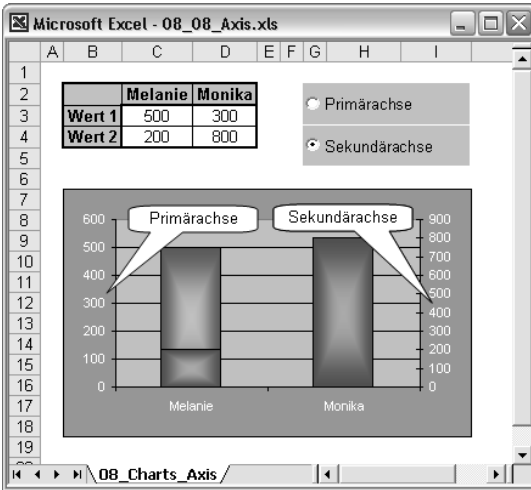


Abbildung 211: Primär- oder Sekundärachse einblenden

In unserer Beispielmappe haben wir zwei Optionsfelder aus der Symbolleiste **STEUERELEMENT-TOOLBOX** ins Tabellenblatt eingefügt. Sie können so bequem per Klick auf das gewünschte Optionsfeld wahlweise die Primär- oder die Sekundärachse einblenden. Damit das möglich ist, werden wir jedem Optionsfeld eine Ereignisprozedur hinterlegen.

Die Programmierung ist denkbar einfach. Jeder Achse ist ein Wert zugeordnet. Für die Primärachse ist es der Wert 1 und für die Sekundärachse der Wert 2. Diesen Wert übergeben wir an die Eigenschaft `AxisGroup` der zweiten Datenreihe.

```
'=====
' Auf CD      Buchdaten\Beispiele\08_Diagramme\
' Dateiname   08_08_PrimarySecondaryAxis.xls
' Ereignis    Tabelle1(08_Charts_Axis)
'=====

' Primärachse
Private Sub OptionButton1_Click()
    ActiveSheet.ChartObjects(1).SeriesCollection(2).AxisGroup = 1
End Sub

' Sekundärachse
Private Sub OptionButton2_Click()
    ActiveSheet.ChartObjects(1).SeriesCollection(2).AxisGroup = 2
End Sub
```

269 Datentabelle formatieren

Sie können innerhalb eines Diagramms wahlweise die zugehörige Datentabelle ein- oder ausblenden. In der Datentabelle selbst können Sie zudem wählen, ob ein Legendensymbol angezeigt werden soll.

Manuell erreichen Sie diese Einstellungen, indem Sie mit der rechten Maustaste auf die Diagrammfläche klicken und aus dem Kontextmenü den Eintrag **DIAGRAMMOPTIONEN** wählen. Aktivieren Sie die Registerkarte **DATENTABELLE**.

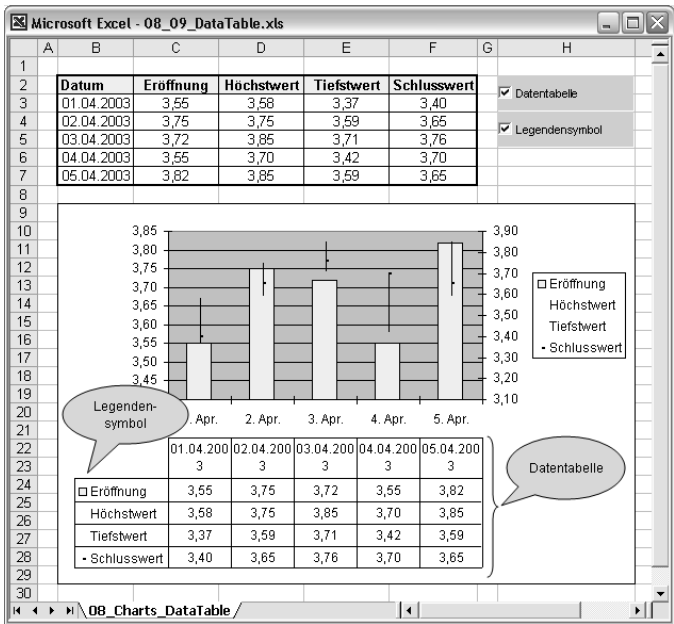


Abbildung 212: Datentabelle und Legendenymbol

Wir möchten diesen Weg etwas verkürzen. In unserem Tabellenblatt fügen wir zwei Kontrollkästchen aus der Symbolleiste STEUERELEMENT-TOOLBOX ein. Diese beiden Kontrollkästchen sind an das Tabellenblatt gebunden, in dem sie erstellt wurden. Um sie zu programmieren, werden wir deshalb dem Tabellenblatt zwei Ereignis-Prozeduren hinterlegen, für jedes Kontrollkästchen eine.

Die Eigenschaft zur Anzeige der Datentabelle lautet `HasDataTable`. Wenn der Wert auf `True` gesetzt ist, wird die Datentabelle eingeblendet, bei `False` ausgeblendet. Dasselbe gilt für das Legenden-symbol `ShowLegendKey`, welches sich in der Datentabelle `DataTable` befindet.

Das Legenden-symbol kann nicht aktiviert werden, wenn die Datentabelle nicht eingeblendet ist. Deshalb wird das Legenden-symbol ebenfalls ausgeblendet, sobald die Datentabelle deaktiviert wird. Bei der Aktivierung des Legenden-symbols muss auf jeden Fall auch die Datentabelle eingeblendet werden, ansonsten würde sich der Debugger melden. Beides wird in den nachfolgenden Prozeduren berücksichtigt.

```
'=====
' Auf CD      Buchdaten\Beispiele\08_Diagramme\
' Dateiname  08_09_DataTable.xls
' Ereignis   Tabelle1(08_Charts_DataTable)
'=====

' Datentabelle
Private Sub CheckBox1_Click()
    With ActiveSheet.ChartObjects(1).Chart
        If CheckBox1 Then
            .HasDataTable = True
        Else
            .DataTable.ShowLegendKey = False
        End If
    End With
End Sub
```

```

        CheckBox2 = False
        .HasDataTable = False
    End If
End With
End Sub

' =====
' Auf CD      Buchdaten\Beispiele\08_Diagramme\
' Dateiname   08_09_DataTable.xls
' Ereignis    Tabelle1(08_Charts_DataTable)
' =====

' Legendensymbol
Private Sub CheckBox2_Click()
    With ActiveSheet.ChartObjects(1).Chart
        If CheckBox2 Then
            .HasDataTable = True
            CheckBox1 = True
            .DataTable.ShowLegendKey = True
        Else
            .DataTable.ShowLegendKey = False
        End If
    End With
End Sub

```

270 Legende verwalten

Die Legende kann in einem Diagramm wahlweise aus- oder eingeblendet werden. Wenn Sie das Legendensymbol einblenden, haben Sie zudem die Möglichkeit zu wählen, wo es platziert werden soll:

- Unten (xlBottom)
- Ecke (xlCorner)
- Oben (xlTop)
- Rechts (xlRight)
- Links (xlLeft)

Falls in Ihrem Diagramm keine Legende angezeigt wird, können Sie diese einblenden, indem Sie mit der rechten Maustaste auf die Diagrammfläche klicken. Wählen Sie aus dem Kontextmenü den Eintrag **DIAGRAMMOPTIONEN**. Aktivieren Sie die Registerkarte **LEGENDE**. In diesem Dialogfenster können Sie sowohl die Legende aktivieren als auch die Platzierung bestimmen.

In unserem Beispiel stellen wir diese Möglichkeiten auf dem Tabellenblatt dar, indem wir sechs Optionsfelder einfügen. Wir wählen diese aus der Symbolleiste **STEUERELEMENT-TOOLBOX** aus. Wie in den vorangegangenen Beispielen bereits beschrieben, werden wir auch hier mit Ereignisprozeduren arbeiten. Diese werden direkt hinter das betroffene Tabellenblatt geschrieben. Es sind sechs Prozeduren erforderlich, für jede Auswahl eine.

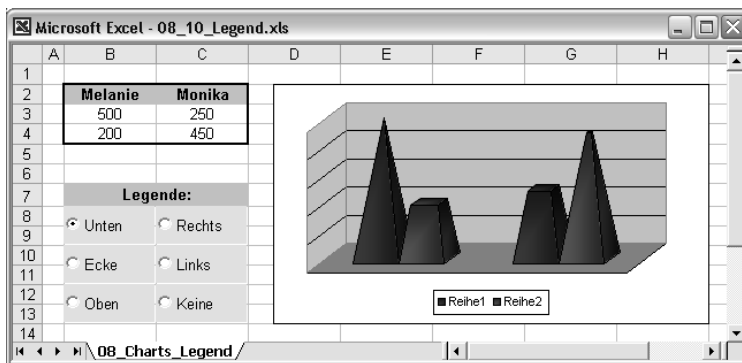


Abbildung 213: Legende positionieren oder ausblenden

Die fünf Optionsfelder für die Ausrichtung der Legende sind alle gleich aufgebaut. Zuerst wird die Legende eingeblendet (`HasLegend=True`). Damit stellen wir sicher, dass die Legende auf jeden Fall eingeblendet ist. Bei ausgeblendeter Legende würde die Prozedur sonst zu einer Fehlermeldung führen. Danach wird dem Optionsfeld die entsprechende Konstante für die Ausrichtung zugewiesen (`Legend.Position = ...`).

```

' =====
' Auf CD      Buchdaten\Beispiele\08_Diagramme\
' Dateiname  08_10_Legend.xls
' Ereignis   Tabelle1(08_Charts_Legend)
' =====

' Unten
Private Sub OptionButton1_Click()
    With ActiveSheet.ChartObjects(1).Chart
        .HasLegend = True
        .Legend.Position = xlBottom
    End With
End Sub

' Ecke
Private Sub OptionButton2_Click()
    With ActiveSheet.ChartObjects(1).Chart
        .HasLegend = True
        .Legend.Position = xlCorner
    End With
End Sub

' Oben
Private Sub OptionButton3_Click()
    With ActiveSheet.ChartObjects(1).Chart
        .HasLegend = True
        .Legend.Position = xlTop
    End With
End Sub

```

Grund-
lagenAllge-
meinDatu
Zeit

tungen

Steuer
elemenBefehl
leisten

Objekt

Dia-
grammEreig-
nisseUser-
FormWeb/
Mail

Extern

Ge-
misch
tes

Specia

```

' Rechts
Private Sub OptionButton4_Click()
    With ActiveSheet.ChartObjects(1).Chart
        .HasLegend = True
        .Legend.Position = xlRight
    End With
End Sub

' Links
Private Sub OptionButton5_Click()
    With ActiveSheet.ChartObjects(1).Chart
        .HasLegend = True
        .Legend.Position = xlLeft
    End With
End Sub

```

Das sechste Optionsfeld ist dazu da, die Legende auszublenden (HasLegend = False).

```

' =====
' Auf CD      Buchdaten\Beispiele\08_Diagramme\
' Dateiname   08_10_Legend.xls
' Ereignis    Tabelle1(08_Charts_Legend)
' =====

' Legende
Private Sub OptionButton6_Click()
    ActiveSheet.ChartObjects(1).Chart.HasLegend = False
End Sub

```

271 3-D-Oberflächen-Diagramm rotieren lassen

3-D-Diagramme lassen sich in verschiedenen Richtungen drehen und wenden. Wenn Sie ein 3-D-Diagramm auf herkömmliche Weise rotieren lassen möchten, dann klicken Sie mit der rechten Maustaste auf das Diagramm und wählen aus dem Kontextmenü den Eintrag 3D-ANSICHT. Der Abbildung 214 können Sie entnehmen, welche Einstellungsmöglichkeiten sich anbieten.

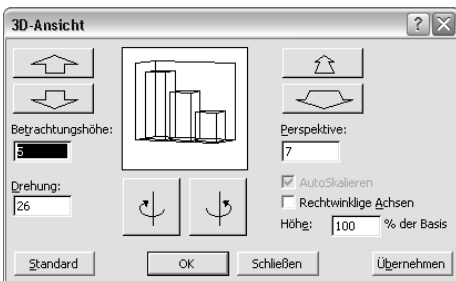


Abbildung 214: 3-D-Ansicht

Wenn Sie in diesem Dialogfenster Ihre Einstellungen vornehmen, verändert sich zwar die Ansicht in der Vorschau, aber daran können Sie noch lange nicht erkennen, wie Ihr eigenes Diagramm letztendlich aussieht. Zwar können Sie auf die Schaltfläche ÜBERNEHMEN klicken und die Einstellungen an das Diagramm übergeben, dennoch erweist sich das Ganze als recht umständlich.

Wir werden in unserer Mappe drei Rollbalken aus der Symbolleiste STEUERELEMENT-TOOLBOX verwenden; eine für die Betrachtungshöhe, eine weitere für die Drehung und eine Letzte für die Perspektive. Wir hinterlegen jedem dieser Rollbalken eine eigene Ereignisprozedur.

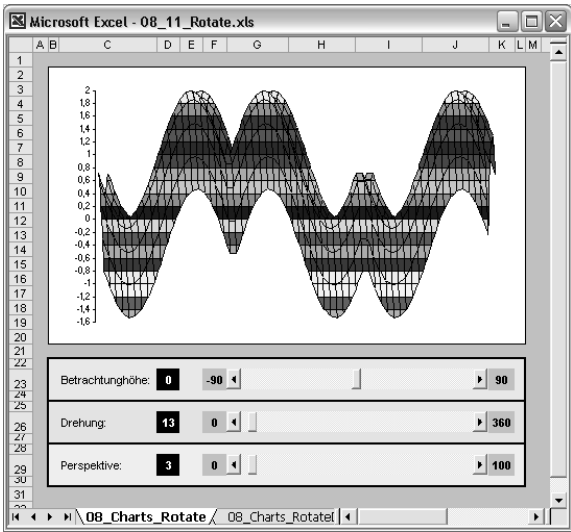


Abbildung 215: Oberflächendiagramm rotieren lassen

Es sind drei Ereignisprozeduren erforderlich, für jeden Rollbalken Scrollbar eine. Wir legen in jeder Prozedur den erlaubten Höchst- und Tiefstwert fest. Danach wird dem Diagramm der aktuelle Wert des Rollbalkens übergeben. Dies geschieht laufend, sobald einer der Rollbalken bewegt wird. Damit der aktuell eingestellte Wert im Tabellenblatt sichtbar ist, wird dieser bei allen drei Rollbalken in eine Zelle geschrieben.

Eigenschaft	Minimal-Wert	Maximal-Wert	Beschreibung
Elevation	-90	90	Betrachtungshöhe
Rotation	0	360	Drehung
Perspective	0	100	Perspektive

Tabelle 71: Eigenschaften zur Rotation

```
'=====
' Auf CD      Buchdaten\Beispiele\08_Diagramme\
' Dateiname  08_11_Rotate.xls
' Ereignis   Tabelle1(08_Charts_Rotate)
'=====

' Betrachtungshöhe
Private Sub ScrollBar1_Change()
    With ScrollBar1
        .Min = -90
        .Max = 90
    End With
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

```

ActiveSheet.ChartObjects(2).Chart.Elevation = ScrollBar1
Range("D23") = ScrollBar1
End Sub

' Drehung
Private Sub ScrollBar2_Change()
    With ScrollBar2
        .Min = 0
        .Max = 360
    End With

    ActiveSheet.ChartObjects(2).Chart.Rotation = ScrollBar2
    Range("D26") = ScrollBar2
End Sub

' Perspektive
Private Sub ScrollBar3_Change()
    With ScrollBar3
        .Min = 0
        .Max = 100
    End With

    ActiveSheet.ChartObjects(2).Chart.Perspective = ScrollBar3
    Range("D29") = ScrollBar3
End Sub

```

Tipp
Farben in einem Oberflächen-Diagramm verändern:

Ein Rechtsklick auf ein Oberflächen-Diagramm erweckt den Eindruck, man könne die einzelnen Datenreihen nicht formatieren, denn die Anzeige eines Kontextmenüs bleibt aus. Das Oberflächen-Diagramm ist wohl das einzige, dessen Farben über die Legende formatiert werden müssen. Klicken Sie mit der rechten Maustaste auf eins der Legendensymbole und wählen Sie aus dem Kontextmenü den Eintrag **LEGENDENEINTRAG FORMATIEREN**. Nehmen Sie die gewünschten Einstellungen vor. Nach dem Schließen des Dialogfensters werden Sie feststellen können, dass die Einstellungen auch auf die Datenreihen übernommen wurden.

Anzahl Farben in einem Oberflächen-Diagramm:

Die Anzahl der Farben in einem Oberflächen-Diagramm kann über die Größenachse (Z) bestimmt werden. Aktivieren Sie die Registerkarte **SKALIERUNG** dieser Achse. Je kleiner der Wert **HAUPTINTERVALL** eingestellt wird, desto mehr Farben werden angezeigt.

272 Ein benutzerdefiniertes Diagramm erstellen

Oftmals verbringt man viel Zeit damit, ein Diagramm so zu gestalten, dass man am Ende mit der Optik zufrieden ist. Nun wäre es natürlich von Vorteil, wenn man das einmal entworfene Design auch auf andere Diagramme anwenden könnte. Sie haben in Excel die Möglichkeit, Ihre Kreation als benutzerdefinierten Diagrammtyp abzuspeichern.

Nachfolgend zeigen wir Ihnen zuerst den manuellen Weg auf:

1. Gestalten Sie Ihr Diagramm.
2. Klicken Sie mit der rechten Maustaste auf die Diagrammfläche und wählen Sie aus dem Kontextmenü den Eintrag **DIAGRAMMTYP**. Sie können alternativ auch bei aktiviertem Diagramm den Menüpunkt **DIAGRAMM | DIAGRAMMTYP** auswählen. Ein Dialogfenster wird geöffnet.
3. Aktivieren Sie die Registerkarte **BENUTZERDEFINIERT** TYPEN.
4. Aktivieren Sie das Optionsfeld **BENUTZERDEFINIERT**.
5. Klicken Sie auf die Schaltfläche **HINZUFÜGEN**.
6. Geben Sie einen Namen und eine Beschreibung zu dem Diagramm ein.
7. Schließen Sie alle Dialogfenster, indem Sie jeweils auf die Schaltfläche **OK** klicken.

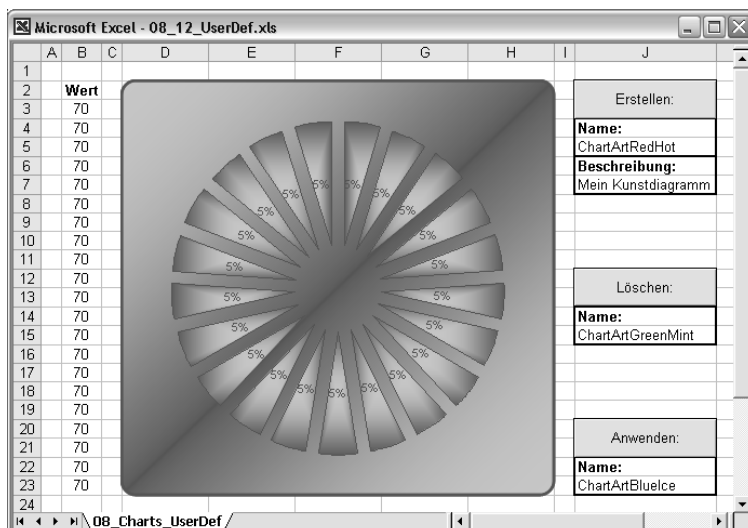


Abbildung 216: Benutzerdefinierter Diagrammtyp

Wir werden insgesamt drei Prozeduren erstellen. Die erste Prozedur zeigt, wie man einen benutzerdefinierten Diagrammtypen auf Knopfdruck erstellen kann. Die zweite Prozedur löscht einen benutzerdefinierten Typen und die dritte Prozedur wendet den Typen an. Alle drei Prozeduren sind in unserem Tabellenblatt mit Schaltflächen aus der Symbolleiste **FORMULAR** verknüpft.

Ein benutzerdefiniertes Diagramm ist nicht an eine Mappe gebunden. Das bedeutet, der Typ wird nach dem Erstellen in allen Mappen bzw. in Excel selbst zur Verfügung stehen. Die Anweisung wird deshalb durch **Application** eingeleitet. Danach folgt die Methode **AddChartAutoFormat**. Sie ist für das Erstellen des benutzerdefinierten Formates zuständig. Dieser Methode werden zwei Werte übergeben. Zum einen der Name **Name**, den der neue Typ erhalten soll, zum anderen ein Text zur Beschreibung **Description**. Beide Werte werden aus je einer Zelle bezogen. Am Ende der Prozedur wird eine Dialogbox angezeigt, welche mitteilt, dass der benutzerdefinierte Typ erstellt wurde.

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

```
'=====
' Auf CD      Buchdaten\Beispiele\08_Diagramme\
' Dateiname   08_12_UserDef.xls
' Modul       mdl_01_CreateUserDef
'=====

Sub CreateUserDef()
    Application.AddChartAutoFormat _
        Chart:=ActiveSheet.ChartObjects(1).Chart, _
        Name:= Range("J5"), _
        Description:= Range("J7")

    MsgBox "Der benutzerdefinierte Diagrammtyp " & _
        "" & Range("J5")& "" & _
        " wurde erstellt.", _
        vbInformation
End Sub
```

273 Einen benutzerdefinierten Diagrammtypen löschen

Dieses Rezept basiert auf dem vorherigen. Damit der Diagrammtyp bequem per Knopfdruck gelöscht werden kann, erstellen wir eine entsprechende Prozedur. Da Excel einen benutzerdefinierten Typen ohne Rückfrage löschen würde, haben wir eine Sicherheit angebracht. Beim Start der Prozedur wird erst gefragt, ob der Diagrammtyp wirklich gelöscht werden soll. Dies geschieht in einer If...Then...Else-Entscheidung. Wenn die Löschabfrage bestätigt wird, dann wird die Methode DeleteChartAutoFormat ausgeführt. Ihr wird der Name des benutzerdefinierten Typs übergeben. Der Name wurde zu Beginn der Prozedur einer Variablen übergeben. Diese bezieht ihren Wert aus einer Zelle in unserer Tabelle.

Für den Fall, dass in der Zelle ein falscher Name steht, wurde zudem eine Fehlerbehandlung eingebaut.

```
'=====
' Auf CD      Buchdaten\Beispiele\08_Diagramme\
' Dateiname   08_12_UserDef.xls
' Modul       mdl_02_DeleteUserDef
'=====

Sub DeleteUserDef()
    Dim strQuestion As String
    On Error GoTo ErrorHandler

    strQuestion = MsgBox("Wollen Sie wirklich löschen?", vbYesNo)
    If strQuestion = vbNo Then
        Exit Sub
    Else
        Application.DeleteChartAutoFormat _
            Name:=Range("J15")

        MsgBox "Der benutzerdefinierte Diagrammtyp " & _
            "" & Range("J15") & "" & _
            " wurde gelöscht.", _
```

```
        vbCritical
    End If

    Exit Sub
ErrorHandler: MsgBox "Falscher Name", vbCritical
End Sub
```

274 Einen benutzerdefinierten Diagrammtypen anwenden

Dieses Rezept basiert auf den beiden vorangegangenen. Zu guter Letzt soll es auch per Knopfdruck möglich sein, einen benutzerdefinierten Typen an das Diagramm zu übergeben. Die Methode dazu lautet `ApplyCustomType`. Ihr wird die Konstante `xlUserDefined` übergeben, welche aussagt, dass es sich um einen benutzerdefinierten Typen handelt. Als weiterer Wert wird der Name des Typen übergeben. Die entsprechende Variable haben wir auch hier zu Beginn der Prozedur deklariert. Der Wert stammt ebenfalls aus einer Zelle. Eine Fehlerbehandlung verhindert den Start des Debuggers für den Fall, dass ein falscher Name eingegeben wurde.

```
' =====
' Auf CD      Buchdaten\Beispiele\08_Diagramme\
' Dateiname  08_12_UserDef.xls
' Modul      mdl_03_ApplyUserDef
' =====

Sub ApplyUserDef()
    On Error GoTo ErrorHandler
    ActiveSheet.ChartObjects(1).Chart. _
        ApplyCustomType _
            ChartType:=xlUserDefined, _
            TypeName:=Range("J23")
    Exit Sub
ErrorHandler: MsgBox "Falscher Name", vbCritical
End Sub
```

275 Pivot-Diagramm

Das Thema Pivot-Tabellen ist sehr umfangreich. Sie eignen sich hervorragend, um Tabellen übersichtlich in verschiedenster Form darzustellen. Spalten und Zeilen lassen sich auf einfachste Weise vertauschen. Wir wollen jedoch an dieser Stelle den Fokus ganz klar auf Pivot-Diagramme legen. Näheres zum Thema Pivot-Tabellen können Sie der Kategorie »Auswertungen« entnehmen.

In unserem Beispiel befindet sich bereits eine bestehende Pivot-Tabelle. Aus einer bestehenden Pivot-Tabelle ein Pivot-Diagramm zu erstellen, ist denkbar einfach. Aktivieren Sie die Pivot-Tabelle, so dass die zugehörige Symbolleiste angezeigt wird. Klicken Sie darin auf das Diagramm-Symbol. Das Pivot-Diagramm wird auf einem eigenen Diagrammblatt erstellt. Wenn Sie lieber ein eingebettetes Diagramm auf Ihrem Tabellenblatt haben möchten, können Sie dies ändern. Wählen Sie aus dem Menü **DIAGRAMM** die Auswahl **SPEICHERORT** aus. In dem Dialogfenster, das sich nun öffnet, können Sie die Umstellung vornehmen.

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

Ein Pivot-Diagramm wird nach dem Erstellen als gestapeltes Säulen-Diagramm angezeigt. Den Diagrammtypen können Sie anschließend umstellen, indem Sie auf den Menüpunkt **DIAGRAMM** klicken und darin den Eintrag **DIAGRAMMTYP** auswählen.

Hinweis

Das Diagramm ist fest mit der Datenquelle, also der Pivot-Tabelle, verknüpft. Wenn sich Daten in der zugrunde liegenden Tabelle ändern, wird automatisch auch das Diagramm angepasst. Wenn Sie in Ihrer Pivot-Tabelle die Spalten- und Zeilenköpfe vertauschen, reflektiert sich das ebenfalls im Diagramm. Auch umgekehrt ist das der Fall. Wenn Sie in Ihrem Diagramm eine Umstellung vornehmen, wirkt sich das automatisch auch auf die Pivot-Tabelle aus.

Die Legende in einem Pivot-Diagramm lässt sich nicht manuell verschieben. Wenn Sie die Legende verschieben möchten, klicken Sie diese mit der rechten Maustaste an und wählen aus dem Kontextmenü den Eintrag **LEGENDE FORMATIEREN**. In dem Dialogfenster, das sich nun öffnet, können Sie die gewünschte Position einstellen.

Der Titel eines Pivot-Diagramms lässt sich weder in der Größe verändern noch verschieben.

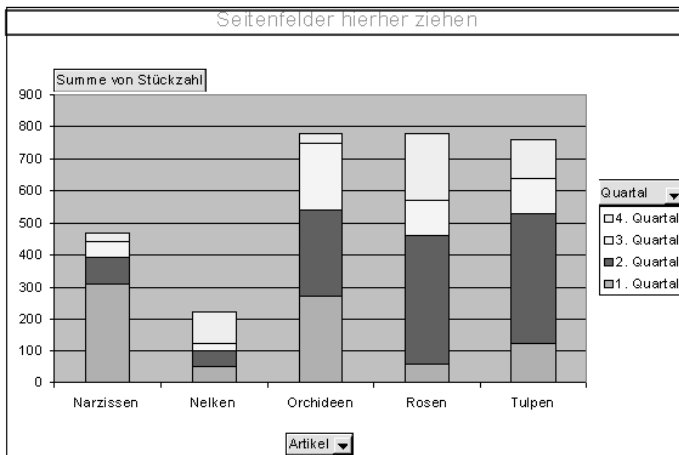


Abbildung 217: Pivot-Diagramm

Achtung

Sie können Ihr Diagramm zwar nach Lust und Laune gestalten, müssen jedoch wissen, dass die ganze Mühe umsonst war, sobald sich die Daten verändern. Bei einer Veränderung der Daten wird das Pivot-Diagramm neu aufgebaut und einige der Formatierungen gehen verloren. Der gewählte Diagrammtyp bleibt zwar erhalten, aber gegebenenfalls vorgenommene Veränderungen z.B. in der Darstellung der Säulen gehen verloren. Falls Sie die Gestaltung schnell wieder zuweisen möchten, empfiehlt es sich entweder, per Makro-Rekorder die gesamte Formatierung einmal aufzuzeichnen, oder noch besser, einen benutzerdefinierten Typen zu erstellen. Eine entsprechende Beschreibung finden Sie im Rezept 274. Der benutzerdefinierte Typ muss ebenfalls nach jeder Umstellung neu zugewiesen werden.

Wir werden an dieser Stelle darauf verzichten, VBA-Code darzustellen. Das Ansprechen des Diagramms auf die gleiche Weise erfolgt, wie dies auch bei gewöhnlichen Diagrammen der Fall ist.

276 Alle Diagramme drucken und löschen

Diagramme können einerseits als eingebettete Diagramme auf einem Tabellenblatt eingebunden werden. Andererseits haben Sie die Möglichkeit, ein Diagramm auf einem separaten Diagrammblatt anzuzeigen. Diese Auswahl kann im letzten Schritt des Diagramm-Assistenten getroffen werden. Sie haben zudem die Möglichkeit, dies im Nachhinein nach Belieben umzustellen. Der einfachste Weg führt über das Kontextmenü. Klicken Sie mit der rechten Maustaste auf die Diagrammfläche und wählen Sie aus dem Kontextmenü den Eintrag **SPICHERORT**. In dem Dialogfenster, das sich nun öffnet, können Sie Ihre Auswahl treffen.

Beim Löschen ist dieser Unterschied zu beachten. Sie unterscheiden zwischen dem Löschen von eingebetteten Diagrammen oder Diagrammblättern.

Achtung

Denken Sie beim Löschen daran, dass Sie VBA-Prozeduren nicht rückgängig machen können, außer Sie erstellen eigens dazu eine Prozedur.

Tipp

Speichern Sie vor dem Löschen die Datei ab. Wenn Sie dann versehentlich ein Diagramm löschen, schließen Sie die Mappe ohne erneutes Speichern. Öffnen Sie die Mappe erneut, sind die Daten wie vor dem Löschen des Diagramms vorhanden.

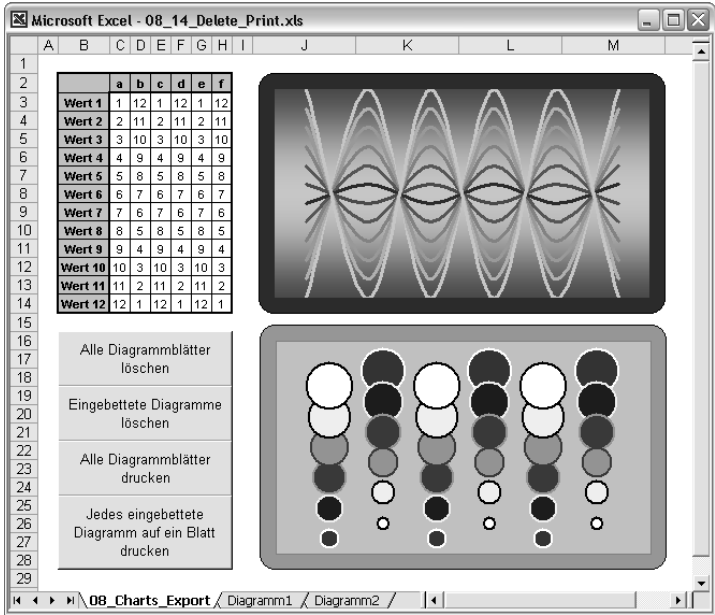


Abbildung 218: Diagramme löschen oder drucken

Grundlagen

Allgemein

Datum
Zeit

Formeln

Steuerelemente

Befehle
leisten

Objekte

Diagramm

Ereignisse

User-Form

Web/
Mail

Extern

Gemischtes

Spezial

Alle Diagrammblätter löschen:

```
'=====
' Auf CD      Buchdaten\Beispiele\08_Diagramme\
' Dateiname  08_14_Delete_Print.xls
' Modul      mdl_01_SheetsDelete
'=====

Sub SheetsDelete()
    With ActiveWorkbook
        If .Charts.Count > 0 Then
            .Charts.Delete
        Else
            MsgBox "Es sind keine Diagramme vorhanden."
        End If
    End With
End Sub
```

Wenn Sie nur ein Diagrammblatt löschen möchten, geben Sie zusätzlich den Index oder den Namen des Diagrammblattes an:

```
ActiveWorkbook.Charts(1).Delete
ActiveWorkbook.Charts("Diagramm1").Delete
```

Alle eingebetteten Diagramme löschen:

```
'=====
' Auf CD      Buchdaten\Beispiele\08_Diagramme\
' Dateiname  08_14_Delete_Print.xls
' Modul      mdl_02_EmbeddedDelete
'=====

Sub ChartsDelete()
    Dim cht as ChartObject

    For Each cht In ActiveSheet.ChartObjects
        cht.Delete
    Next cht
End Sub
```

Um nur ein einzelnes eingebettetes Diagramm zu löschen, verzichten Sie auf die Schleife und geben den Index des Diagramms an:

```
ActiveSheet.ChartObjects(1).Delete
```

277 Diagramme drucken

Wenn Sie ein Tabellenblatt mit eingebetteten Grafiken ausdrucken, werden die Daten gemeinsam mit den Diagrammen auf ein Blatt gedruckt. Sie haben jedoch auch die Möglichkeit, die eingebetteten Diagramme einzeln auf je ein separates Blatt Papier auszudrucken. Die Diagramme werden so vergrößert, dass sie ein ganzes A4-Blatt füllen.

Sie können das auch manuell erreichen, indem Sie ein Diagramm aktivieren und dann die Schaltfläche für DRUCKEN in der Symbolleiste STANDARD betätigen. Auf diese Weise wird jedoch immer nur ein Diagramm ausgedruckt. Wenn in Ihrem Tabellenblatt mehrere Diagramme enthalten sind, können Sie diese mit gedrückter **[Strg]**-Taste markieren. Wenn Sie dann auf DRUCKEN klicken, wird jedes Diagramm in Vollgröße auf ein eigenes Blatt gedruckt.

Hinweis

Per VBA lässt sich das mit einer For...Each-Schleife umsetzen. Wie immer bei dieser Art von Schleife muss dabei die Hierarchie der Objekte bekannt sein. Dem Objekt ChartObjects ist das Objekt ChartObject (ohne »s«) untergeordnet. Das bedeutet, dass alle ChartObject innerhalb von ChartObjects des aktiven Tabellenblattes gedruckt werden sollen.

```
'=====
' Auf CD      Buchdaten\Beispiele\08_Diagramme\
' Dateiname   08_14_Delete_Print.xls
' Modul       mdl_02_PrintEmbedded
'=====

Sub PrintAllEmbeddedCharts()
    Dim chtObj As ChartObject

    For Each chtObj In ActiveSheet.ChartObjects
        chtObj.Chart.PrintOut
    Next chtObj
End Sub
```

Wenn Sie alle Diagrammblätter drucken möchten:

```
'=====
' Auf CD      Buchdaten\Beispiele\08_Diagramme\
' Dateiname   08_14_Delete_Print.xls
' Modul       mdl_01_SheetsPrint
'=====

Sub SheetsPrint()
    With ActiveWorkbook
        If .Charts.Count > 0 Then
            .Charts.PrintOut
        Else
            MsgBox "Es sind keine Diagramme vorhanden."
        End If
    End With
End Sub
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

278 Diagramm-Säulen mit Grafiken

Sie können in Excel anstelle von gewöhnlichen Datenreihen, wie zum Beispiel Säulen, auch Grafiken verwenden.

Manuell erreichen Sie dies, indem Sie mit der rechten Maustaste auf die Datenreihe klicken. Wählen Sie aus dem Kontextmenü den Eintrag DATENREIHEN FORMATIEREN. In der Registerkarte MUSTER klicken Sie auf die Schaltfläche FÜLLEFFEKTE. Im nächsten Dialogfenster aktivieren Sie die Registerkarte GRAFIK. Klicken Sie dort auf die Schaltfläche GRAFIK AUSWÄHLEN. Wählen Sie im Explorer die gewünschte Grafik aus. Danach können sämtliche offenen Dialogfenster geschlossen werden.

Wie Sie sehen können, ist das Ganze etwas umständlich. Wir wollen einerseits diesen Vorgang automatisieren und andererseits bei negativen Zahlen eine andere Grafik zuweisen als bei positiven Zahlen. Dazu benötigen wir zwei Grafiken. Für positive Zahlen verwenden wir ein Smilie, das lächelt, für negative Zahlen eines, das weint. Beide Grafiken sind auf der CD bei den Beispielen zu den Diagrammen zu finden (*smile.gif* und *sad.gif*).

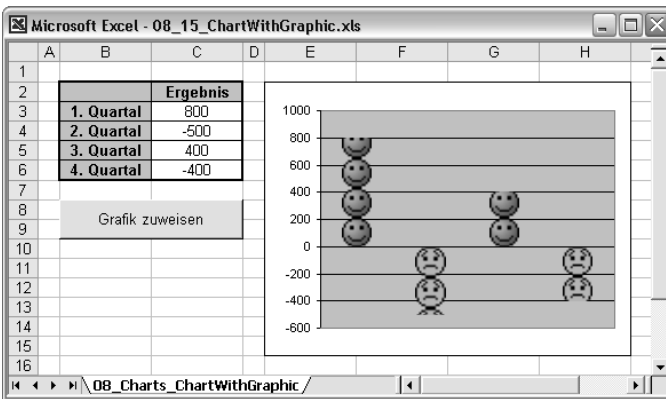


Abbildung 219: Diagramm mit Grafiksäulen

In unserem Beispiel arbeiten wir mit zwei Schleifen. Die äußere Schleife zählt die Datenpunkte und arbeitet diese einzeln ab. Um zu ermitteln, ob es sich bei dem Datenpunkt um einen negativen oder positiven Wert handelt, benötigen wir eine zweite Schleife. In diesem Fall ist es eine `For...Each`-Schleife. Diese Schleife spricht in unserer Datentabelle jede Zelle an und prüft in einer `If...Then...Else`-Entscheidung, ob der Wert positiv ≥ 0 oder negativ ist. Je nachdem wird die eine oder andere Grafik zugewiesen.

Da sich die Grafiken im selben Verzeichnis befinden wie die Mappe selbst, können wir den Befehl `ActiveWorkbook.Path` verwenden. Diesen verbinden wir (&) mit dem Dateinamen der Grafik.

Sie können zudem wählen, wie die Grafik dargestellt werden soll. Dabei gibt es drei Möglichkeiten. Sie können diese der Tabelle 72 entnehmen.

Konstante zu PictureFormat	Beschreibung
xlStrech	Strecken. Das Bild wird nur einmal angezeigt und je nach Säulenhöhe entsprechend gestreckt.
xlStack	Das Bild wird gestapelt angezeigt. Das bedeutet, dass die Anzahl der Grafiken je nach Säulenhöhe variiert.

Tabelle 72: Konstanten zu PictureFormat

Konstante zu PictureFormat	Beschreibung
xlStackScale	Sie können damit die Anzahl der Grafiken pro Säule erhöhen oder reduzieren. Dazu ist ein weiterer Befehl erforderlich: PictureStakkUnit:=x. Wobei x durch den gewünschten Wert ersetzt wird. Die Grafik wird dabei entsprechend gedehnt oder zusammengezogen.

Tabelle 72: Konstanten zu PictureFormat (Forts.)

<pre>'===== ' Auf CD Buchdaten\Beispiele\08_Diagramme\ ' Dateiname 08_15_ChartWithGraphic.xls ' Modul mdl_01_ChartWithGraphic '=====</pre>		Grundlagen
<pre>Sub ChartWithGraphic() Dim cht As Chart Dim chtSC As Series Dim i As Integer Dim c As Range Set cht = ActiveSheet.ChartObjects(1).Chart Set chtSC = cht.SeriesCollection(1) For i = 1 To chtSC.Points.Count For Each c In Range("C3:C6") If chtSC.Values(i) >= 0 Then chtSC.Points(i).Fill.UserPicture PictureFile:= _ ActiveWorkbook.Path & "\smile.gif", _ PictureFormat:=xlStack Else chtSC.Points(i).Fill.UserPicture PictureFile:= _ ActiveWorkbook.Path & "\sad.gif", _ PictureFormat:=xlStack End If Next c Next i Set cht = Nothing Set chtSC = Nothing End Sub</pre>		Allgemein
		Datum Zeit
		tungen
		Steuer elemente
		Befehl leisten
		Objekt
		Diagramm
		Ereignisse
		User- Form
		Web/ Mail
		Extern
		Gemis- ches
		Specia

279 Diagramm als Grafik exportieren

Auf sehr einfache Weise ist es möglich, ein Diagramm als Grafik abzuspeichern. Sie können dabei die Grafik in einem gewünschten Format abspeichern. Beispielsweise als Dateityp *.gif oder *.jpg. Das Diagramm wird dabei exportiert (Export).

Sie können sowohl ein eingebettetes Diagramm als auch ein ganzes Diagrammblatt exportieren. Wenn sich Ihr Diagramm auf einem eigenen Diagrammblatt befindet, wird dieses Blatt über das

Objekt Sheet angesprochen. Dies im Gegensatz zu Tabellenblättern, deren Objektname explizit Worksheet lautet. Mit dem Objekt Sheet können sowohl Diagramme als auch Tabellenblätter angesprochen werden.

Ein eingebettetes Diagramm exportieren:

```
'=====
' Auf CD      Buchdaten\Beispiele\08_Diagramme\
' Dateiname   08_16_Export.xls
' Tabelle     08_Charts_Export
' Modul       mdl_01_ExportEmbed
'=====

Sub ExportEmbeddedChart()
    ' Ein eingebettetes Diagramm exportieren
    ActiveSheet.ChartObjects(1).Chart.Export _
        ("C:\ArtChart.gif")
End Sub
```

Ein ganzes Diagrammblatt exportieren:

```
'=====
' Auf CD      Buchdaten\Beispiele\08_Diagramme\
' Dateiname   08_16_Export.xls
' Tabelle     08_Charts_Export
' Modul       mdl_02_ExportSheet
'=====

Sub ExportChart()
    ' Ein Diagrammblatt exportieren
    Sheets(1).ChartObjects(1).Chart.Export _
        ("C:\ArtChart.gif")
End Sub
```

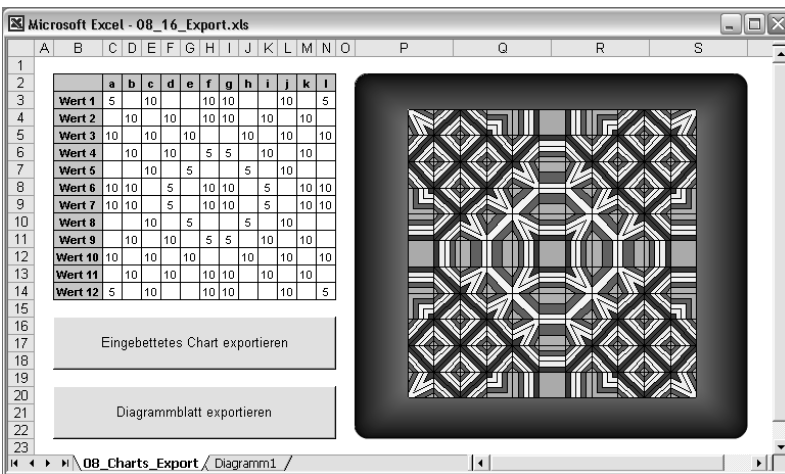


Abbildung 220: Ein Diagramm als Grafik abspeichern

280 Objektnamen in Diagrammen ermitteln

Oftmals ist es recht schwierig herauszufinden, wie sich das Objekt nennt, das per VBA angesprochen werden soll. Wir haben einerseits die Möglichkeit, den Makro-Rekorder zu benutzen und mit dessen Hilfe die Namen aufzuzeichnen. Die Objektnamen können jedoch auch mit Hilfe einer kleinen Prozedur ermittelt werden.

In unserer Beispielmappe ist eine einfache Tabelle enthalten. Basierend auf dieser Tabelle wurde ein Diagramm erstellt, in dem alle gewünschten Objekte angezeigt werden. Unsere Prozedur ist so aufgebaut, dass Sie per Klick auf die Schaltfläche den Namen des aktiven Diagramm-Objektes erfahren. Ein Nachrichtenfenster zeigt die gewünschte Information an.

Die If...Then...Else-Entscheidung prüft, ob ein Diagramm-Element aktiviert wurde. Wenn nicht, wird eine Fehlermeldung ausgegeben, ansonsten der Objektname.

```
'=====
' Auf CD      Buchdaten\Beispiele\08_Diagramme\
' Dateiname   08_17_SelectedObject.xls
' Modul       mdl_01_Object
'=====
```

```
Sub ChartObject()
    If ActiveChart Is Nothing Then
        MsgBox "Es wurde kein Diagramm-Element aktiviert", vbCritical
    ElseIf TypeName(Selection) = "ChartObject" Then
        Selection.Activate
        MsgBox TypeName(Selection)
    End If
End Sub
```

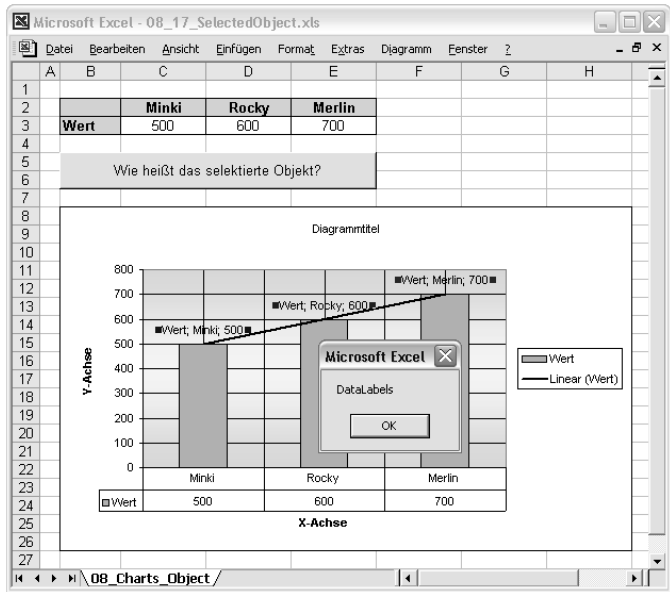


Abbildung 221: Objektnamen ermitteln

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

281 Zellfarben an Datenreihe übergeben

In diesem Beispiel können Sie im ersten Tabellenblatt in der Zelle B3 die Farbe für die erste Datenreihe hinterlegen. In der Zelle C3 wird die Farbe für die zweite Datenreihe bestimmt. Per Klick auf die Schaltfläche SÄULEN-FARBE ANPASSEN werden die in den beiden Zellen festgelegten Farben an die beiden Datenreihen übergeben.

Tipp

Die Farbpalette, die in der Symbolleiste FORMAT zu finden ist, lässt sich auskoppeln. Klicken Sie mit der linken Maustaste auf den kleinen Pfeil links neben dem Symbol FÜLLFARBE. Die Farbpalette wird angezeigt. Fassen Sie die Farbpalette mit gedrückter linker Maustaste am oberen Rand an und ziehen Sie diese nach unten ins Tabellenblatt. Die Farbpalette lässt sich nun frei auf dem Tabellenblatt platzieren. Durch einen Klick auf das Kreuz in der linken oberen Ecke der Farbpalette können Sie diese wieder schließen. Dieser Vorgang wirkt sich auf die gesamte Excel-Anwendung aus.

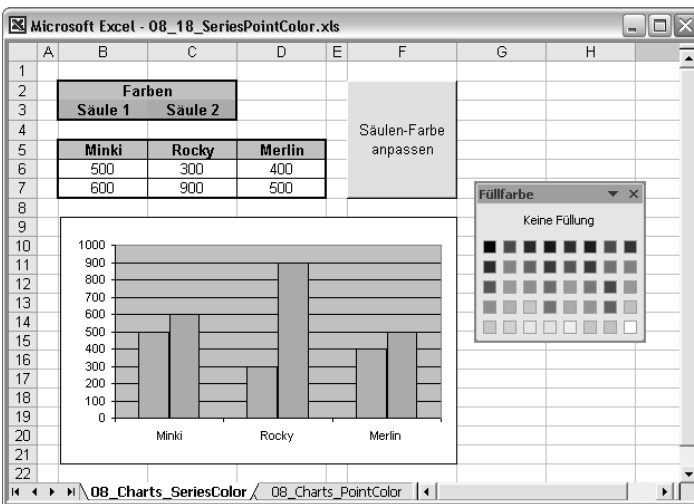


Abbildung 222: Farbe aus Zellen an Datenreihen (Säulen) übergeben

```

=====
' Auf CD      Buchdaten\Beispiele\08_Diagramme\
' Dateiname  08_18_SeriesPointColor.xls
' Tabelle    08_Charts_SeriesColor
' Modul      mdl_01_SeriesColor
=====

```

```

Sub SeriesColor()
    Dim cht As Chart
    Dim chtSC As SeriesCollection

    Set cht = Worksheets(1).ChartObjects(1).Chart
    Set chtSC = cht.SeriesCollection

```



```
' Farben an Datenreihen übergeben
chtSC(1).Interior.ColorIndex = Range("B3").Interior.ColorIndex
chtSC(2).Interior.ColorIndex = Range("C3").Interior.ColorIndex

Set cht = Nothing
Set chtSC = Nothing
End Sub
```

282 Zellfarben an Datenpunkte übergeben

Dieses Rezept ist dem vorherigen sehr ähnlich. Sie finden es auf dem zweiten Tabellenblatt in derselben Mappe. Der Unterschied besteht lediglich darin, dass wir mit nur einer Datenreihe arbeiten. Hier wird die Farbe nicht an die Datenreihe selbst, sondern an die einzelnen Datenpunkte übergeben. Die verschiedenen Farben werden in den entsprechenden Zellen hinterlegt (B3, C3 und D3).

```
' =====
' Auf CD      Buchdaten\Beispiele\08_Diagramme\
' Dateiname  08_18_SeriesPointColor.xls
' Tabelle    08_Charts_PointColor
' Modul      mdl_02_PointColor
' =====

Sub PointColor()
    Dim cht As Chart
    Dim chtSC As Series

    Set cht = Worksheets(2).ChartObjects(1).Chart
    Set chtSC = cht.SeriesCollection(1)

    ' Farben an Datenpunkte übergeben
    With chtSC
        .Points(1).Interior.ColorIndex = _
            Range("B3").Interior.ColorIndex
        .Points(2).Interior.ColorIndex = _
            Range("C3").Interior.ColorIndex
        .Points(3).Interior.ColorIndex = _
            Range("D3").Interior.ColorIndex
    End With

    Set cht = Nothing
    Set chtSC = Nothing
End Sub
```

283 Minuswerte invertieren

Falls in Ihrem Diagramm Minuswerte auftauchen, können Sie diese Datenpunkte invertiert darstellen. Sie heben sich damit deutlich von den positiven Werten ab.

Um die Einstellung manuell vorzunehmen, klicken Sie mit der rechten Maustaste auf die Datenreihe. Wählen Sie aus dem Kontextmenü den Eintrag DATENREIHEN FORMATIEREN. Aktivieren Sie die Registerkarte MUSTER und aktivieren Sie darin das Kontrollkästchen INVERTIEREN FALLS NEGATIV.

Grund-lagen

Allge-mein

Datu-Zeit

tungen

Steuer-elemen

Befehl-leisten

Objekt

Dia-gramm

Ereig-nisse

User-Form

Web/Mail

Extern

Ge-mischtes

Specia

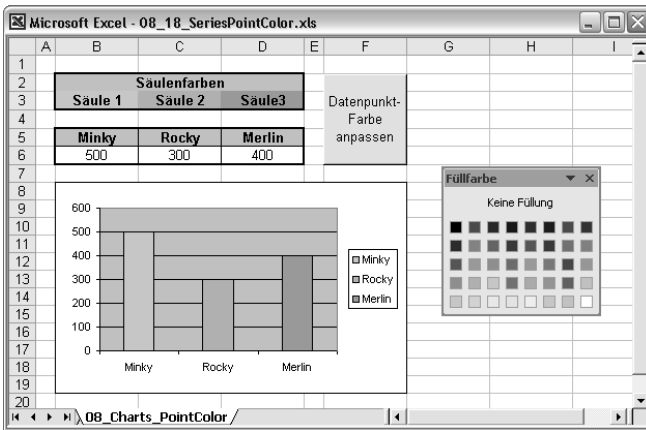


Abbildung 223: Farben aus Zellen an Datenpunkte übergeben

In unserem Beispiel möchten wir nicht den umständlichen Umweg über ein Menü gehen. Die Invertierung soll sich direkt im Tabellenblatt aktivieren oder deaktivieren lassen. In der Zelle C25 haben wir über DATEN | GÜLTIGKEIT | LISTE ein DropDown erstellt. In diesem DropDown-Feld kann die Auswahl JA (aktivieren) oder NEIN (deaktivieren) ausgewählt werden.

Damit die Einstellung direkt ausgeführt wird, arbeiten wir mit einer Ereignisprozedur, welche direkt an das Tabellenblatt gebunden ist. Mehr zum Thema Ereignisprozeduren finden Sie in der Kategorie »Ereignisse«. Für unsere Zwecke eignet sich am besten ein `Workbook_Change`-Ereignis. Damit dieses nur ausgeführt wird, wenn sich in der Zelle C25 etwas ändert, werden wir die Ziel-Adresse `Target.Address` entsprechend festlegen. Dies geschieht über eine `If...Then...Else`-Entscheidung.

Eine weitere Entscheidung verwenden wir, um zu prüfen, ob in der Zelle C25 ein JA oder ein NEIN ausgewählt wurde. Bei einem JA soll die Invertierung aktiviert werden (`InvertIfNegative`), bei einem NEIN deaktiviert.

```
' =====
' Auf CD      Buchdaten\Beispiele\08_Diagramme\
' Dateiname  08_19_Invert.xls
' Ereignis   Tabelle1(08_Charts_Invert)
' =====

' Minuswerte invertieren
Private Sub Worksheet_Change(ByVal Target As Range)
    Dim cht As Chart
    Dim chtSC As SeriesCollection

    Set cht = ActiveSheet.ChartObjects(1).Chart
    Set chtSC = cht.SeriesCollection

    ' Ziel-Adresse festlegen
    If Target.Address <> "$C$25" Then
        Exit Sub
    End If
```

```
' Invertierung festlegen
If Range("C25") = "Ja" Then
    chtSC(1).InvertIfNegative = True
Else
    chtSC(1).InvertIfNegative = False
End If

Set cht = Nothing
Set chtSC = Nothing

End Sub
```

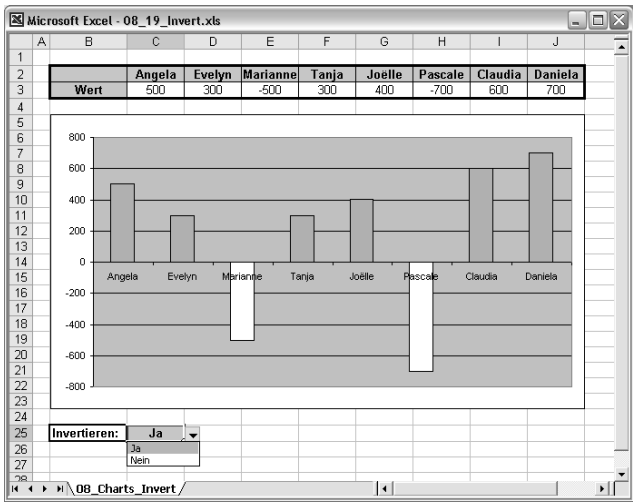


Abbildung 224: Minuswerte invertiert darstellen

284 Diagramm mit bedingter Formatierung (Säulenfarbe)

Leider gibt es bis heute für Diagramme, oder besser gesagt, für das Objekt `Chart`, weder eine `BEDINGTE FORMATIERUNG` noch Ereignisprozeduren, das heißt, Sie können nicht erreichen, dass sich automatisch auf Veränderung eines Wertes in einer Zelle auch die Säulenfarbe ändert, außer Sie verwenden beispielsweise das Ereignis `Worksheet_Change`. Dieses bezieht sich jedoch nicht direkt auf ein Diagramm, sondern auf eine Zelle oder einen Bereich im Tabellenblatt. Genau einen solchen Fall wollen wir in unserem nächsten Beispiel darstellen.

Damit das Diagramm laufend, bei Eingabe eines Wertes im Bereich `D3:D9`, die Säulenfarbe verändert, müssen wir eine Ereignisprozedur verwenden, die an das Tabellenblatt gebunden ist. Das bedeutet, dass wir den VBA-Code direkt hinter das betroffene Tabellenblatt schreiben.

Unser Beispiel besteht aus einer kleinen Datenmenge sowie einem Säulendiagramm. Die bedingte Formatierung soll so festgelegt werden, dass bei einem Wert von kleiner oder gleich 300 (≤ 300) die Säulenfarbe rot dargestellt wird. Bei einem Wert 300 oder kleiner/gleich 500 (> 300 und ≤ 500) wird die Säulenfarbe gelb dargestellt. Alle Zahlen, die größer als 500 (> 500) sind, werden in grüner Farbe angezeigt.

Grundlagen

Allgemein

Datum
Zeit

Diagrammen

Steuerelemente

Befehle
leisten

Objekt

Diagramm

Ereignisse

User-Form

Web/
Mail

Extern

Gemischtes

Spezial

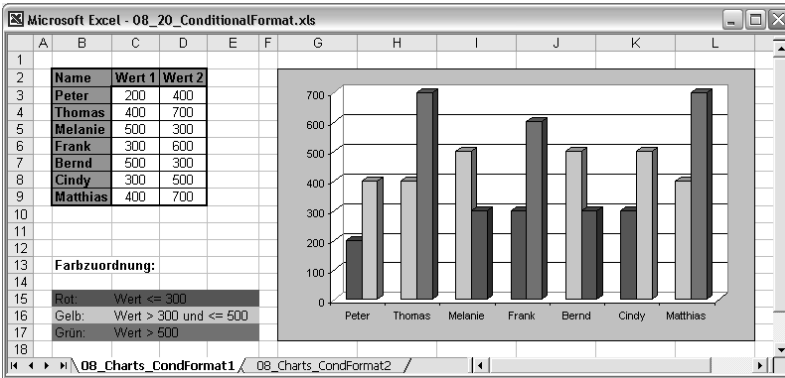


Abbildung 225: Bedingte Formatierung bezogen auf Datenpunkte

Nach Eintritt in die Ereignisprozedur der Tabelle wird zuerst festgelegt, in welchem Bereich die Prozedur ausgeführt werden soll (D3:D9). Wenn wir dies nicht tun würden, dann würde die Prozedur bei jeder Veränderung einer Zelle ausgeführt. Das wäre natürlich überflüssig und könnte zudem zu einer Fehlermeldung führen. Die Prozedur soll nur dann ausgeführt werden, wenn sich im Bereich D3:D9 etwas verändert. Direkt danach werden wir die Applikation für eine Sekunde anhalten. Unsere Tests haben gezeigt, dass dies, je nach Rechner, unerlässlich ist, wenn mit Minuswerten gearbeitet wird. Die Synchronisation zwischen dem Ereignis und dem Diagramm kann bei der Eingabe von Minuswerten für den Rechner eine zu hohe Belastung sein, was zu einem Absturz der Anwendung führen könnte.

Nach der Dimensionierung und Referenzierung werden wir mit zwei ineinander verschachtelten Schleifen arbeiten. Dies, damit wir möglichst flexibel sind in Bezug auf die Anzahl der Datenreihen und -punkte. Das heißt, das Diagramm dürfte auch mehr Datenreihen und Datenpunkte enthalten, als in unserem Beispiel, und wäre dennoch lauffähig.

Die äußere For-Schleife ist dazu da, die Datenreihen zu zählen und nacheinander anzusprechen. In der inneren For-Schleife werden die Datenpunkte der Datenreihe X gezählt und nacheinander abgearbeitet. Da die BEDINGTE FORMATIERUNG auf jeden Datenpunkt angewendet werden soll, werden wir die If...Then...Else-Entscheidung in der inneren Schleife durchführen. Die Entscheidung prüft, welche Farbe dem Datenpunkt zugewiesen werden muss.

```
'=====
' Auf CD      Buchdaten\Beispiele\08_Diagramme\
' Dateiname  08_20_ConditionalFormat.xls
' Tabelle    08_Charts_CondFormat1
' Ereignis   Tabelle1(08_Charts_CondFormat1)
'=====
```

```
Private Sub Worksheet_Change(ByVal Target As Range)
    If Application.Intersect(Target, Range("C3:D9")) Is Nothing Then
        Exit Sub
    End If

    Application.Wait Now + TimeValue("00:00:01")
```

```

' Variablen für das Diagramm
Dim chtSC As SeriesCollection
Dim chtPT As Point

' Variablen für die Zähler Datenreihen/-punkte
Dim intSC As Integer
Dim intPT As Integer
Dim varValues As Variant

Set chtSC = ActiveSheet.ChartObjects(1).Chart.SeriesCollection

' Jede Datenreihe durchlaufen
For intSC = 1 To chtSC.Count

    ' Jeden Datenpunkt durchlaufen
    For intPT = 1 To chtSC(intSC).Points.Count

        Set chtPT = chtSC(intSC).Points(intPT)
        varValues = chtSC(intSC).Values

        ' Bedingte Formatierung je nach Wert
        If varValues(intPT) <= 300 Then
            chtPT.Interior.ColorIndex = 3 ' Rot
        ElseIf varValues(intPT) > 500 Then
            chtPT.Interior.ColorIndex = 4 ' Grün
        Else
            chtPT.Interior.ColorIndex = 6 ' Gelb
        End If

    Next intPT

Next intSC

Set chtSC = Nothing
Set chtPT = Nothing
End Sub

```

Grund-
lagenAllge-
meinDatum
Zeit

tungen

Steuer
elementeBefehl
leisten

Objekt

Dia-
grammEreig-
nisseUser-
FormWeb/
Mail

Extern

Ge-
misch
tes

Specia

285 Diagramm mit bedingter Formatierung (Werte)

Dieses Rezept baut auf das Vorherige auf. Wir werden deshalb die Beschreibung etwas verkürzen. Der einzige Unterschied besteht darin, dass in diesem Beispiel nicht die Säulen selbst, sondern die zugehörigen Werte formatiert werden. Wir gehen dabei davon aus, dass die Werte über den Säulen in Ihrem Diagramm bereits eingeblendet sind. Die BEDINGTE FORMATIERUNG soll sich lediglich auf die Zahlen über den Diagrammen und nicht auf die Datenpunkte (z.B. Säulen) beziehen.

Falls in Ihrem Diagramm die Werte über den Datenreihen nicht eingeblendet sind, können Sie dies manuell erreichen, indem Sie das Diagramm aktivieren und aus dem Menü DIAGRAMM den Eintrag DIAGRAMMOPTIONEN wählen. Alternativ können Sie diesen Menüpunkt auch aus dem Kontextmenü auswählen, indem Sie zuvor mit der rechten Maustaste auf das Diagramm klicken. In dem Dialogfenster, das sich nun öffnet, aktivieren Sie die Registerkarte DATENBESCHRIFTUNG und aktivieren darin den Eintrag WERT. Schließen Sie dann das Fenster, indem Sie auf die Schaltfläche OK klicken. Die Werte zu den Datenpunkten werden nun angezeigt.

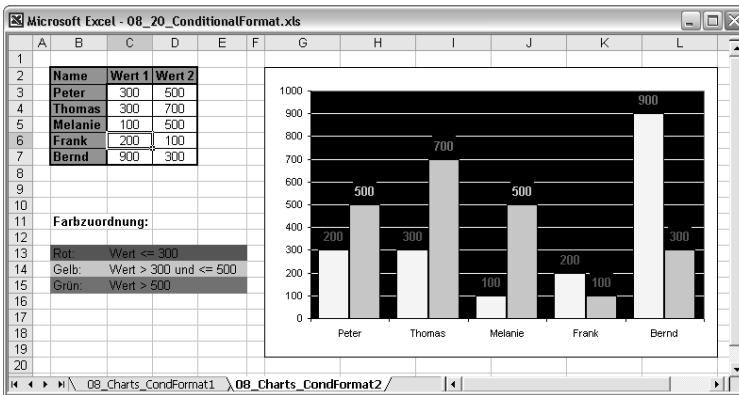


Abbildung 226: Bedingte Formatierung bezogen auf Werte über den Säulen

Die BEDINGTE FORMATIERUNG, also die If...Then...Else-Entscheidung, nimmt Bezug auf den Inhalt des DataLabels und somit auf den darin enthaltenen Text. Dieser wird durch die Methode Characters(...) angesprochen. Innerhalb der runden Klammern muss das erste Zeichen festlegen, ab welchem die Formatierung gelten soll (Start). Der zweite Wert gilt der Textlänge Length. Die Länge wurde hier auf vier Zeichen eingestellt. Das bedeutet, dass Zahlen bis zu vier Stellen formatiert werden. Sollten in Ihrem Diagramm Werte von fünf und mehr Stellen enthalten sein, müssen Sie den Wert Length erhöhen, da sonst innerhalb eines Labels nicht alle Zeichen in der angegebenen Farbe formatiert werden.

```
'=====
' Auf CD      Buchdaten\Beispiele\08_Diagramme\
' Dateiname   08_20_ConditionalFormat.xls
' Tabelle     08_Charts_CondFormat2
' Ereignis    Tabelle1(08_Charts_CondFormat2)
'=====
```

```
Private Sub Worksheet_Change(ByVal Target As Range)
    If Intersect(Target, Range("C3:D7")) Is Nothing Then
        Exit Sub
    End If

    Application.Wait Now + TimeValue("00:00:01")

    ' Variablen für das Diagramm
    Dim chtSC As SeriesCollection
    Dim chtLBL As DataLabel

    ' Variablen für die Zähler Datenreihen/-punkte
    Dim intSC As Integer
    Dim intPT As Integer
    Dim varValue As Variant

    Set chtSC = ActiveSheet.ChartObjects(1).Chart.SeriesCollection
```

```
' Jede Datenreihe durchlaufen
For intSC = 1 To chtSC.Count

    ' Jeden Datenpunkt durchlaufen
    For intPT = 1 To chtSC(intSC).Points.Count
        Set chtLBL = chtSC(intSC).Points(intPT).DataLabel
        varValue = chtSC(intSC).Values

        ' Bedingte Formatierung über Werte
        If varValue(intPT) <= 300 Then
            chtLBL.Characters(Start:=1, Length:=4) _
                .Font.ColorIndex = 3          ' Rot
        ElseIf varValue(intPT) > 500 Then
            chtLBL.Characters(Start:=1, Length:=4) _
                .Font.ColorIndex = 4          ' Grün
        Else
            chtLBL.Characters(Start:=1, Length:=4) _
                .Font.ColorIndex = 6          ' Gelb
        End If
    Next intPT
Next intSC

Set chtSC = Nothing
Set chtLBL = Nothing
End Sub
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

286 Halbtransparente Diagramme

Von den AutoFormen her kennen Sie vielleicht den Effekt von halbtransparenten Farben. Sie können dabei die Farben in einem gewünschten Prozentsatz transparent einstellen. Transparenz bedeutet, dass man durch das Objekt hindurchsehen kann. Leider gibt es diesen Effekt für Diagrammreihen nicht. Das heißt, nur mit einem Griff in die Trickkiste. Um das Ganze zu programmieren, müssen Sie erst wissen, wie dieser Effekt manuell erreicht werden kann.

Achtung

Das Verfahren eignet sich ausschließlich für 2-D-Balken- und Säulendiagramme, vorzugsweise ohne 3-D-Effekt.

Gehen wir davon aus, dass in Ihrem Tabellenblatt ein Balken- oder Säulendiagramm vorhanden ist. Dann verfahren Sie wie folgt:

1. Fügen Sie in das Tabellenblatt pro Datenreihe ein Rechteck aus der Symbolleiste ZEICHNEN ein (diese Rechtecke können am Schluss wieder entfernt werden). Das Rechteck sollte möglichst groß sein, da sonst am Ende die Größe der Datenreihen verfälscht wird.
2. Klicken Sie mit der rechten Maustaste auf das Rechteck und wählen Sie aus dem Kontextmenü den Eintrag AUTOFORM FORMATIEREN.
3. Aktivieren Sie die Registerkarte FARBEN UND LINIEN. Im ersten Drittel dieses Dialogfensters können Sie nun die Farbe und den gewünschten Transparenzgrad einstellen.

4. Vorzugsweise sollten Sie auch die Linien entfernen, denn es könnte sonst zu einem unschönen Effekt im Diagramm führen. Diese Einstellung finden Sie im Dialogfenster im mittleren Teil, auf derselben Registerkarte. Nachdem Sie diese Einstellungen vorgenommen haben, können Sie das Dialogfenster schließen, indem Sie auf die Schaltfläche OK klicken.
5. Klicken Sie nun mit gedrückter -Taste auf den Menüpunkt BEARBEITEN | BILD KOPIEREN. Übernehmen Sie die Einstellungen des Dialogfensters, das sich nun öffnet, indem Sie auf die Schaltfläche OK klicken.
6. Aktivieren Sie nun im Diagramm eine der Datenreihen und drücken Sie dann die Tastenkombination  +  für Einfügen. Damit ist die Datenreihe nun transparent dargestellt.
7. Verfahren Sie für jede Datenreihe gleich.



Abbildung 227: Halbtransparente Farbe einstellen

Nachdem Sie nun wissen, wie umständlich es ist, einer Datenreihe eine halbtransparente Farbe zuzuweisen, wollen wir uns ansehen, wie das Ganze per VBA umgesetzt werden kann.

Da Sie VBA-Prozeduren nicht rückgängig machen können, außer Sie erstellen eigens dazu eine Prozedur, werden wir in unserem Code direkt nach der Deklaration und Referenzierung zuerst eine Sicherheitskopie unseres Diagramms erstellen.

Danach wird in einer `For...Next`-Schleife gezählt, wie viele Datenreihen das Diagramm aufweist, um entsprechend viele transparente Rechtecke zu erstellen. Die Rechtecke werden mit der Methode `AddShape` hinzugefügt. Innerhalb der runden Klammer nach der Anweisung `AddShape` wird mittels der ersten beiden Zahlen die Position des Rechteckes auf dem Tabellenblatt bestimmt. Die erste Zahl bedeutet den Abstand von links und die Zweite den Abstand von oben. Beide Werte spielen in unserem Beispiel keine Rolle, da die Rechtecke ohnehin wieder entfernt werden. Die letzten beiden Werte bedeuten die Breite und die Höhe. Wie bereits erwähnt, sollten die Rechtecke möglichst groß sein.

Vor dem Erstellen des Rechtecks wird an die Variable `intColor` der Original-Farbwert der Datenreihe übergeben. Dies ist erforderlich, damit die Farbe nach dem Erstellen des Rechteckes an dieses übergeben werden kann. Damit erreichen wir, dass das zweite Diagramm, also jenes, welches halbtransparente Datenreihen aufweisen soll, dieselben Farbwerte erhält. Direkt nach dem Erstellen des Rechteckes wird dieser Farbwert übergeben. Danach wird die Transparenz `Transparency` eingestellt. In unserem Beispiel ist es der Wert 0,7, was 70% entspricht.

Nach der Einstellung der Transparenz wird der Rahmen um das Rechteck entfernt. Danach wird das Rechteck als Bild kopiert (`CopyPicture`) und kann schließlich gelöscht (`Delete`) werden, da sich die gewünschten Daten nun in der Zwischenablage befinden. Der Vorgang für das Rechteck ist damit beendet.

Der Inhalt der Zwischenablage, also die halbtransparente Farbe, kann in die Datenreihe eingefügt werden (`Paste`). Vor Abschluss der Schleife wird ein Rahmen um die Datenreihen gezogen. Damit wäre das halbtransparente Diagramm fertig gestellt. Es ist, bis auf die Halbtransparenz, mit dem Original identisch und kann jetzt nach Belieben ausgerichtet werden.

```
'=====
' Auf CD      Buchdaten\Beispiele\08_Diagramme\
' Dateiname  08_21_Semitransparent.xls
' Modul      mdl_01_Semitransparent
'=====
```

```
Sub Semitransparent()
    Dim shpS As Shapes
    Dim cht As Chart
    Dim chtSC As SeriesCollection
    Dim i As Integer
    Dim dblColor As Double

    Set shpS = ActiveSheet.Shapes
    Set cht = ActiveSheet.ChartObjects(1).Chart
    Set chtSC = cht.SeriesCollection

    Application.ScreenUpdating = False

    ' Sicherheitskopie vom Original-Diagramm erstellen
    cht.ChartArea.Copy
    ActiveSheet.Paste

    ' Pro Datenreihe ein halbtransparentes
    ' Rechteck erstellen
    For i = 1 To ActiveChart.SeriesCollection.Count
        dblColor = chtSC(i).Fill.ForeColor.RGB
        With shpS.AddShape(msoShapeRectangle, 100, 0, 500, 500)
            .Fill.ForeColor.RGB = dblColor
            .Fill.Transparency = 0.7
            .Line.Visible = msoFalse
            .CopyPicture Appearance:=xlScreen, Format:=xlPicture
            .Delete
        End With
    End For
```

Grund-
lagenAllge-
meinDatum
Zeit

tungen

Steuer
elemenBefehl
leisten

Objekt

Dia-
grammEreig-
nisseUser-
FormWeb/
Mail

Extern

Ge-
misch
tes

Specia

```

' Die halbtransparente Farbe den Datenreihen zuweisen
' und eine Linie um die Datenreihen einfügen
With chtSC(i)
.Paste
.Border.Weight = xlContinuous
End With
Next i

```

```

Set shpS = Nothing
Set cht = Nothing
Set chtSC = Nothing

```

```

Application.ScreenUpdating = True
End Sub

```

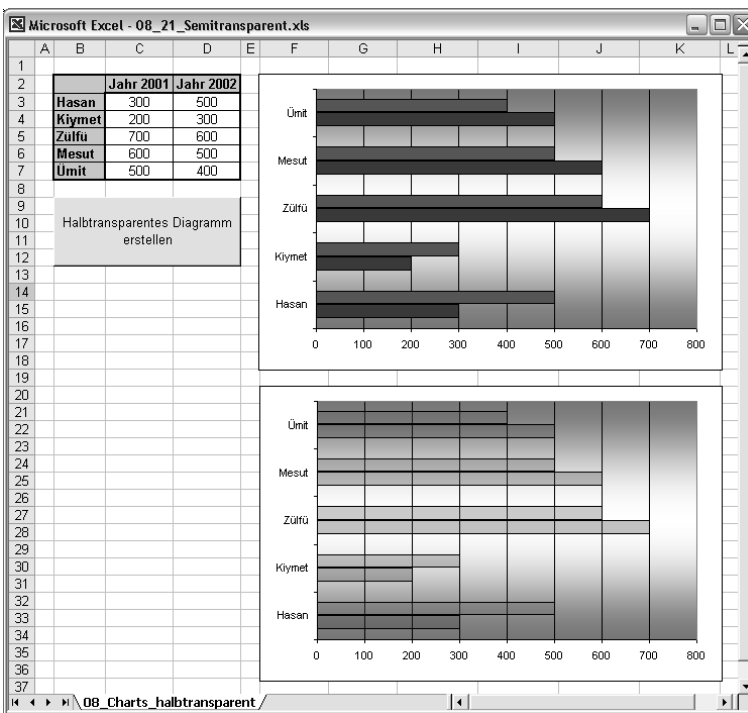


Abbildung 228: Original und halbtransparentes Diagramm

287 Verbund-Diagramm

Ein Verbund-Diagramm besteht aus mindestens zwei Datenreihen, welche je einen anderen Diagrammtypen aufweisen. Das Diagramm wird wie gewohnt erstellt, beispielsweise als Säulendiagramm. Gewisse Daten lassen sich jedoch besser vergleichen, wenn die eine Datenreihe eine Säule ist und die andere z.B. eine Linie oder eine Fläche ist.

Wie können nun zwei oder mehr Diagrammtypen in einem Diagramm vereint werden? Der Weg ist recht einfach. Gehen wir von einem bestehenden Säulendiagramm mit zwei Datenreihen aus. Die eine Datenreihe soll von einer Säule in eine Fläche umgewandelt werden. Klicken Sie mit der

rechten Maustaste auf die Datenreihe, der Sie einen anderen Typen zuweisen möchten. Wählen Sie aus dem Kontextmenü den Eintrag **DIAGRAMMTYP** oder klicken Sie alternativ, nach dem Aktivieren der zu verändernden Datenreihe, auf den Menüpunkt **DIAGRAMM | DIAGRAMMTYP**. In dem Dialogfenster, das sich nun öffnet, wählen Sie den gewünschten Typen aus.

Achtung

Beachten Sie bei der Auswahl des Diagrammtyps, dass keine 3-D-Diagramme als Verbund verwendet werden können. Wählen Sie stattdessen ein 2-D-Diagramm aus.

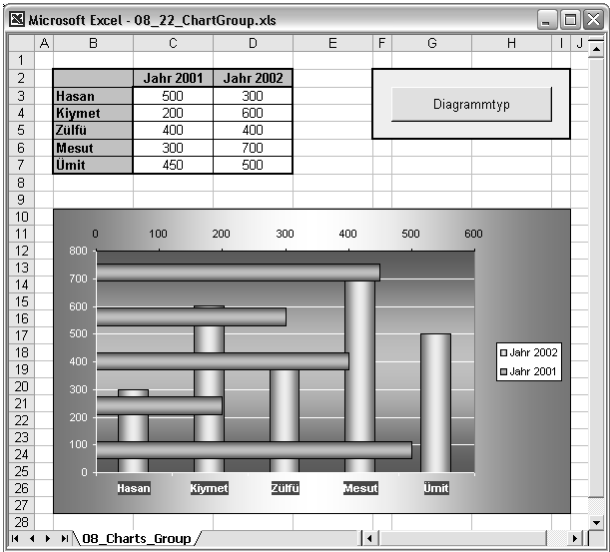


Abbildung 229: Verbunddiagramm

Unsere Prozedur ist diesmal so aufgebaut, dass nach dem Start nacheinander zwei Eingabemasken erscheinen. Im ersten Eingabefenster kann die zu verändernde Datenreihe ausgewählt werden und im zweiten Eingabefenster kann der Diagrammtyp ausgewählt werden.

Datenreihe

1 für die erste Datenreihe
2 für die zweite Datenreihe

1

OK Abbrechen

Diagrammtyp

1 für Säule
2 für Balken
3 für Linie
4 für Fläche

1

OK Abbrechen

Abbildung 230: Eingabefenster zur Auswahl der Datenreihe und des Diagrammtyps

Die erste Hälfte der Prozedur widmet sich lediglich den Eingabefenstern. Es werden verschiedene If...Then...Else-Entscheidungen verwendet, um zu prüfen, ob die Eingaben korrekt sind. Bei einer Falscheingabe wird die Prozedur abgebrochen. Ansonsten wird der eingegebene Wert an die Variable `bytRow` übergeben.

Grund-lagen

Allge-mein

Datu Zeit

tungen

Steuer elemen

Befehl leisten

Objekt

Dia-gramm

Ereig-nisse

User-Form

Web/ Mail

Extern

Ge-misch tes

Specia

In der zweiten Hälfte der Prozedur werden die Werte, die im Eingabefenster eingetippt wurden, dem Diagramm zugewiesen. Dies geschieht ebenfalls in einer If...Then...Else-Entscheidung.

```
'=====
' Auf CD      Buchdaten\Beispiele\08_Diagramme\
' Dateiname   08_22_ChartGroup.xls
' Modul       mdl_01_Group
'=====

Sub Group()
    Dim cht As Chart
    Dim chtSC As SeriesCollection
    Dim bytRow As Byte
    Dim bytType As Byte

    Set cht = ActiveSheet.ChartObjects(1).Chart
    Set chtSC = cht.SeriesCollection

    ' Eingabefenster zur Auswahl der Datenreihe
    bytRow = Application.InputBox _
        ("1 für die erste Datenreihe" & vbCrLf & _
        "2 für die zweite Datenreihe", _
        Default:="1", Title:="Datenreihe", Type:=1)

    ' Abbruch bei Falscheingabe
    If bytRow = False Then
        Exit Sub
    ElseIf bytRow < 1 Or bytRow > 2 Then
        MsgBox "Falsche Eingabe"
    End If

    ' Eingabefenster zur Auswahl des Diagrammtyps
    bytType = Application.InputBox _
        ("1 für Säule" & vbCrLf & _
        "2 für Balken" & vbCrLf & _
        "3 für Linie" & vbCrLf & _
        "4 für Fläche", _
        Default:="1", Title:="Diagrammtyp", Type:=1)

    ' Abbruch bei Falscheingabe
    If bytType = False Then
        Exit Sub
    ElseIf bytType < 1 Or bytType > 4 Then
        MsgBox "Falsche Eingabe"
    End If

    ' ZUWEISUNG DES DIAGRAMMTYPS
    ' Erste Datenreihe
    If bytRow = 1 Then
        If bytType = 1 Then
            chtSC(1).ChartType = xlColumnClustered    ' Säule
        ElseIf bytType = 2 Then
            chtSC(1).ChartType = xlBarClustered        ' Balken
```

```

    ElseIf bytType = 3 Then
        chtSC(1).ChartType = xlLineMarkers      ' Linie
    ElseIf bytType = 4 Then
        chtSC(1).ChartType = xlAreaStacked      ' Fläche
    End If

    ' Zweite Datenreihe
    ElseIf bytRow = 2 Then
        If bytType = 1 Then
            chtSC(2).ChartType = xlColumnClustered  ' Säule
        ElseIf bytType = 2 Then
            chtSC(2).ChartType = xlBarClustered    ' Balken
        ElseIf bytType = 3 Then
            chtSC(2).ChartType = xlLineMarkers      ' Linie
        ElseIf bytType = 4 Then
            chtSC(2).ChartType = xlAreaStacked      ' Fläche
        End If
    End If

    Set cht = Nothing
    Set chtSC = Nothing
End Sub
```

Grund-lagen

Allge-mein

Datu Zeit

tungen

Steuer elemen

Befehl leisten

Objekt

Dia-gramm

288 Blasendiagramm-Blume

Um mit Diagrammen zu arbeiten, ist es von Vorteil, wenn man ein bisschen eine grafische Ader hat, denn Diagramme wirken erst richtig ansprechend, wenn sie schön gestaltet sind. Mit Diagrammen kann man in der Tat sehr kreativ sein.

Zum Abschluss dieses Kapitels wollen wir Ihnen zeigen, dass man mit Diagrammen sogar richtige Kunstwerke erstellen kann. Dieses Beispiel ist zum reinen Spaß gedacht. Es soll den oftmals trockenen VBA-Alltag etwas auflockern. Das Diagramm basiert auf einer Tabelle mit vorgegebenen Werten. Es handelt sich dabei um ein Blasendiagramm, das komplett per VBA erstellt wird. Das Ergebnis können Sie der Abbildung 231 entnehmen. In Farbe sieht die Blume natürlich besser aus, als hier auf dem schwarz-weißen Druck des Buches. Am besten sehen Sie sich das Beispiel an. Es befindet sich auf der beiliegenden CD.

Um ein solches Diagramm zu erstellen, braucht es etwas Geduld. Die Werte und die Größe des Diagramms müssen aufeinander abgestimmt werden. Wenn diese Daten einmal vorhanden sind und Sie das Ganze per VBA automatisieren wollen, nehmen Sie sich am besten den Makro-Rekorder zu Hilfe. Zeichnen Sie jeden einzelnen Schritt auf. Der erste Blick auf den VBA-Code mag etwas erschreckend sein, denn was sich Ihnen im VBA-Editor präsentiert, ist weit entfernt von ordentlicher Programmierung. Sie können es jedoch als eine hervorragende Übung im Umgang mit Diagramm-Objekten, -Eigenschaften und -Methoden ansehen.

Die Beschreibung der einzelnen Fragmente ist direkt im Code untergebracht. Um es jedoch an Erläuterungen nicht mangeln zu lassen, hier ein kurzer Überblick: Nach der Variablendeklaration wird die Bildschirmaktualisierung `ScreenUpdating` deaktiviert, damit einerseits der Aufbau des Diagramms nicht in Einzelschritten sichtbar ist und andererseits die Prozedur schneller abläuft. Da es sich um einen anwendungsbezogenen Befehl handelt, muss dieser am Ende der Prozedur wieder aktiviert, das heißt auf `True` gesetzt werden, denn sonst würde die Aktualisierung generell für Excel deaktiviert bleiben.

Ereig-nisse

User-Form

Web/ Mail

Extern

Ge-misch tes

Specia

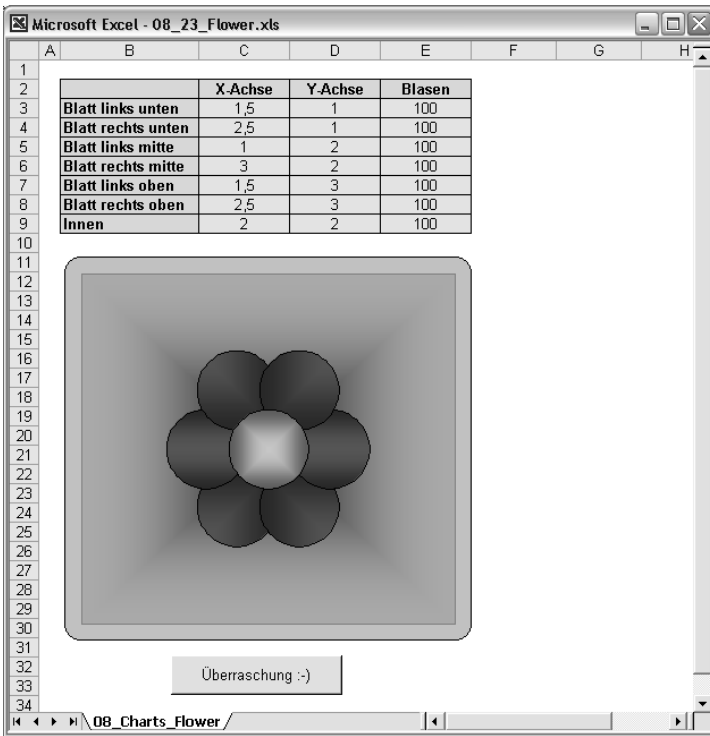


Abbildung 231: Kunstvolles Blasendiagramm

In einem zweiten Schritt werden eventuell vorhandene Diagramme in der 08_CHARTS_FLOWER entfernt. Dies vor allem für den Fall, dass die Prozedur mehrmals ausgeführt wird. So wird das enthaltene Diagramm jeweils zuerst entfernt, denn sonst würde das Ansprechen per Index durcheinander geraten. Direkt danach wird der Bereich mit den Daten selektiert.

In einem dritten Schritt wird das Diagramm vom Typ Blasen erstellt beziehungsweise zum Tabellenblatt hinzugefügt (`Charts.Add`). Dabei wird der Typ `xlBubble` angegeben und die Quelle festgelegt (`SetSourceData`). Damit das Diagramm eingebettet und nicht auf einem eigenen Diagrammblatt erstellt wird, muss angegeben werden, wo das Diagramm eingebettet werden soll. Dazu verwenden wir die Konstante `xlLocationAsObject` und den Zielort, also die 08_CHARTS_FLOWER.

Nun, da das Diagramm besteht, können wir darauf referenzieren. Es wird einerseits das Diagramm selbst sowie die Datenreihe, welche wir später vermehrt wieder verwenden werden, referenziert.

Nach der Referenzierung können wir damit beginnen, das Diagramm zu bearbeiten. Die horizontalen und vertikalen Gitternetzlinien werden ausgeblendet. Den Achsen werden bestimmte Werte zugewiesen, damit die Blume in der Mitte der Diagrammfläche erscheint. Danach werden die Achsen gelöscht. Somit erhalten wir ein sauberes Bild ohne irgendwelche Linien. Die Legende wird ebenfalls ausgeblendet.

Nachdem das getan ist, können wir damit beginnen, die Datenpunkte, also die Blume selbst zu formatieren. Damit sie schön ansprechend aussieht, weisen wir den einzelnen Blütenblättern verschiedene Fülleffekte bzw. Farbverläufe zu. Auch die Blütenmitte erhält einen Farbverlauf, ausgehend

von der Mitte. Der Mitte der Blüte bzw. deren Objekt weisen wir zudem eine gelbe Farbe zu. Den Blütenblättern weisen wir unter Zuhilfenahme einer `For...Next`-Schleife eine rote Füllfarbe zu.

Nachdem das Diagramm in Form einer Blume formatiert ist, wird das Diagramm mittels `IncrementTop` und `IncrementLeft` ausgerichtet. Mittels `ScaleWidth` und `ScaleHeight` wird die Breite und Höhe des Diagramms festgelegt. Damit erreichen wir, dass die Blume schön rund wird und keine Lücken zwischen den Blättern und der Blütenmitte aufweist.

Die Zeichenfläche soll ebenfalls formatiert werden. Der Hintergrund erhält einen Farbverlauf, der von der Mitte ausgeht. Als Farbe wurde Cyan ausgewählt. Zu guter Letzt wird auch noch der Diagrammfläche selbst eine violette Farbe zugewiesen und die Ecken werden abgerundet (`RoundedCorner = True`).

Wie Sie sehen, steckt schon ein ganzes Stück Arbeit dahinter, den Code, welcher einmal durch die Aufzeichnung mit dem Makro-Rekorder entstanden ist, in ein übersichtliches und leserliches Programm zu verwandeln.

```
'=====
' Auf CD      Buchdaten\Beispiele\08_Diagramme\
' Dateiname  08_23_Flower.xls
' Modul      mdl_01_Flower
'=====
```

```
Sub Flower()
    Dim ws As Worksheet
    Dim chtObj As ChartObject
    Dim cht As Chart
    Dim chtSC As SeriesCollection
    Dim i As Integer

    Application.ScreenUpdating = False

    ' Das Tabellenblatt referenzieren
    Set ws = Worksheets("08_Charts_Flower")

    ' Evtl. vorhandene Diagramme löschen
    For Each chtObj In ws.ChartObjects
        chtObj.Delete
    Next chtObj

    ' Daten fürs Diagramm selektieren
    Range("B3:D9").Select

    ' Blasen-Diagramm erstellen
    With Charts.Add
        .ChartType = xlBubble
        .SetSourceData Source:=ws.Range("B3:D9")
        .Location Where:=xlLocationAsObject, _
            Name:="08_Charts_Flower"
    End With

    ' Diagramm-Objekte Referenzieren
    Set cht = ws.ChartObjects(1).Chart
```

Grund-
lagenAllge-
meinDatu
Zeit

tungen

Steuer
elemenBefehl
leisten

Objekt

Dia-
grammEreig-
nisseUser-
FormWeb/
Mail

Extern

Ge-
misch
tes

Specia

```

Set chtSC = cht.SeriesCollection

' DIAGRAMM BEARBEITEN
With cht
    ' Vertikale Gitternetzlinien ausblenden
    ' Achsen skalieren und löschen
    With .Axes(xlCategory)
        .HasMajorGridlines = False
        .HasMinorGridlines = False
        .MinimumScale = -1
        .MaximumScale = 5
        .Delete
    End With

    ' Horizontale Gitternetzlinien ausblenden
    ' Achsen skalieren und löschen
    With .Axes(xlValue)
        .HasMajorGridlines = False
        .HasMinorGridlines = False
        .MinimumScale = -1
        .MaximumScale = 5
        .Delete
    End With

    ' Legende ausblenden
    .HasLegend = False
End With

' DATENPUNKTE BEARBEITEN
' Farbverlauf zuweisen
With chtSC(1)
    ' Blütenblatt links unten
    .Points(1).Fill.OneColorGradient _
        Style:=msoGradientFromCorner, _
        Variant:=3, _
        Degree:=0.2313725490196

    ' Blütenblatt rechts unten
    .Points(2).Fill.OneColorGradient _
        Style:=msoGradientFromCorner, _
        Variant:=4, _
        Degree:=0.231372549019608

    ' Blütenblatt links mitte
    .Points(3).Fill.OneColorGradient _
        Style:=msoGradientHorizontal, _
        Variant:=4, _
        Degree:=0.231372549019608

    ' Blütenblatt rechts mitte
    .Points(4).Fill.OneColorGradient _
        Style:=msoGradientHorizontal, _
        Variant:=4, _

```

Degree:=0.231372549019608	Grund-lagen
' Blütenblatt links oben	
.Points(5).Fill.OneColorGradient _	Allge-mein
Style:=msoGradientFromCorner, _	
Variant:=1, _	
Degree:=0.231372549019608	Datu Zeit
' Blütenblatt rechts oben	
.Points(6).Fill.OneColorGradient _	tungen
Style:=msoGradientFromCorner, _	
Variant:=2, _	
Degree:=0.231372549019608	Steuer elemen
' Blütenmitte (gelb)	
With .Points(7).Fill	Befehl leisten
.OneColorGradient _	
Style:=msoGradientFromCenter, _	
Variant:=1, _	
Degree:=0.231372549019608	Objekt
.ForeColor.SchemeColor = 6	
End With	Dia-gramm
End With	
' Blütenblätter rot einfärben	Ereig-nisse
For i = 1 To 6	
chtSC(1).Points(i).Fill.ForeColor.SchemeColor = 3	User-Form
Next i	
' DIAGRAMM AUSRICHTEN UND GRÖSSE FESTLEGEN	Web/Mail
With ws.Shapes(2)	
' Ausrichtung links und oben	Extern
.IncrementLeft -230	
.IncrementTop -15	
' Breite und Höhe festlegen	Ge-misch tes
.ScaleWidth 0.58, msoFalse, msoScaleFromTopLeft	
.ScaleHeight 0.75, msoFalse, msoScaleFromTopLeft	Specia
End With	
' ZEICHENFLÄCHE FORMATIEREN (Verlauf cyan)	
With cht.PlotArea	
.Fill.OneColorGradient _	
Style:=msoGradientFromCenter, _	
Variant:=2, _	
Degree:=0	
.Fill.ForeColor.SchemeColor = 8	
End With	
' DIAGRAMMFLÄCHE FORMATIEREN (violett)	
cht.ChartArea.Interior.ColorIndex = 39	

```
' Ecken der Diagrammfläche abrunden
ws.DrawingObjects(2).RoundedCorners = True

Set ws = Nothing
Set cht = Nothing
Set chtSC = Nothing

Application.ScreenUpdating = True
End Sub
```

Die Idee, ein solches Kunstwerk-Diagramm zu erstellen, stammt von einem unserer MVP-Kollegen, John Walkenbach, welcher uns erlaubt hat, die Idee für dieses Buch zu verwenden. Er hat in seinem Buch »Excel Charts« ähnliche Diagramme manuell dargestellt. Ein Blick auf seine Webseite (in englischer Sprache) ist auf jeden Fall lohnenswert: <http://www.j-walk.com/ss>.

Damit wären wir also am Ende der Kategorie »Diagramme« angelangt. Das Thema ist schier unerschöpflich und es ließe sich eigens dazu ein ganzes Buch schreiben. Wir haben uns jedoch bemüht, wenigstens die wichtigsten Themen abzudecken.

Ereignisse

289 Allgemeine Prozeduren

Bevor wir auf die Ereignisse eingehen, geben wir Ihnen zum besseren Verständnis eine kurze Zusammenfassung über die Entstehung und Ausführung von herkömmlichen Prozeduren.

Das erste Makro erstellen die meisten Anwender, indem Sie über den Menüpunkt EXTRAS | MAKRO | AUFZEICHNEN den Makro-Rekorder starten und dann einige Aktionen auf dem Tabellenblatt ausführen. In dem Dialogfenster MAKRO AUFZEICHNEN laut Abbildung 232 geben Sie vor, in welcher Arbeitsmappe das Makro gespeichert werden soll.

Tipp

Wenn sich noch keine *PERSONL.xls* in Ihrem XLSTART-Ordner befindet, wählen Sie bei der Makroaufzeichnung als Speicherort PERSÖNLICHE MAKROARBEITSMAPPE aus. Daraufhin wird *PERSONL.xls* angelegt und muss nur noch gespeichert werden. Diese Datei wird dann künftig bei jedem Start von Excel im Hintergrund mitgeladen. Andere geöffnete Excel-Arbeitsmappen können auf die in dieser Excel-Datei gespeicherten Makros zugreifen.

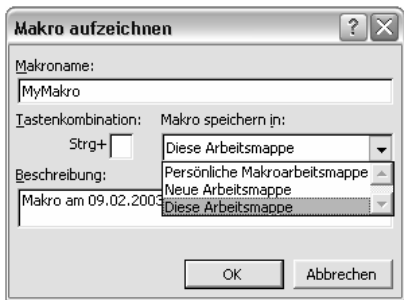


Abbildung 232: Speicherort des Makros vorgeben

Excel erstellt daraufhin im entsprechenden VBAPROJECT der ausgewählten Arbeitsmappe ein neues Standardmodul. Die durchgeführten Aktionen, zum Beispiel: Daten eingeben, Zellen formatieren, Einrichten der Kopf- und Fußzeilen etc., werden von Excel »mitgeschrieben« und entweder in dem neu angelegten oder in einem bereits vorhandenen Standardmodul gespeichert. Die Standardmodule werden standardmäßig mit MODUL(N) fortlaufend benannt, zum Beispiel MODUL1, MODUL2 etc.

Tipp

Im Eigenschaften-Fenster des Moduls können Sie zur besseren Orientierung den Modulnamen umbenennen. Der Modulname muss mit einem Buchstaben beginnen und darf keine Leer- oder Sonderzeichen enthalten. Achten Sie darauf, dass Modulnamen nicht gleich einem Makronamen lauten, da es sonst zu Namenskonflikten kommen kann (Modulname: MYMAKRO Sub MyMakro).

Grundlagen
Allgemein
Datum und Zeit
Diagramme
Formeln
Objekte
Diagramme
Ereignisse
User-Form
Web/Mail
Extern
Gemischtes
Spezial

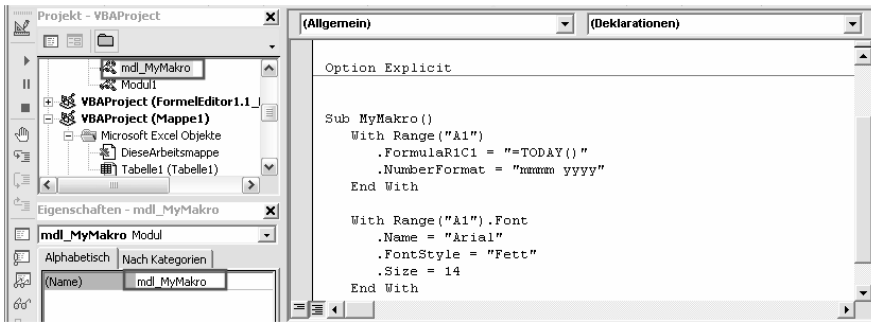


Abbildung 233: Module eines VBA-Projekts

Aufgezeichnete bzw. selbst erstellte Makros können, wenn sie mit der Bezeichnung `Sub` oder `Public Sub` beginnen, über verschiedene Möglichkeiten aufgerufen werden. Einen manuellen Aufruf erwirken Sie zum Beispiel über den Menüpunkt **EXTRAS | MAKRO | MAKROS**. Wählen Sie aus den angezeigten Prozeduren das gewünschte Makro aus und klicken Sie auf **AUSFÜHREN**, um das Makro zu starten.

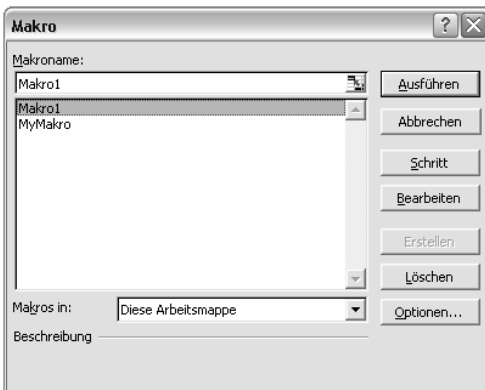


Abbildung 234: Makro manuell starten

Befinden Sie sich in der VBA-Umgebung, genügt es, wenn Sie den Schreibcursor innerhalb einer Prozedur in einer Zeile postieren. Mit dem Betätigen der Taste **[F5]** wird das gesamte Makro durchlaufen. Durch einzelnes Drücken der Taste **[F8]** wird es in Einzelschritten ausgeführt.

Dies sind nur zwei der möglichen Varianten, ein Makro durch den Anwender manuell zu starten.

290 Ereignis-Prozeduren

Im Gegensatz zu den oben beschriebenen, manuell zu startenden Makros, kann Excel auf bestimmte Ereignisse reagieren und dadurch automatisch hinterlegte Befehle zu dem jeweiligen Ereignis ausführen.

VBA ist eine ereignisorientierte Programmiersprache. Ereignisse kann man mit den Aktionen des Anwenders vergleichen. Öffnen Sie zum Beispiel eine Excel-Datei, so ist das erste mögliche Ereignis bereits eingetreten. Excel interpretiert das Öffnen einer Datei als `Workbook_Open`-Ereignis. Das Schließen einer Datei wird von Excel über das `Workbook_Close`-Ereignis kontrolliert und stellt das

letztmögliche Ereignis einer Excel-Arbeitsmappe dar. Dazwischen gibt es eine schier unzählige Anzahl von Ereignissen, von denen wir Ihnen in diesem Kapitel die wichtigsten anhand von Beispielen erklären werden.

Excel unterscheidet bei den Ereignissen, ob es sich um eine Änderung handelt, welche die Arbeitsmappe *Workbook* betrifft, oder ob eine Aktion in oder an einem Tabellenblatt *Worksheet* aufgetreten ist. Aus diesem Grund werden die entsprechenden Prozeduren in den dafür vorgesehenen Containern des VBAPROJEKTS hinterlegt. In Abbildung 235 sehen Sie, dass allgemeine Makros in Standardmodulen gespeichert werden. Ereignisse rund um die Arbeitsmappe werden im Codeblatt des Klassenmoduls *DIESEARBEITSMAPPE* hinterlegt. Für die Tabellenblatt-Ereignisse ist das Codeblatt des betreffenden Tabellenblattes, zum Beispiel *TABELLE1*, vorgesehen.

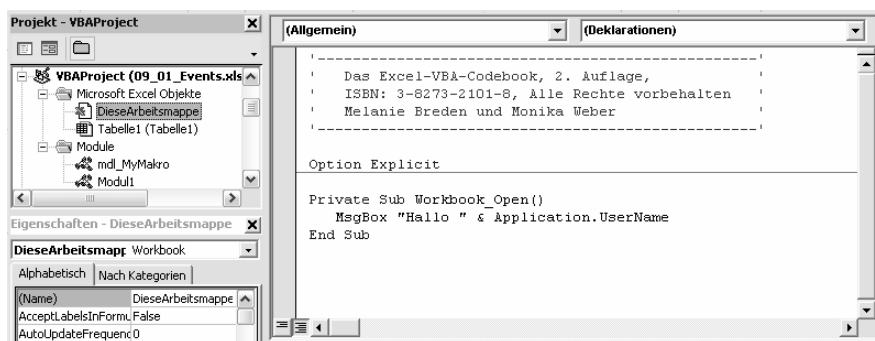


Abbildung 235: Speicherort für Workbook-Ereignisse

Ereignis-Prozeduren beginnen grundsätzlich mit den Wörtern `Private Sub`. Aus diesem Grund sind sie nicht wie öffentliche Prozeduren unter **EXTRAS | MAKRO | MAKROS** in der Liste der vorhandenen Makros (siehe Abbildung 235) aufgeführt und können nicht manuell aufgerufen werden.

In einer Arbeitsmappe können Sie jeweils nur ein gleichnamiges Ereignis einsetzen. Wollen Sie mehrere Befehle bei einem auftretenden Ereignis ausführen lassen, müssen diese in »einer« Ereignis-Prozedur vereint und nacheinander abgearbeitet werden. Um Ihnen zu einem Ereignis verschiedene Beispiele aufzeigen zu können, verwenden wir unterschiedliche Arbeitsmappen. Dem Hinweistext oberhalb einer Prozedur entnehmen Sie jeweils den Dateinamen und das Modul, in welchem der Code zu finden ist.

Nachfolgend werden die *Workbook*- und *Worksheet*-Ereignisse getrennt erläutert.

291 Workbook-Ereignisse

Wie im vorherigen Abschnitt bereits erwähnt, werden die *Workbook*-Ereignisse im Container *DIESEARBEITSMAPPE* der *VBAPROJECTE* gespeichert. Befinden Sie sich bereits im VBA-Editor, genügt ein Doppelklick auf *DIESEARBEITSMAPPE*, um in der rechten Fensterhälfte das Codeblatt von *DIESEARBEITSMAPPE* anzuzeigen. Wenn Sie von der Excel-Anwendung direkt zum Klassenmodul *DIESEARBEITSMAPPE* mit dem dazugehörigen Codebereich wechseln wollen, führen Sie folgende Schritte aus:

Grundlagen

Allgemein

Datum
Zeit

Formeln

Steuerelemente

Befehle
leisten

Objekt

Diagramm

Ereignisse

User-Form

Web/
Mail

Extern

Gemischtes

Spezial

Tipp

- ▶ Klicken Sie in der Menüleiste von Excel mit der rechten Maustaste auf das grüne EXCEL-SYMBOL links neben dem Menüpunkt DATEI.
- ▶ Wählen Sie dann in dem aufgeklappten Kontextmenü den untersten Eintrag CODE ANZEIGEN.
- ▶ Der VBA-Editor öffnet sich daraufhin bzw. bei schon geöffnetem Editor wird die VBA-Umgebung zum aktiven Fenster und zeigt in der rechten Fensterhälfte den Codebereich zu DIESEARBEITSMAPPE an.

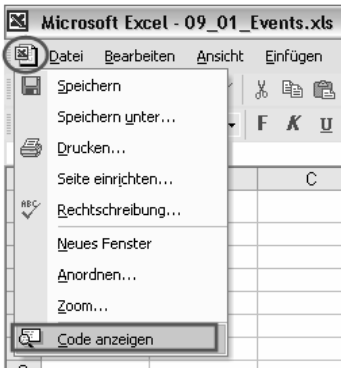


Abbildung 236: Schneller Weg zu »DieseArbeitsmappe«

Um ein neues Ereignis in Ihre Arbeitsmappe zu integrieren, müssen Sie nicht die komplette Syntax der Aufrufzeile auswendig lernen oder in der Online-Hilfe bzw. sonstiger Lektüre nachschlagen.

Oberhalb des Codefensters von DIESEARBEITSMAPPE befinden sich zwei Listenfelder. Klicken Sie zunächst auf den DropDown-Pfeil des linken Listenfeldes OBJEKT und wählen Sie den Eintrag WORKBOOK. Excel erstellt daraufhin die Codeschablone des ersten möglichen Ereignisses: Workbook_Open.

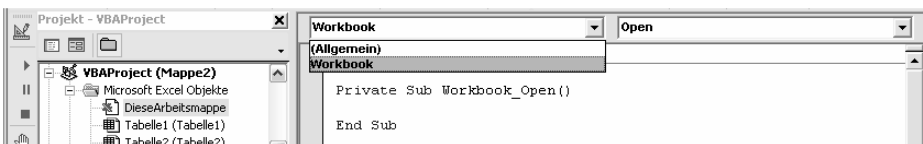


Abbildung 237: Codegerüst einer Ereignis-Prozedur

Klicken Sie jetzt auf den DropDown-Pfeil des rechten Listenfeldes PROZEDUR, erhalten Sie eine Liste aller verfügbarer Ereignis-Prozeduren für das Tabellenblatt. Durch Auswahl eines Eintrages aus dieser Liste wird das entsprechende Codegerüst im Codeblatt erzeugt.

Zwischen der Aufruf- und Schlusszeile können Sie die Befehlszeilen einfügen, die bei Eintritt des Ereignisses ausgeführt werden sollen. In unserem ersten Beispiel wird der hinterlegte Benutzername ausgelesen und in einem Meldungsfenster angezeigt. Wenn Sie die Beispieldatei öffnen und die Ausführung von Makros erlauben, wird das hinterlegte Workbook_Open-Ereignis ausgeführt. Sie werden über ein Meldungsfenster mit dem in den Excel-Optionen hinterlegtem Benutzernamen begrüßt.

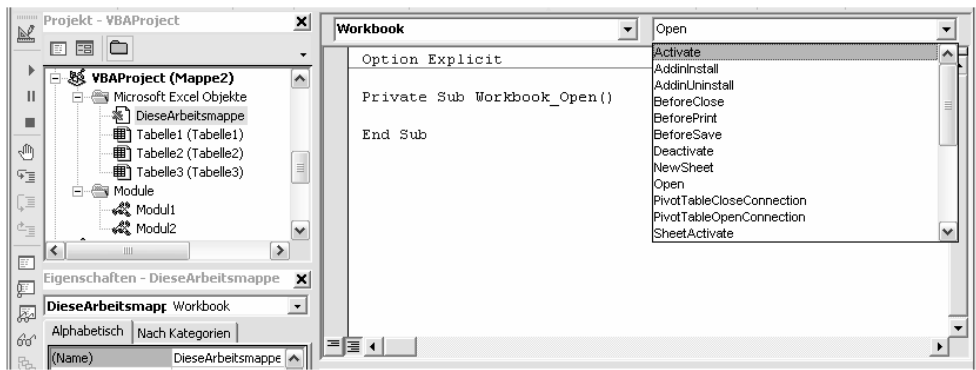


Abbildung 238: Listenfeld der vorhandenen Workbook-Ereignis-Prozeduren

```
' =====  
' Auf CD      Buchdaten\Beispiele\09_Ereignisse\  
' Dateiname  09_01_ Events.xls  
' Ereignis   DieseArbeitsmappe  
' =====  
  
Private Sub Workbook_Open()  
    MsgBox "Hallo " & Application.UserName  
End Sub
```

Hinweis

Diagrammblätter weisen eigene Ereignis-Prozeduren auf, auf die wir an dieser Stelle nicht näher eingehen.

Nachfolgend finden Sie eine Zusammenfassung aller möglichen Ereignisse, die Sie für die Arbeitsmappe verwenden können. In der zweiten Spalte der Tabelle erfahren Sie, wann der Aufruf erfolgt und welche Besonderheiten es zu den einzelnen Ereignissen gibt. Die Liste enthält alle möglichen Ereignisse, welche in der Excel-Version »Excel 2002« zur Verfügung stehen. In anderen Excel-Versionen kann es Abweichungen geben.

Ereignis	Aufruf / Beschreibung
Workbook_Activate	Wenn eine Arbeitsmappe aktiviert wird.
Workbook_AddinInstall	Wenn ein Add-In eingebunden wird. ▶ durch Einbinden über den Add-Ins-Manager ▶ durch direktes Öffnen des Add-Ins
Workbook_AddinUninstall	Wenn ein Add-In über den Add-Ins-Manager deaktiviert wird.
Workbook_BeforeClose	Bevor eine Arbeitsmappe geschlossen wird.
Workbook_BeforePrint	Bevor ein Druckauftrag ausgeführt wird.
Workbook_BeforeSave	Bevor eine Arbeitsmappe gespeichert wird.

Tabelle 73: Alle Ereignisse für die Arbeitsmappe

Grund-lagen
Allge-mein
Datu-Zeit
tungen
Steuer-elemen
Befehl-leisten
Objekt
Dia-gramm
Ereig-nisse
User-Form
Web/Mail
Extern
Ge-mischtes
Specia

Ereignis	Aufruf / Beschreibung
Workbook_Deactivate	Wenn eine Arbeitsmappe deaktiviert wird, indem eine andere Excel-Arbeitsmappe aktiviert oder die Arbeitsmappe minimiert wird.
Workbook_NewSheet	Wenn ein neues Blatt in die Arbeitsmappe eingefügt wird. ► Tabellenblatt ► Diagrammblatt ► Makroblatt Tritt nicht ein, wenn ► ein Blatt innerhalb der Arbeitsmappe kopiert oder verschoben wird. ► ein Blatt aus einer anderen Arbeitsmappe importiert wird.
Workbook_Open	Wenn eine Arbeitsmappe geöffnet wird.
Workbook_PivotTableCloseConnection	Nachdem ein PivotTable-Bericht die Verbindung zu seiner Datenquelle geschlossen hat.
Workbook_PivotTableOpenConnection	Nachdem ein PivotTable-Bericht die Verbindung zu seiner Datenquelle geöffnet hat.
Workbook_SheetActivate	Wenn ein beliebiges Tabellen- oder Diagrammblatt der Arbeitsmappe aktiviert wird.
Workbook_SheetBeforeDoubleClick	Wenn in einer Zelle eines beliebigen Tabellenblattes der Arbeitsmappe mit der linken Maustaste ein Doppelklick durchgeführt wird.
Workbook_SheetBeforeRightClick	Wenn in einer Zelle oder Zeilen-/Spaltenkopf eines beliebigen Tabellenblattes der Arbeitsmappe ein Klick mit der rechten Maustaste durchgeführt wird. Die übliche Anzeige des Kontextmenüs nach einem Rechtsklick erfolgt nach Verarbeitung der Ereignis-Prozedur.
Workbook_SheetCalculate	Nachdem in einem beliebigen Tabellenblatt einer beliebigen geöffneten Arbeitsmappe eine Neuberechnung ausgeführt wurde. Dieses Ereignis ist das einzige, welches auch bei Neuberechnungen in einer anderen geöffneten Arbeitsmappe aufgerufen wird.
Workbook_SheetChange	Nachdem auf einem beliebigen Tabellenblatt der Arbeitsmappe ein Zellinhalt geändert wird.
Workbook_SheetDeactivate	Nachdem ein beliebiges Tabellenblatt verlassen wurde und dafür ein anderes Tabellenblatt der Arbeitsmappe aktiviert wird. ► Tritt vor dem Workbook_SheetActivate-Ereignis ein.
Workbook_SheetFollowHyperlink	Wenn der Anwender auf einen in einer Zelle hinterlegten Hyperlink in einer beliebigen Tabelle klickt.
Workbook_SheetPivotTableUpdate	Wenn eine Pivot-Tabelle auf einem beliebigen Tabellenblatt aktualisiert wird.

Tabelle 73: Alle Ereignisse für die Arbeitsmappe (Forts.)

Ereignis	Aufruf / Beschreibung
Workbook_SheetSelection-Change	Wenn sich die Zellmarkierung auf einem beliebigen Tabellenblatt der Arbeitsmappe ändert. ► Bei Zellwechsel ► Bei Markierung von Zeilen-/Spaltenköpfen
Workbook_WindowActivate	Wenn das Arbeitsmappenfenster der Arbeitsmappe aktiviert wird. ► Tritt nach dem Workbook_Activate -Ereignis ein.
Workbook_Window-Deactivate	Wenn das Arbeitsmappenfenster der Arbeitsmappe deaktiviert wird. ► Tritt vor dem Workbook_Deactivate-Ereignis ein.
Workbook_WindowResize	Wenn die Größe des Arbeitsmappenfensters geändert oder die Arbeitsmappe minimiert wird. Ist das Arbeitsmappenfenster nicht maximiert und wird von einer anderen Arbeitsmappe zu dieser gewechselt, treten folgende Ereignisse nacheinander ein: 1. Workbook_Activate 2. Workbook_WindowActivate 3. Workbook_WindowResize

Tabelle 73: Alle Ereignisse für die Arbeitsmappe (Forts.)

In der Beispieldatei werden alle Workbook-Ereignisse überwacht. Sobald ein Ereignis eintritt, werden Sie über ein Meldungsfenster mit Angabe des aufgerufenen Ereignisses informiert.

Da wir hier aus Platzgründen nicht alle Makros der Beispieldatei aufführen können, stellen wir Ihnen an dieser Stelle stellvertretend für alle weiteren Ereignisse das erste und letzte Ereignis aus der Liste vor.

```
'=====
' Auf CD      Buchdaten\Beispiele\09_Ereignisse\
' Dateiname  09_02_AllWorkbookEvents.xls
' Ereignis   DieseArbeitsmappe
'=====

Private Sub Workbook_Activate()
    MsgBox "Workbook_Activate"
End Sub

Private Sub Workbook_WindowResize(ByVal Wn As Window)
    MsgBox "Workbook_WindowResize"
End Sub
```

Zu den wichtigsten Ereignissen haben wir in den nächsten Rezepten einige Beispiele zusammengestellt.

292 Aktuelles Monatsblatt anzeigen

Oft verwendet man eine Excel-Datei, um Daten eines Jahres in einzelnen Monatsblättern abzulegen. Praktisch wäre es, wenn gleich nach dem Öffnen einer Jahresmappe das Tabellenblatt des aktuellen Monats angezeigt würde. In unserer Beispieldatei *09_03_WorkbookOpenFindSheet.xls*

Grundlagen
Allgemein
Datum und Zeit
Formeln
Diagramme
Objekte
Diagramme
Ereignisse
User-Formen
Web/Mail
Extern
Gemischtes
Spezial

stehen in jedem Tabellenblatt in der Zelle A1 fortlaufend die Monatsnamen. Diese setzen sich aus der Formel

```
=DATUM(JAHR(HEUTE());1;1) ' für Januar
```

zusammen und sind im benutzerdefinierten Format »MMMM« formatiert.

Die Beschriftung der Tabellenregister haben wir nicht von Hand nach dem entsprechenden Monat aus dem Zellinhalt A1 vorgenommen, sondern diese Aufgabe einem kleinen Makro übergeben.

Kommt derselbe Monatsname in mehreren Tabellenblättern in der Zelle A1 vor, würde Excel bei dem Versuch, einen bereits verwendeten Tabellennamen noch einmal zu vergeben, eine Fehlermeldung ausgeben. Diese Fehlermeldung fangen wir der Prozedur `RenameSheetName` mit der Anweisung `On Error Resume Next` ab. Der ursprüngliche Tabellennamen wird in diesem Fall beibehalten.

```
' =====
' Auf CD      Buchdaten\Beispiele\09_Ereignisse\
' Dateiname  09_03_WorkbookOpenFindSheet.xls
' Modul      mdl_01_RenameSheetName
' =====
```

```
Sub RenameSheetName()
    Dim ws As Worksheet

    For Each ws In ActiveWorkbook.Worksheets
        On Error Resume Next
        ws.Name = Format(ws.Range("A1"), "MMMM")
    Next
End Sub
```

Wird nun die Datei geöffnet, wird das `Workbook_Open`-Ereignis aktiv und liest den aktuellen Monat anhand des eingestellten Datums aus der Systemsteuerung aus. Dieses Datum wird im Monatsformat in der Variablen `strName` festgehalten.

Excel würde eine Fehlermeldung ausgeben, wenn in der Arbeitsmappe kein Tabellenblatt mit dem Namen des aktuellen Monats existiert. Aus diesem Grund fragen wir in einer Schleife alle vorhandenen Tabellenblattnamen ab. Wenn das Blatt gefunden wird, wird es selektiert und das Makro beendet. Ist kein Tabellenblatt mit dem angeforderten Monat vorhanden, erscheint eine im Code definierte Fehlermeldung.

```
' =====
' Auf CD      Buchdaten\Beispiele\09_Ereignisse\
' Dateiname  09_03_WorkbookOpenFindSheet.xls
' Ereignis    DieseArbeitsmappe
' =====

Private Sub Workbook_Open()
    Dim strName As String
    Dim ws As Worksheet
```

```

strName = Format(Date, "MMMM")

For Each ws In ActiveWorkbook.Worksheets
    If ws.Name = strName Then
        Worksheets(strName).Select
        Exit Sub
    End If
Next

MsgBox "Das Tabellenblatt" & _
    vbNewLine & _
    "      " & strName & "      " & _
    vbNewLine & _
    "existiert nicht in dieser Arbeitsmappe!", _
    vbExclamation

End Sub

```

293 Ausschneiden und Löschen verhindern

In unserem nächsten Beispiel zeigen wir Ihnen, wie Sie, ohne einen Blattschutz für das Tabellenblatt zu vergeben, gezielt Funktionen von Excel außer Kraft setzen können. Wir machen es dem Benutzer unmöglich, die Befehle AUSSCHNEIDEN, VERSCHIEBEN und LÖSCHEN zu verwenden. Dies kann sinnvoll sein, wenn Sie in einer Datei Bezüge verwenden, die durch diesen Befehl verloren gehen könnten. In Excel gibt es viele Möglichkeiten, einen Zellinhalt auszuschneiden oder zu entfernen. Dennoch ist es in unserem Beispiel möglich, einen Zellinhalt durch Überschreiben mit einem anderen Wert zu verändern. Lediglich das Ausschneiden, Verschieben und Entfernen eines Zellinhalts wird hier unterbunden.

Alle uns bekannten Möglichkeiten, einen Zellinhalt auszuschneiden oder zu löschen, sind nachfolgend aufgelistet.

- ▶ Menü EXTRAS | OPTIONEN | BEARBEITEN | DRAG AND DROP VON ZELLEN
- ▶ Menü BEARBEITEN | AUSSCHNEIDEN
- ▶ Menü BEARBEITEN | LÖSCHEN...
- ▶ Menü BEARBEITEN | ZELLEN LÖSCHEN...
- ▶ Zelle, Zeile, Spalte KONTEXTMENÜ | AUSSCHNEIDEN
- ▶ Zelle, Zeile, Spalte KONTEXTMENÜ | ZELLEN LÖSCHEN
- ▶ Zelle, Zeile, Spalte KONTEXTMENÜ | INHALTE LÖSCHEN
- ▶ Tastenkombination  +  (ausschneiden)
- ▶ Tastenkombination  +  (ausschneiden)
- ▶ Taste  (löschen)
- ▶ Taste  (löschen)

Im Workbook_Aktiviere-Ereignis der Beispieldatei wird das in einem Modul gespeicherte Makro UnableToCut zur Deaktivierung der Befehle hinterlegt. Dieses Ereignis wird sowohl beim Öffnen als auch beim Aktivieren der Datei, zum Beispiel beim Wechseln aus einer anderen Excel-Arbeitsmappe, ausgelöst. Im Workbook_Deaktiviere-Ereignis setzen Sie den Aufruf des Makros AbleToCut

Grundlagen

Allgemein

Datum
Zeit

Funktionen

Steuerelemente

Befehle
leisten

Objekt

Diagramm

Ereignisse

User-Form

Web/
Mail

Extern

Gemischtes

Spezial

ein, damit in anderen Excel-Anwendungen die vormals gesperrten Befehle wieder funktionieren. Dieses Ereignis wird bei jeder Deaktivierung der Arbeitsmappe ausgelöst, also auch beim Schließen der Excel-Datei.

```
'=====
' Auf CD      Buchdaten\Beispiele\09_Ereignisse\
' Dateiname  09_04_CutAndDelete.xls
' Ereignis   DieseArbeitsmappe
'=====

Private Sub Workbook_Activate()
    UnableToCut
End Sub

Private Sub Workbook_Deactivate()
    AbleToCut
End Sub
```

Um sicher zu gehen, dass kein Benutzer der Datei die Makros über das Menü EXTRAS | MAKRO | MAKROS aufrufen kann, haben wir die Module als `Private` deklariert.

```
Option Private Module
```

Die Makros, die mit der Option `Private Module`-Anweisung als privat deklariert wurden, stehen immer noch innerhalb des Projekts zur Verfügung, aber nicht für andere Anwendungen oder Projekte. Ebenso werden sie in der Liste der zur Verfügung stehenden Makros nicht aufgeführt.

Auf eine mit `Private` deklarierte Sub-Prozedur kann nur durch andere Prozeduren aus dem Modul zugegriffen werden, in dem sie deklariert wurde.

```
Private Sub MeinMakro ()
```

Damit ein Aufruf der Makros aus den Ereignis-Prozeduren dennoch möglich ist, muss der Makroaufruf über die `Run`-Methode durchgeführt werden.

```
Application.Run "MeinMakro"
```

Die oben aufgeführten Möglichkeiten, um einen Zellinhalt auszuschneiden oder zu löschen, werden in den folgenden Makros deaktiviert bzw. aktiviert.

Hinweis

Auf die `OnKey`-Methode, welche ebenfalls zu den Ereignissen zählt, wird in dieser Kategorie an späterer Stelle eingegangen.

Die Bearbeitung der Menübefehle wird in der Kategorie Befehlsleisten ausführlich erläutert.

```
'=====
' Auf CD      Buchdaten\Beispiele\09_Ereignisse\
' Dateiname  09_04_CutAndDelete.xls
```

```
' Modul      mdl_01_UnableCutAndDelete
'=====

Option Explicit
Option Private Module

Sub UnableToCut()
    'Tastenkombinationen deaktivieren
    Application.OnKey "^x", ""      ' Strg + x
    Application.OnKey "^v", ""      ' Strg + v
    Application.OnKey "{DEL}", ""   ' Entf
    Application.OnKey "+{DEL}", ""  ' Shift + Entf
    Application.OnKey "{BACKSPACE}", "" ' Rücktaste

    ' Schaltflächen deaktivieren
    CutAndDelete 21, False          ' Ausschneiden
    CutAndDelete 478, False         ' Zellen löschen...
    CutAndDelete 292, False         ' Zellen löschen (Zelle)
    CutAndDelete 293, False         ' Zellen löschen (Zeile)
    CutAndDelete 294, False         ' Zellen löschen (Spalte)
    CutAndDelete 313, False         ' Ersetzen
    CutAndDelete 3125, False        ' Inhalte löschen
    CutAndDelete 30021, False       ' Löschen
End Sub

'=====
' Auf CD      Buchdaten\Beispiele\09_Ereignisse\
' Dateiname   09_04_CutAndDelete.xls
' Modul      mdl_02_AbleCutAndDelete
'=====

Option Explicit
Option Private Module

Sub AbleToCut()
    'Tastenkombinationen aktivieren
    Application.OnKey "^x"          ' Strg + x
    Application.OnKey "^v"          ' Strg + v
    Application.OnKey "{DEL}"       ' Entf
    Application.OnKey "+{DEL}"      ' Shift + Entf
    Application.OnKey "{BACKSPACE}" ' Rücktaste

    ' Schaltflächen aktivieren
    CutAndDelete 21, True           ' Ausschneiden
    CutAndDelete 478, True          ' Zellen löschen...
    CutAndDelete 292, Trur          ' Zellen löschen (Zelle)
    CutAndDelete 293, True          ' Zellen löschen (Zeile)
    CutAndDelete 294, True          ' Zellen löschen (Spalte)
    CutAndDelete 313, True          ' Ersetzen
    CutAndDelete 3125, True         ' Inhalte löschen
    CutAndDelete 30021, True        ' Löschen
End Sub
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

Beim Aufruf der `CutAndDelete`-Prozedur zum Deaktivieren bzw. Aktivieren der Schaltflächen müssen die jeweilige ID und der Wahrheitswert `True` oder `False` als Argumente an die Prozedur übergeben werden. In einer `For...Each`-Schleife werden alle Befehlsleisten der Excel-Anwendung durchlaufen. Mit der `FindControl`-Methode wird das Steuerelement anhand der angegebenen ID in den Leisten gesucht. Das Argument `Recursive` wird auf `True` gesetzt, so dass die Befehlsleiste und alle zugehörigen Untersymbolleisten in die Suche eingeschlossen werden.

Wird das gesuchte Steuerelement nicht gefunden, gibt die Objektvariable `cmdBarCtrl` den Wert `Nothing` zurück. In einer `If`-Anweisung wird daraufhin überprüft, ob `cmdBarCtrl` nicht den Wert `Nothing` enthält:

```
If Not cmdBarCtrl Is Nothing Then
```

Die `Enabled`-Eigenschaft des Steuerelementes wird dann auf den angegebenen Wahrheitswert gesetzt. Die `CellDragAndDrop`-Eigenschaft, welche das Ziehen und Kopieren von Zellen steuert, wird entsprechend dem angegebenen Argument `blnState` eingestellt.

```
' =====
' Auf CD      Buchdaten\Beispiele\09_Ereignisse\
' Dateiname   09_04_CutAndDelete.xls
' Modul       mdl_03_CutAndDelete
' =====

Sub CutAndDelete(intId As Integer, _
                 blnState As Boolean)

    Dim cmdBar As CommandBar
    Dim cmdBarCtrl As CommandBarControl

    For Each cmdBar In Application.CommandBars
        Set cmdBarCtrl = cmdBar.FindControl(ID:=intId, _
                                           Recursive:=True)

        If Not cmdBarCtrl Is Nothing Then
            cmdBarCtrl.Enabled = blnState
        End If
    Next

    Application.CellDragAndDrop = blnState    ' Drag & Drop
End Sub
```

294 Ereignisse ignorieren

Es gibt einen kleinen Trick, um Ereignis-Prozeduren schon vor dem Öffnen einer Excel-Datei vorübergehend zu deaktivieren:

Vor dem Öffnen einer Excel-Arbeitsmappe, welche Makros enthält, erhalten Sie einen Startdialog. In diesem müssen Sie angeben, ob die Datei mit oder ohne Makrounterstützung geöffnet wird. Halten Sie die -Taste auf Ihrer Tastatur gedrückt und klicken Sie dann im Startdialog auf die Schaltfläche **MAKROS AKTIVIEREN**. Auf diese Weise werden alle vorhandenen Ereignis-Prozeduren ignoriert.

Im Gegensatz zu einer Arbeitsmappe, in welcher Makros deaktiviert wurden, werden bei dieser Methode manuell aufgerufene Makros weiterhin ausgeführt. Sobald eine allgemeine Prozedur abgearbeitet wurde, sind auch die Ereignis-Prozeduren wieder einsatzbereit. Befindet sich eine benutzerdefinierte Funktion in einem Modul der Datei, dann wird diese durch eine Neuberechnung des Tabellenblattes aufgerufen. Dieser Vorgang löst ebenfalls die Aktivierung von Ereignis-Prozeduren aus.

Starten Sie zum Ausprobieren die Beispieldatei *09_05_IgnoreEvents.xls* mit gedrückter -Taste, während Sie die Schaltfläche **MAKROS AKTIVIEREN** drücken. In der Datei sind das `Workbook_Open`- sowie `Worksheet_SelectionChange`-Ereignis hinterlegt. Diese werden so lange nicht ausgeführt, bis Ereignisse durch einen Sub-Aufruf aktiviert werden.

```
'=====
' Auf CD      Buchdaten\Beispiele\09_Ereignisse\
' Dateiname  09_05_IgnoreEvents.xls
' Ereignis   DieseArbeitsmappe
'=====

Private Sub Workbook_Open()
    MsgBox "Datei wird geöffnet"
End Sub

Private Sub Worksheet_SelectionChange(ByVal Target As Range)
    MsgBox Target.Address(0, 0)
End Sub
```

Zur Aktivierung von ignorierten Ereignissen muss ein Makro durchlaufen werden. Die Prozedur muss nicht einmal Befehle enthalten, der Makroaufruf alleine reicht schon für die Aktivierung aus. In der Prozedur `AbleEvents` haben wir das kürzeste Makro, das es in VBA gibt, einer Schaltfläche zugewiesen.

```
'=====
' Auf CD      Buchdaten\Beispiele\09_Ereignisse\
' Dateiname  09_05_IgnoreEvents.xls
' Modul      mdl_01_AbleEvents
'=====

Sub AbleEvents()
End Sub
```

295 Ereignisse aus- und einschalten

Wählen Sie beim Öffnen einer Excel-Datei im angezeigten Dialog die Schaltfläche **MAKROS DEAKTIVIEREN**, dann werden in der Arbeitsmappe weder Ereignis-Prozeduren noch allgemeine Makros ausgeführt. Erst nachdem die Mappe geschlossen wurde und Sie bei erneutem Öffnen **MAKROS AKTIVIEREN** wählen, sind alle verfügbaren Prozeduren wieder einsatzbereit. Haben Sie eine Arbeitsmappe mit Makrounterstützung geöffnet, dann sind vorhandene Ereignis-Prozeduren wie

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

Workbook_Open und Workbook_Aktiviere bereits abgearbeitet. Erst danach können Ereignisse mit der EnableEvents-Eigenschaft deaktiviert werden. Wenn Sie eine Datei speichern wollen, ohne das BeforeSave-Ereignis auszulösen, deaktivieren Sie vorab alle Ereignisse.

```
Application.EnableEvents = False
ActiveWorkbook.Save
Application.EnableEvents = True
```

Damit Ereignis-Prozeduren wieder ausgeführt werden, muss dieser Befehl am Ende der Prozedur wieder zurückgesetzt werden.

```
'=====
' Auf CD      Buchdaten\Beispiele\09_Ereignisse\
' Dateiname   09_05_IgnoreEvents.xls
' Modul       mdl_02_Events
'=====

Sub EventsDeaktiviere()
    Application.EnableEvents = False
End Sub

Sub EventsAktiviere()
    Application.EnableEvents = True
End Sub
```

296 Excel-Datei ohne Makros unbrauchbar

Manchmal wird eine Excel-Datei durchgängig von Makros unterstützt und gesteuert. Zum Beispiel wird beim Öffnen der Arbeitsmappe eine Symbolleiste mit integrierten Makros zur Bedienung des Programms erstellt, oder bestimmte Funktionen sollen für den Anwender nicht ausführbar sein. Wir wollen nun erreichen, dass eine Datei für den Anwender nur mit Makrounterstützung zu gebrauchen ist.

Das Prinzip ist relativ einfach. Vor jedem Speichern der Datei wird ein »Startblatt« StartSheet eingeblendet und aktiviert. Alle anderen Tabellen- und Diagrammblätter der Arbeitsmappe werden mit der xlSheetVeryHidden-Methode der Worksheet-Eigenschaft Visible ausgeblendet. Durch diese Eigenschaft werden die ausgeblendeten Blätter über FORMAT | BLATT | EINBLENDEN nicht mehr angezeigt. Nach dem Speichervorgang werden die verwendeten Blätter der Arbeitsmappe wieder sichtbar gemacht und das StartSheet versteckt. Beim Öffnen der Datei ist also nur ein Blatt in der Arbeitsmappe sichtbar: das StartSheet. Hier können Sie Informationen für den Anwender hinterlegen. Den Informationstext schreiben Sie zum Beispiel in eine Zelle des Startblattes und formatieren das Blatt nach Ihren eigenen Vorstellungen.

In der Beispielmappe zeigt das StartSheet ein Excel-Meldungsfenster in Form einer Grafik, welche wir selbst angefertigt haben, an (Wie die Grafik erzeugt wurde, erfahren Sie nach dem Code). Dazu erstellen Sie eine Prozedur und lassen Ihren Text in einem Meldungsfenster MsgBox anzeigen.


```
' =====  
' Auf CD      Buchdaten\Beispiele\09_Ereignisse\  
' Dateiname   09_06_UnableWithoutMakros.xls  
' Tabelle     StartSheet  
' Modul       mdl_01_ErrorMessage  
' =====
```

```
Sub ErrorMessage()  
    MsgBox _  
        "Sollte dieser Startbildschirm nicht automatisch " & _  
        "nach ca. 5 Sekunden ausgeblendet werden," & _  
        vbNewLine & _  
        "wurde die Ausführung von Makros in dieser Arbeitsmappe" & _  
        "untersagt!" & _  
        vbNewLine & vbNewLine & _  
        "Mögliche Gründe:" & _  
        vbNewLine & _  
        "- Die Sicherheitsstufe für Makros wurde auf >>niedrig<<" & _  
        "gestellt" & _  
        vbNewLine & _  
        "- Beim Programmstart wurden Makros deaktiviert" & _  
        vbNewLine & vbNewLine & _  
        "Diese Datei kann ohne Makrounterstützung nicht" & _  
        "bearbeitet werden." & _  
        vbNewLine & _  
        "Zur Bearbeitung müssen Sie diese Datei neu starten und" & _  
        "Makros aktivieren.", _  
        vbExclamation, _  
        "Information"  
End Sub
```

Rufen Sie das Makro manuell auf, um das Meldungsfenster anzuzeigen. Aus dieser Meldung können Sie nun eine eigenständige Grafik erzeugen, indem Sie die Tastenkombination **Alt + Druck** drücken, während die Meldung auf dem Bildschirm angezeigt wird. Durch diese Tastenkombination wird das aktive Fenster (Meldungsfenster) als Grafik in die Zwischenablage befördert. Nachdem Sie das Meldungsfenster mit einem Klick auf OK geschlossen haben, fügen Sie über **BEARBEITEN | EINFÜGEN** die erzeugte Grafik, auch »Screenshot« (Bildschirmfoto) genannt, in Ihr Tabellenblatt ein.

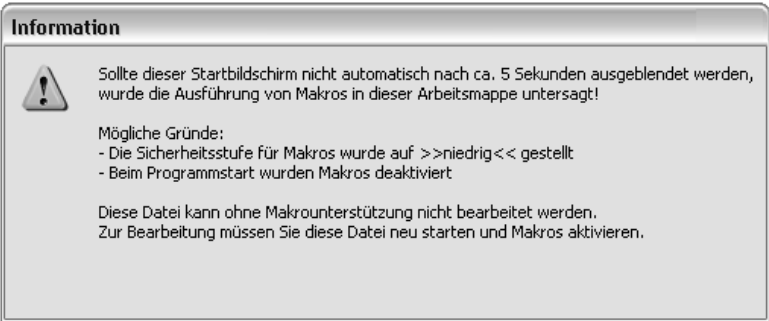


Abbildung 239: Meldungsfenster als Grafik erstellen

Grundlagen
Allgemein

Datum
Zeit

Diagramme

Steuerelemente

Befehle
leisten

Objekte

Diagramm

Ereignisse

User-Form

Web/Mail

Extern

Gemischtes

Spezial

Auf den zweiten Blick fällt Ihnen bestimmt auf, dass die Abbildung 239 kein Originalabbild des erzeugten Meldungsfensters aus dem Makro `ErrorMessage` ist. Wir haben den OK-Button und das Schließen-Kreuz verschwinden lassen. Diese Grafik ruft keine Aktionen hervor, und würde eventuell Verwirrung beim Anwender stiften. Ziehen Sie dazu aus der ZEICHNEN-Symbolleiste jeweils ein Rechteck (etwas größer als die OK-Schaltfläche bzw. das Kreuz) auf und verwenden als Füllfarbe die gewünschte Hintergrundfarbe. In unserem Beispiel wurden die verwendeten Rechtecke über ein Bildbearbeitungsprogramm hergestellt. Diese Zeichnungsobjekte platzieren Sie exakt über die zu versteckenden Objekte und verdecken sie somit.

Das Startblatt haben wir mit einem Blattschutz und dem Passwort »Codebook« versehen. Da die Beispieldatei mit der Excel-2002-Version erstellt wurde, haben wir unter EXTRAS | SCHUTZ | BLATT SCHÜTZEN die Option NICHT GESPERRTE ZELLEN AUSWÄHLEN gewählt. Diese Einstellung bewirkt, dass auf diesem Blatt nur »nicht gesperrte Zellen« markiert oder geändert werden können. Die Liste der verschiedenen Optionen zum Blattschutz wurde erst mit Excel 2002 eingeführt und steht vorherigen Excel-Versionen nicht zur Verfügung.

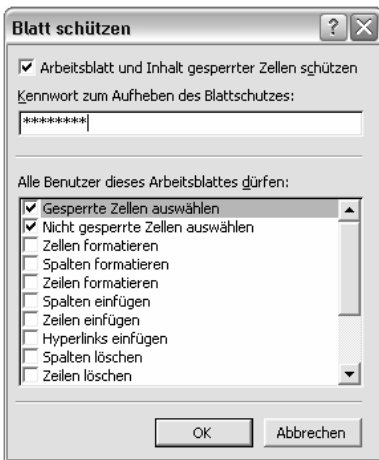


Abbildung 240: Blattschutzoptionen auswählen

Arbeiten Sie mit früheren Excel-Versionen, dann ist es gegenüber Excel 2002 möglich, bei eingeschaltetem Blattschutz gesperrte Zellen auszuwählen. Erst bei dem Versuch, einen Inhalt zu ändern, wird Excel mit einer Fehlermeldung antworten. Um in älteren Excel-Versionen das Selektieren gesperrter Zellen bei eingestelltem Blattschutz zu verhindern, müssen Sie bei jedem Öffnen der Arbeitsmappe im `Workbook_Open`-Ereignis den Befehl dazu neu setzen. Verwenden Sie dazu folgende Anweisung:

```
Worksheets("Tabelle1").EnableSelection = xlUnlockedCells
```

Ausführliche Informationen und Beispiele zum Blattschutz finden Sie in der Kategorie »Allgemein«.

Kommen wir wieder zurück zu unserer Beispielmappe. Zunächst beschäftigen wir uns mit dem Ausblenden der relevanten Blätter der Datei und dem Anzeigen des Startblattes. Das Makro

HideSheets wird in einem allgemeinen Modul mit der Option Private Module-Anweisung gespeichert (siehe auch Rezept 292).

In diesem Modul wird die Variable wsMemorySheet öffentlich deklariert, damit das Makro ShowSheets, welches in einem anderen Modul gespeichert ist, ebenfalls auf diese Variable zugreifen kann. Bevor die Blätter ausgeblendet werden, wird der Name des gerade aktiven Blattes in dieser Variablen gespeichert. Beim späteren Einblenden der Blätter, nach dem Speichervorgang, wird das vor der Speicherung aktive Blatt wieder selektiert. Ohne diese Maßnahme würde Excel das letzte Blatt in der Reihe der Tabellenregister auf dem Bildschirm anzeigen und nicht das zuletzt aktive. Weitere Erläuterungen zum Ablauf des Makros wurden als Kommentar im Code hinterlegt.

```
'=====
' Auf CD      Buchdaten\Beispiele\09_Ereignisse\
' Dateiname  09_06_UnableWithoutMakros.xls
' Modul      mdl_02_HideSheets
'=====

Option Explicit
Option Private Module

Public wsMemorySheet As Worksheet

Sub HideSheets()
    Dim objSheet As Object
    Dim strName As String

    ' Aktives Tabellenblatt vor Speicherung merken
    Set wsMemorySheet = ThisWorkbook.ActiveSheet

    ' Name des Startblattes
    strName = "StartSheet"

    ' Startblatt einblenden
    ThisWorkbook.Worksheets(strName).Visible = xlSheetVisible

    ' Alle Blätter bis auf das "StartSheet" verstecken
    For Each objSheet In ThisWorkbook.Sheets
        If objSheet.Name <> strName Then
            objSheet.Visible = xlSheetVeryHidden
        End If
    Next objSheet

End Sub
```

Das nachfolgende Makro blendet alle Blätter der Arbeitsmappe wieder ein und versteckt das Startblatt vor den Augen des Anwenders.

```
'=====
' Auf CD      Buchdaten\Beispiele\09_Ereignisse\
' Dateiname  09_06_UnableWithoutMakros.xls
' Modul      mdl_03_ShowSheets
'=====
```

Grundlagen

Allgemein

Datum
Zeit

tungen

Steuer
elemente

Befehl
leisten

Objekt

Diagramm

Ereignisse

User-
Form

Web/
Mail

Extern

Gemischtes

Spezial

Option Explicit

Option Private Module

Sub ShowSheets()

Dim objSheet As Object

Dim strName As String

' Name des Startblattes

strName = "StartSheet"

' Alle Blätter einblenden

For Each objSheet In ThisWorkbook.Sheets

If objSheet.Name <> strName Then

objSheet.Visible = xlSheetVisible

End If

Next objSheet

' Startblatt verstecken

ThisWorkbook.Worksheets(strName).Visible = xlSheetVeryHidden

' Aktives Tabellenblatt vor Speicherung selektieren

If Not wsMemorySheet Is Nothing Then

wsMemorySheet.Select

End If

End Sub

Nachfolgend gehen wir davon aus, dass die Makro-Sicherheitseinstellungen auf MITTEL eingestellt sind. Wenn Sie die Beispieldatei öffnen und in dem folgenden Dialog Makros deaktivieren, dann sehen Sie nur das Startblatt STARTSHEET mit der als Grafik hinterlegten Informationsmeldung (siehe Abbildung 239). Um den Eindruck einer nicht funktionsfähigen Datei zu verstärken, wurde vor dem letzten Schließen der Datei die Anzeige der Blattregister sowie der Bildlaufleisten UNTER EXTRAS | OPTIONEN | ANSICHT deaktiviert, so dass diese bei einem Öffnen der Arbeitsmappe ohne Makrounterstützung nicht angezeigt werden.

Kommen wir nun zu den Ereignis-Prozeduren dieses Beispiels, welche im Klassenmodul DIESE-ARBEITSMAPPE hinterlegt sind. Starten Sie die Beispieldatei *09_08_UnableWithoutMakros.xls* »mit« Makrounterstützung, dann wird zunächst das Startblatt angezeigt, bevor direkt im Anschluss das Workbook_Open-Ereignis und das darin enthaltene Makro ShowSheets aufgerufen wird. Für einen kurzen Augenblick ist das Startblatt für den Anwender sichtbar. Damit dieses Aufblitzen des Tabellenblattes mit der Informationsmeldung nicht als Fehler bewertet wird, halten wir das Makro über die Wait-Methode fünf Sekunden lang, bevor die Bildschirmaktualisierung wieder aktiviert und das Makro beendet wird.

```
' =====
' Auf CD      Buchdaten\Beispiele\09_Ereignisse\
' Dateiname   09_06_UnableWithoutMakros.xls
' Ereignis    DieseArbeitsmappe
' =====
```

<pre>Private Sub Workbook_Open() 'Bildschirmaktualisierung ausschalten Application.ScreenUpdating = False ' Blätter einblenden ShowSheets ' Blattregister und Bildlaufleisten einblenden With ActiveWindow .DisplayWorkbookTabs = True .DisplayHorizontalScrollBar = True .DisplayVerticalScrollBar = True End With ' offizielles Anfangsblatt selektieren Worksheets(1).Select ' Makro 5 Sekunden anhalten Application.Wait (Now + TimeValue("0:00:05")) 'Bildschirmaktualisierung einschalten Application.ScreenUpdating = True End Sub</pre>	

Sobald die Datei gespeichert werden soll, müssen wir eingreifen, um »vor« dem Speichervorgang alle sichtbar vorhandenen Blätter der Datei aus- und das Startblatt einzublenden.

Schauen wir uns zunächst die Aufrufzeile des Workbook_BeforeSave-Ereignisses an:

<pre>Private Sub Workbook_BeforeSave(ByVal SaveAsUI As Boolean, _ Cancel As Boolean)</pre>	

In der Parameterliste wird der Parameter `SaveAsUI` vom Datentyp `Boolean` angegeben. Durch das vorangestellte Schlüsselwort `ByVal` gilt dieser Parameter als Wertparameter und tritt innerhalb der Prozedur als eigenständige Variable auf. Der Wert dieser Variablen kann durch die nachfolgende Prozedur verändert werden. Welcher Wert an `SaveAsUI` übergeben wird, entscheidet sich anhand der ausgeführten Speicherart, durch die das Ereignis aufgerufen wurde.

Hat der Anwender die Funktion `SPEICHERN` gewählt, dann wird der Variablen `SaveAsUI` der Wert `Falsch` übergeben. Dabei spielt es keine Rolle, ob der Speicher-Befehl über den Menüpunkt `DATEI | SPEICHERN`, das entsprechende Icon in der Standard-Symboleiste oder durch die Tastenkombination `[Strg] + [S]` ausgelöst wurde. Auch bei einem durch VBA ausgelösten Speichervorgang wird zwischen `SPEICHERN` und `SPEICHERN UNTER` unterschieden und der Wert der jeweiligen Speicherart in der Parameterliste übergeben.

Der Wert `Wahr` wird an die Variable `SaveAsUI` übertragen, wenn die Speicherungsart `SPEICHERN UNTER` ausgewählt wurde. Standardmäßig kann diese Funktion entweder über den Menüpunkt `DATEI | SPEICHERN UNTER`, die Taste `[F12]` oder durch VBA-Code herbeigeführt werden.

Innerhalb des `Workbook_BeforeSave`-Ereignisses soll die Arbeitsmappe zwischen dem Aus- und Einblenden der Blätter gespeichert werden. Wenn Excel keine explizite Anweisung der Speicherart

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

erhält, dann wird die im Code angesprochene Datei im Workbook_BeforeSave-Ereignis grundsätzlich überschrieben. Erfragen Sie in einer If...Then...Else-Anweisung den Wahrheitswert der Variablen SaveAsUI, um den SPEICHERN UNTER-Befehl ebenfalls zu berücksichtigen.

Der zweite Parameter, Cancel vom Datentyp Boolean, welcher in der Aufrufzeile des Workbook_BeforeSave-Ereignisses angegeben wird, ist für unsere Beispieldatei besonders wichtig. Wird der Standardwert False innerhalb der Prozedur nicht auf True geändert, dann wird der Speicherbefehl, der das Ereignis ausgelöst hat, vor Beendigung der Prozedur ausgeführt. Dabei wird ein vorheriger Speichervorgang durch den Code nicht berücksichtigt. Dies wäre für unsere Beispiel fatal, da ja zu diesem Zeitpunkt wieder alle Blätter sichtbar sind und in diesem Zustand gespeichert würden. Bei einem erneuten Öffnen der Beispielmappe stünden alle Blätter zur Verfügung und das Startblatt bliebe im Hintergrund versteckt. Setzen Sie den Wahrheitswert des Parameters Cancel auf True, um den Ereignis-Speichervorgang zu unterdrücken.

Deaktivieren Sie vor dem Speichern alle Ereignis-Aufrufe über die EnableEvents-Eigenschaft. Der Befehl ThisWorkbook.Save würde ansonsten einen weiteren Aufruf des Workbook_BeforeSave-Ereignisses bewirken. Im weiteren Verlauf der aufgerufenen Prozeduren würde dies unweigerlich zu Fehlermeldungen führen. Aktivieren Sie nach dem Speichervorgang die Ereignis-Aufrufe wieder, damit Excel wieder auf eintretende Ereignisse reagieren kann.

```
'=====
' Auf CD      Buchdaten\Beispiele\09_Ereignisse\
' Dateiname   09_06_UnableWithoutMakros.xls
' Ereignis    DieseArbeitsmappe
'=====
```

```
Private Sub Workbook_BeforeSave(ByVal SaveAsUI As Boolean, _
                                Cancel As Boolean)
```

```
    'Bildschirmaktualisierung ausschalten
    Application.ScreenUpdating = False
```

```
    'Blätter ausblenden
    HideSheets
```

```
    'Ereignisse deaktivieren
    Application.EnableEvents = False
```

```
    If SaveAsUI = True Then
        ' »Speichern unter« wurde gewählt
        Application.Dialogs(xlDialogSaveAs).Show
    Else
        ' »Speichern« wurde gewählt
        ThisWorkbook.Save
    End If
```

```
    ' Ereignisse aktivieren
    Application.EnableEvents = True
```

```
    ' Blätter einblenden
    ShowSheets
```

```
    ' Bildschirmaktualisierung einschalten
```

```
Application.ScreenUpdating = True

' Speichern aufgrund Workbook_BeforeSave-Ereignis zurücksetzen
Cancel = True

End Sub
```

Will der Benutzer die Beispielmappe zu diesem Rezept schließen, dann wird die Arbeitsmappe im Workbook_Close-Ereignis für den nächsten Öffnen-Vorgang vorbereitet. Das Workbook_Close-Ereignis tritt ein, wenn der Anwender über den Menüpunkt DATEI | SCHLIESSEN oder das »Schließen-Kreuz« des aktiven Fensters bzw. die Tastenkombination **Strg** + **F4** die Datei schließen will.



Abbildung 241: Aktive Excel-Datei schließen

Das Workbook_Close-Ereignis wird ebenfalls ausgelöst, wenn die »gesamte Excel-Anwendung« mit allen geöffneten Arbeitsmappen geschlossen wird. Über das DATEI-Menü können Sie mit einem kleinen Trick alle derzeitigen Arbeitsmappen schließen. Halten Sie die **Alt**-Taste gedrückt und klicken Sie mit der rechten Maustaste auf den Menüpunkt DATEI. Im Gegensatz zu der üblichen Option SCHLIESSEN sehen Sie jetzt die Option ALLE SCHLIESSEN.



Abbildung 242: Geänderte Schließen-Option

Alternativ können Sie über das »Schließen-Kreuz« der Excel-Anwendung bzw. die Tastenkombination **Alt** + **F4** alle geöffneten Excel-Dateien schließen.



Abbildung 243: Excel beenden

Grundlagen

Allgemein

Datum
Zeit

Formeln

Steuerelemente

Befehle
leisten

Objekt

Diagramm

Ereignisse

User-Form

Web/Mail

Extern

Gemischtes

Spezial

Werden in einer Excel-Datei nach dem letzten Speichervorgang weitere Änderungen durchgeführt, dann werden Sie vor dem Schließen von Excel über ein Meldungsfenster gefragt, ob Sie diese Änderungen speichern wollen. In unserem Beispiel wollen wir aber auf jeden Fall die Datei vor dem Schließen speichern, um unser Startblatt für die nächste Dateiöffnung wieder hervorzuholen. Von dieser Speicherung setzen Sie den Benutzer der Datei durch ein Meldungsfenster in Kenntnis. Bevor die Datei gespeichert und anschließend geschlossen wird, werden alle Blätter der Datei durch das aufgerufene Makro `HideSheets` wieder versteckt, das Startblatt angezeigt und die Blattregister sowie Bildlaufleisten ausgeblendet.

```
'=====
' Auf CD      Buchdaten\Beispiele\09_Ereignisse\
' Dateiname  09_06_UnableWithoutMakros.xls
' Ereignis   DieseArbeitsmappe
'=====
```

```
Private Sub Workbook_BeforeClose(Cancel As Boolean)

    ' Bildschirmaktualisierung ausschalten
    Application.ScreenUpdating = False

    MsgBox "Diese Datei wird beim Beenden automatisch gespeichert"

    ' Blätter ausblenden
    HideSheets

    ' Blattregister und Bildlaufleisten ausblenden
    With ActiveWindow
        .DisplayWorkbookTabs = False
        .DisplayHorizontalScrollBar = False
        .DisplayVerticalScrollBar = False
    End With

    ' Ereignisse deaktivieren
    Application.EnableEvents = False

    ThisWorkbook.Save

    ' Ereignisse aktivieren
    Application.EnableEvents = True

    ' Bildschirmaktualisierung einschalten
    Application.ScreenUpdating = True
End Sub
```

297 Datei nur mit Makrounterstützung anzeigen

Es gibt eine sehr einfache Möglichkeit, die Anzeige einer Excel-Arbeitsmappe zu verhindern, wenn Makros deaktiviert werden. Beim Schließen der Beispielmappe wird im `Workbook_BeforeClose`-Ereignis die `IsAddin`-Eigenschaft auf `True` gestellt. Dadurch besitzt die Arbeitsmappe folgende Kennzeichen:

- Das Arbeitsmappenfenster ist nicht sichtbar.
- Es können keine Tabellenblätter eingeblendet werden.

- ▶ Eventuell in der Arbeitsmappe enthaltene Makros werden im Makro-Dialogfeld nicht dargestellt (das Dialogfeld wird angezeigt, wenn Sie auf Makro im EXTRAS-Menü zeigen und auf Makros klicken).
- ▶ Sie werden nicht aufgefordert, die Arbeitsmappe zu speichern, falls Änderungen vorgenommen werden, während die Arbeitsmappe geöffnet ist.
- ▶ Makros der Arbeitsmappe können, obwohl sie nicht dargestellt werden, über das Makro-Dialogfeld ausgeführt werden. Außerdem müssen Makronamen nicht mit dem Namen der Arbeitsmappe qualifiziert werden.
- ▶ Wenn Sie beim Öffnen der Arbeitsmappe die UMSCHALTASTE gedrückt halten, hat dies keine Wirkung.
- ▶ Durch die Dateiendung *.xls wird die Arbeitsmappe nicht als Add-In (*.xla) erkannt und steht nicht unter EXTRAS | ADD-INS... zur Auswahl zur Verfügung.

Nachdem die IsAddin-Eigenschaft auf True gestellt wurde, muss die Arbeitsmappe gespeichert werden. Excel würde ansonsten das SPEICHERN-Dialogfenster anzeigen. Würde dieses verneint und wurde die Datei vorher mit der IsAddin-Eigenschaft False gespeichert, würde die Mappe mit dem Standardwert False der AddIns-Eigenschaft gespeichert.

```
'=====
' Auf CD      Buchdaten\Beispiele\09_Ereignisse\
' Dateiname  09_06a_JustWithMacros.xls
' Ereignis   DieseArbeitsmappe
'=====
```

```
Private Sub Workbook_BeforeClose(Cancel As Boolean)
    With ThisWorkbook
        .IsAddin = True
        .Save
    End With
End Sub
```

Wenn der Anwender diese Datei ohne Makrounterstützung öffnet, wird sie als Add-In geladen und das Arbeitsmappenfenster wird in der Excel-Ansicht nicht angezeigt. Sie können die Datei dann lediglich im Visual Basic Editor sehen und bearbeiten. Um die Mappe wieder normal ansehen und bearbeiten zu können, stellen Sie die IsAddin-Eigenschaft im Eigenschaftsfenster des Codemoduls DieseArbeitsmappe auf False. Makros sind dadurch jedoch weiterhin deaktiviert. Anwender, welche keine VBA-Kenntnisse haben, oder ein vorhandenes VBA-Projekt-Passwort nicht kennen, müssen zunächst die Excel-Anwendung beenden. Nur dadurch wird die Add-In-Mappe ebenfalls geschlossen. Wird die Datei anschließend mit Makrounterstützung geöffnet, erfolgt im Workbook_Open-Ereignis die Einstellung der IsAddin-Eigenschaft auf False. Dadurch ist die Arbeitsmappe wieder eine voll funktionsfähige Excel-Arbeitsmappe und verhält sich ganz normal.

```
'=====
' Auf CD      Buchdaten\Beispiele\09_Ereignisse\
' Dateiname  09_06a_JustWithMacros.xls
' Ereignis   DieseArbeitsmappe
'=====
```

```
Private Sub Workbook_Open()
```

Grundlagen

Allgemein

Datum
Zeit

ungen

Steuer
elemente

Befehl
leisten

Objekt

Diagramm

Ereignisse

User-
Form

Web/
Mail

Extern

Gemischtes

Spezial

```
ThisWorkbook.IsAddin = False
End Sub
```

298 Registerreiter neu nummerieren

Über den Menüpunkt EINFÜGEN | TABELLENBLATT oder die Tastenkombination  + **F11** fügen Sie in die aktive Arbeitsmappe ein neues Tabellenblatt ein. Sowohl bei diesem manuellen Einfügen durch den Anwender als auch durch das Einfügen neuer Blätter via VBA-Code wird das `Workbook_NewSheet`-Ereignis ausgelöst. Nachfolgende Befehle fügen aktiven Arbeitsmappen jeweils ein Tabellenblatt hinzu.

```
Sheets.Add ' erstellt ein Tabellenblatt
```

```
Sheets.Add Type:=xlWorksheet ' erstellt ein Tabellenblatt
```

Um zum Beispiel ein Diagrammblatt einzufügen, verwenden Sie die `xlSheetType`-Konstante `xlChart`:

```
Sheets.Add Type:=xlChart ' erstellt ein Diagrammblatt
```

Das neu erstellte Tabellenblatt wird zum aktiven Blatt und im Tabellenregister links neben dem zuletzt aktiven Blatt eingefügt. Excel gibt diesem Tabellenblatt selbstständig einen Namen, welcher sich standardmäßig aus dem Wort »Tabelle« und der Anzahl der seit Erstellung der Arbeitsmappe eingefügten Blätter zusammensetzt. Bestand eine Arbeitsmappe zum Beispiel aus zehn Tabellenblättern und Sie haben neun davon gelöscht, dann wird ein neu erstelltes Tabellenblatt den Namen »Tabelle11« erhalten, obwohl in dem Augenblick nur zwei Tabellenblätter in der Arbeitsmappe vorhanden sind.

Unabhängig von den Tabellenblättern werden Diagrammblätter in gleicher Weise benannt und durchnummeriert (»Diagramm1« etc). Diagrammblätter können Sie zum Beispiel über den DIAGRAMM-ASSISTENTEN erstellen. Diesen rufen Sie entweder über den Menüpunkt EINFÜGEN | DIAGRAMM oder das entsprechende Icon in der Symbolleiste auf.



Abbildung 244: Diagramm als neues Blatt einfügen

Auch für das Einfügen eines Diagrammblattes gibt es eine Tastenkombination: Markieren Sie den Bereich Ihrer Tabelle, welchen Sie als Diagramm erstellen möchten, und drücken Sie die Tastenkombination `[Windows-Taste]+[F11]`. Anhand des markierten Bereichs erstellt Excel daraufhin ein neues Diagrammblatt mit den Standardeinstellungen der Diagramm-Optionen.

Mit folgenden Syntaxen erzeugen Sie mit VBA ein neues Diagrammblatt.

```
Charts.Add          ' erstellt ein Diagrammblatt
Sheets.Add Type:=xlChart ' erstellt ein Diagrammblatt
```

Tipp

Ausführliche Informationen zu Diagrammen finden Sie in der Kategorie »Diagramme«.

In Abbildung 245 sehen Sie, dass durch das Einfügen, Löschen bzw. Verschieben von Blättern in einer Arbeitsmappe leicht ein heiles Durcheinander der einzelnen Registerbeschriftungen entstehen kann.

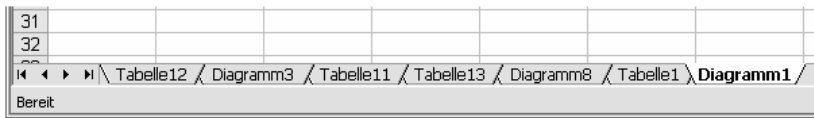


Abbildung 245: Registerreiterbeschriftung

Nachdem Sie ein Tabellen- oder Diagrammblatt in Ihre Arbeitsmappe eingefügt haben, vergibt Excel nach oben beschriebener Namensgebung einen entsprechenden Namen. Erst im Anschluss an die Namensvergabe wird das `Workbook_NewSheet`-Ereignis ausgelöst. Überraschenderweise gibt es in Excel kein analoges Ereignis, das vor dem Löschen eines Blattes ausgelöst wird.

Mit nachfolgenden Makro werden alle in der Excel-Datei vorhandenen Blätter beschriftet und fortlaufend nummeriert. Die beiden Blatt-Typen, Tabellenblatt und Diagrammblatt, werden dabei getrennt durchnummeriert. Das Ergebnis sehen Sie in Abbildung 246.

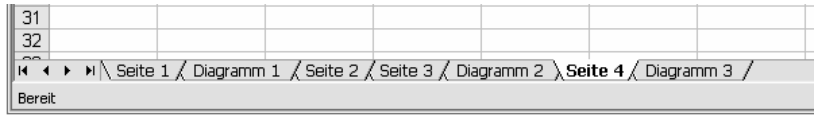


Abbildung 246: Geordnete Registerreiterbeschriftung

In der Beispieldatei erfolgt im `Workbook_NewSheet`-Ereignis lediglich ein Aufruf des Makros `RenameSheetTabs` zur Durchführung der Beschriftung. Die Prozedur `RenameSheetTabs` wird in einem allgemeinen Modul gespeichert.

```
' =====
' Auf CD   Buchdaten\Beispiele\09_Ereignisse\
' Dateiname 09_07_RenameWorkbookTabs.xls
' Ereignis  DieseArbeitsmappe
' =====

Private Sub Workbook_NewSheet(ByVal Sh As Object)
```

Grund-lagen

Allge-mein

Datu-Zeit

tungen

Steuer-elemen

Befehl-leisten

Objekt

Diagramm

Ereig-nisse

User-Form

Web/Mail

Extern

Ge-mischtes

Specia

```

    RenameSheetTabs
End Sub

Sub RenameSheetTabs()
    Dim intI As Integer
    Dim intSh As Integer
    Dim intCh As Integer

    ' alle Blätter durchnummerieren, dadurch werden spätere _
    ' Fehlermeldungen durch doppelte Namensvergabe vermieden
    For intSh = 1 To ActiveWorkbook.Sheets.Count
        Sheets(intSh).Name = intSh
    Next

    intSh = 1
    intCh = 1

    For intI = 1 To ActiveWorkbook.Sheets.Count

        ' Tabellenblätter durchnummerieren
        If TypeName(Sheets(intI)) = "Worksheet" Then
            Sheets(intI).Name = "Seite " & intSh
            intSh = intSh + 1

            ' Diagrammblätter durchnummerieren
        ElseIf TypeName(Sheets(intI)) = "Chart" Then
            Sheets(intI).Name = "Diagramm " & intCh
            intCh = intCh + 1
        End If

    Next
End Sub

```

299 Neues Blatt beschriften

Wenn Sie einen Registerreiter beschriften wollen, müssen dabei mehrere Kriterien beachtet werden: Siehe Abbildung 247 und Abbildung 248.

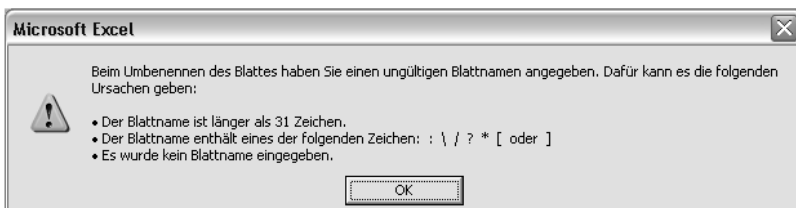


Abbildung 247: Fehlermeldung bei Namensvergabe

In unserem nächsten Beispiel verwenden wir das `Workbook_NewSheet`-Ereignis, um ein neu eingefügtes Blatt über ein Eingabefenster `InputBox` zu beschriften. Die Kopf- und Fußzeilen werden ebenfalls über diese Prozedur eingerichtet.

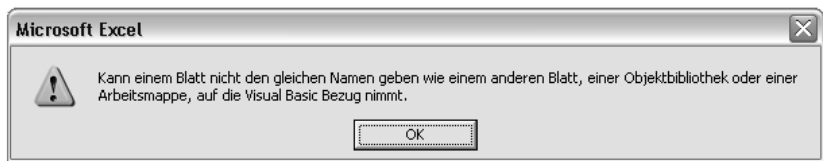


Abbildung 248: Fehlermeldung bei doppelter Namensvergabe

In der Aufrufzeile

```
Private Sub Workbook_NewSheet(ByVal Sh As Object)
```

wird der Parameter Sh als eigenständige Object-Variable an die Prozedur übergeben. Diese enthält alle Eigenschaften des eingefügten Blattes. In unserem Code wird diese Variable zum Beispiel dazu verwendet, den Blattnamen als Default-Text in der Eingabezeile einer InputBox anzuzeigen. Des Weiteren können Sie dem Sh-Objekt innerhalb der ausgelösten Ereignis-Prozedur auch Methoden zuweisen. Mit nachfolgendem Befehl wird das eingefügte Blatt an die letzte Stelle der Registerreiter gesetzt.

```
Sh.Move After:=Sheets(Sheets.Count)
```

In unserem Beispiel erfolgt die Namensvergabe über eine InputBox. Diese wird innerhalb einer Do...Loop-Schleife so lange aufgerufen, bis Sie entweder einen gültigen Namen eingeben oder deren ABBRECHEN-Schaltfläche anklicken. Nachdem Sie die InputBox abgebrochen und die nachfolgende Sicherheitsabfrage mit OK bestätigt haben, wird das eingefügte Blatt gelöscht und die Prozedur beendet.

Die Fehlerroutine zur Überprüfung eines gültigen Blattnamens erfolgt in der Funktion IsSheetNameOK(MyName As String) und wird in der Loop Until-Zeile der Do...Loop-Schleife aufgerufen.

Die Do...Loop-Schleife wird so lange wiederholt, bis die Funktion IsSheetNameOK den Wert Wahr zurückgibt.

```
'=====
' Auf CD      Buchdaten\Beispiele\09_Ereignisse\
' Dateiname   09_08_NewSheetName.xls
' Ereignis    DieseArbeitsmappe
'=====

Private Sub Workbook_NewSheet(ByVal Sh As Object)

    Do
NameInput:
        strName = Application.InputBox _
            ("Geben Sie einen Tabellenblattnamen ein", _
            , Sh.Name, , , , Type:=2)

        ' Abbrechen wurde geklickt
        If strName = "Falsch" Then
```

Grund-lagen

Allge-mein

Datu-Zeit

tungen

Steuer-elemen

Befehl-leisten

Objekt

Dia-gramm

Ereig-nisse

User-Form

Web/Mail

Extern

Ge-mischtes

Specia

```

    If MsgBox("Soll das neu erstellte Blatt " & _
        vbNewLine & _
        "      "" & ActiveSheet.Name & """" & _
        vbNewLine & _
        " wirklich gelöscht werden?", _
        vbYesNo + vbQuestion, _
        "Wirklich Abbrechen?") _
        = vbYes Then
        ' in MsgBox wurde Ja geklickt
        Application.DisplayAlerts = False
        Sh.Delete
        Application.DisplayAlerts = True
        Exit Sub
    Else
        ' in MsgBox wurde Nein geklickt
        GoTo NameInput
    End If
End If

' Schleife so lange aufrufen, bis ein gültiger _
  Tabellenblattname eingegeben wurde
Loop Until IsSheetNameOK(strName) = True

' Makroaufruf für Kopf- und Fußzeilen
HeaderAndFooter
End Sub

Public Function IsSheetNameOK(MyName As String) As Boolean
    Dim bytSh As Byte

    ' Fehlerbehandlung zurückstellen
    On Error Resume Next

    ' nichtverarbeitete Fehler aus dem Err-Objekt löschen
    ErrClear.

    ' Tabellenblatt umbenennen
    ActiveSheet.Name = MyName

    ' bei gültiger Namensvergabe erfolgt keine Fehlermeldung
    If Str(Err.Number) = 0 Then
        ' Funktion erhält den Wert <<True>>
        IsSheetNameOK = True
    Else
        ' Beschriftung des Blattes hat Fehler verursacht; _
          Überprüfung welche Ursache die Fehlermeldung hatte

        ' keine Eingabe in InputBox
        If MyName = "" Then
            MsgBox "Es wurde kein Name angegeben!" & _
                vbNewLine & _
                "Geben Sie für das neu erstellte Blatt einen " & _

```

```

        "Namen ein!", _
        vbCritical, _
        "Fehler: keine Eingabe"
    Exit Function
End If

' Namenseingabe hat mehr als 31 Zeichen
If Len(MyName) > 31 Then
    MsgBox "Der Dateiname enthält " & Len(MyName) & _
        " Zeichen!" & _
        vbNewLine & _
        "Kürzen Sie den Tabellennamen auf max. 31" & _
        " Zeichen!", _
        vbCritical, _
        "Fehler in der Zeichenlänge"
    Exit Function
End If

'Prüfung ob Name bereits existiert
For bytSh = 1 To ActiveWorkbook.Sheets.Count
    If Sheets(bytSh).Name = MyName Then
        MsgBox "Der Tabellenname existiert bereits in der" & _
            " Datei!" & _
            vbNewLine & _
            "Geben Sie einen anderen Namen ein!", _
            vbCritical, _
            "Name bereits vorhanden"
        Exit Function
    End If
Next

' keine Übereinstimmung der vorherigen Überprüfungen
MsgBox "Der gewählte Tabellenblattnamen enthält ungültige" & _
    "Sonderzeichen!" & _
    vbNewLine & _
    "Geben Sie einen anderen Namen ein!", _
    vbCritical, _
    "Blattnamen enthält eins der Sonderzeichen: " & _
    " : \ / ? *"

End If
End Function

```

Grund-
lagenAllge-
meinDatum
Zeit

tungen

Steuer
elemenBefehl
leisten

Objekt

Dia-
grammEreig-
nisseUser-
FormWeb/
Mail

Extern

Ge-
misch
tes

Specia

Nachdem der neue Tabellenblattnamen zugewiesen wurde, wird als letzter Befehl in der `Workbook_NewSheet-Prozedur` das Makro `HeaderAndFooter` zur Erstellung der Kopf- und Fußzeilen aufgerufen. Hier haben Sie die Möglichkeit, über eine `InputBox` den Text der mittleren Kopfzeile einzugeben. Belassen Sie die Eingabezeile der `InputBox` leer, oder klicken Sie auf die Schaltfläche **ABBRECHEN**, dann wird kein Text in die Kopfzeile geschrieben. Den Text der linken und rechten Fußzeile können Sie dem nachfolgenden Code entnehmen.

```
'=====
' Auf CD      Buchdaten\Beispiele\09_Ereignisse\
' Dateiname   09_08_NewSheetName.xls
' Modul       mdl_02_HeaderAndFooter
'=====

Sub HeaderAndFooter()
    Dim strCenterHeader As String

    strCenterHeader = Application.InputBox _
        ("Welcher Text soll in der mittleren Kopfzeile" & _
        " erstellt werden?", _
        "Kopf- und Fußzeilen", , , , , Type:=2)

    With ActiveSheet.PageSetup

        ' Wenn Abbruch der Inputbox, dann leere Kopfzeile erstellen
        .CenterHeader = IIf(strCenterHeader = "Falsch", "", _
            strCenterHeader)

        .LeftFooter = "Das Excel-VBA-Codebook, 2. Auflage"

        ' Angabe des Pfades, Dateiname und Blattname
        .RightFooter = ActiveWorkbook.Path & _
            Chr(10) & _
            ActiveWorkbook.Name & _
            Chr(10) & _
            strName

    End With
End Sub
```

300 Diagramm nach Ansicht löschen

In unserem nächsten Beispiel wird nach Klick auf einen Button das Makro `FastChart` aufgerufen. Der benutzte Bereich des aktiven Tabelleblattes wird über diese Prozedur als Diagramm in einem neuen Diagrammblatt angezeigt. Nachdem das Diagrammblatt erstellt wurde, wird es der öffentlichen Objekt-Variablen `objChart` zugewiesen.

Dadurch kann das Diagrammblatt in anderen Prozeduren identifiziert und bearbeitet werden.

```
'=====
' Auf CD      Buchdaten\Beispiele\09_Ereignisse\
' Dateiname   09_09_DeleteFastChart
' Modul       mdl_01_FastChart
'=====

Public objChart As Object

Sub FastChart()
    Dim rngSource As Range
```

```
' benutzten Bereich zuweisen
Set rngSource = ActiveSheet.UsedRange

' Zelle außerhalb des benutzten Bereichs markieren
Selection.SpecialCells(xlCellTypeLastCell).Offset(1, 1).Select

Application.EnableEvents = False ' Ereignisse deaktivieren
Charts.Add ' Diagrammblatt einfügen
Application.EnableEvents = True ' Ereignisse aktivieren

' Datenquelle zuweisen
ActiveChart.SetSourceData Source:=rngSource

' Objekt-Variablen aktives Blatt zuweisen
Set objChart = ActiveSheet
End Sub
```

Nachdem Sie sich das Diagrammblatt angeschaut und zu einem x-beliebigen Blatt der Arbeitsmappe gewechselt haben, soll das soeben erstellte Diagrammblatt wieder gelöscht werden. Schreiben Sie den Code dazu in das Workbook_SheetActivate-Ereignis. Es wird ausgelöst, nachdem ein Blatt der Arbeitsmappe aktiviert wurde.

```
'=====
' Auf CD      Buchdaten\Beispiele\09_Ereignisse\
' Dateiname  09_09_DeleteFastChart
' Ereignis   DieseArbeitsmappe
'=====

Private Sub Workbook_SheetActivate(ByVal Sh As Object)
' Prüfung, ob Objekt-Variable nichtleer ist
If objChart Is Nothing Then Exit Sub

With Application
.EnableEvents = False ' Ereignisse deaktivieren
.DisplayAlerts = False ' Fehlermeldungen deaktivieren
objChart.Delete ' Diagrammblatt löschen
.DisplayAlerts = True ' Fehlermeldungen aktivieren
.EnableEvents = True ' Ereignisse aktivieren
End With

Set objChart = Nothing ' Objekt-Variable zurücksetzen
End Sub
```

301 Tabellenblatt-Ereignisse (Worksheet)

Neben den Workbook-Ereignissen, welche die gesamte Excel-Datei betreffen, gibt es für jedes einzelne Tabellenblatt unabhängige Worksheet-Ereignisse. Jedes Tabellenblatt einer Excel-Arbeitsmappe verwaltet seine Ereignis-Prozeduren in »seinem« Klassenmodul. Klicken Sie im Projekt-Explorer des VBA-Editors doppelt auf einen Tabellennamen, damit in der rechten Fensterhälfte das Codeblatt zu dieser Tabelle angezeigt wird.

Grund-lagen

Allge-mein

Datu Zeit

tungen

Steuer elemen

Befehl leisten

Objekt

Dia-gramm

Ereig-nisse

User-Form

Web/ Mail

Extern

Ge-misch tes

Specia

Tipp Um von der Excel-Tabellenansicht schnell zum VBA-Editor mit dem dazugehörigen Code-
modul zu wechseln, klicken Sie mit der rechten Maustaste auf den Tabellenregister und
wählen den untersten Eintrag CODE ANZEIGEN.

Klicken Sie oberhalb des Codefensters auf den linken DropDown-Pfeil und wählen Sie den Ein-
trag WORKSHEET. Daraufhin wird die Codeschablone des Worksheet_SelectionChange-Ereignisses
erstellt. Im rechten Listenfeld werden alle verfügbaren Ereignisse für die Tabelle aufgeführt.

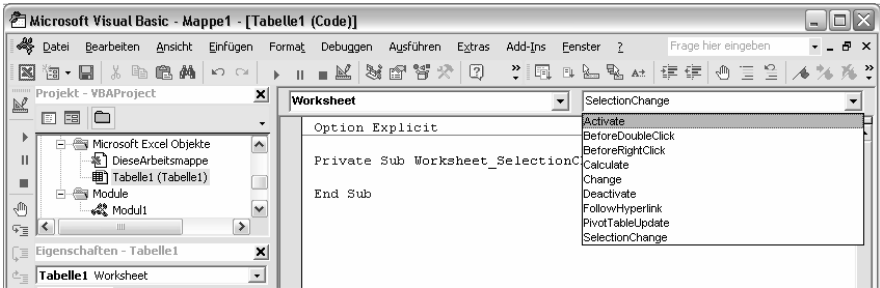


Abbildung 249: Klassenmodul der Tabelle

Nach Auswahl eines Ereignisses aus dem Listenfeld wird die entsprechende Codeschablone in
dem Codeblatt ausgegeben. Schreiben Sie die Befehle, welche beim Eintritt des gewählten Ereig-
nisses ausgeführt werden sollen, zwischen die aufgeführte Aufruf- und Endzeile der Ereignis-Pro-
zedur.

Tipp Löschen Sie Ereignis-Prozeduren in dem Codeblatt, die Sie nicht benötigen, oder auch sol-
che Codeschablonen, die keine Befehle enthalten. Diese würden sonst bei jedem eintreten-
den Ereignis unnötig aufgerufen.

Folgende Ereignisse stehen jedem Tabellenblatt zur Verfügung:

Ereignis	Aufruf /Beschreibung
Worksheet_Activate	Nachdem das Tabellenblatt aktiviert wurde
Worksheet_BeforeDoubleClick	Nach einem Doppelklick mit der linken Maustaste ► Bevor die Standardfunktion des Doppelklicks aus- geführt wird (Zellbearbeitung)
Worksheet_BeforeRightClick	Nach einem Klick mit der rechten Maustaste ► Bevor die Standardfunktion des rechten Mausclicks ausgeführt wird (Kontextmenü)
Worksheet_Calculate	Nachdem das Tabellenblatt neu berechnet wurde ► Zellformatierungen lösen keine Neuberechnung aus

Tabelle 74: Alle Ereignisse für das Tabellenblatt

Ereignis	Aufruf /Beschreibung
Worksheet_Change	Nachdem der Inhalt einer Zelle in dem Tabellenblatt durch eine Eingabe oder durch eine externe Verknüpfung verändert oder gelöscht wurde ▶ Wird nach Aktivierung eines Tabellenblattes, vor dem Sheet_Aktivate-Ereignis ausgelöst ▶ Zellformatierungen lösen kein Change-Ereignis aus
Worksheet_Deactivate	Nachdem das Tabellenblatt deaktiviert wurde
Worksheet_FollowHyperlink	Nachdem auf einen Hyperlink geklickt wurde
Worksheet_SelectionChange	Nachdem die Auswahl der markierten Zelle(n) verändert wurde ▶ Bei Zellwechsel ▶ Bei Markierung von Zeilen-/Spaltenköpfen, Bereichen ▶ Zellformatierungen lösen kein SelectionChange-Ereignis aus

Tabelle 74: Alle Ereignisse für das Tabellenblatt (Forts.)

In der Beispieldatei werden alle Worksheet-Ereignisse überwacht. Sobald ein Ereignis eintritt, werden Sie über ein Meldungsfenster mit Angabe des aufgerufenen Ereignisses informiert.

Da wir hier aus Platzgründen nicht alle Makros der Beispieldatei aufführen wollen, stellen wir Ihnen an dieser Stelle stellvertretend für alle weiteren Ereignisse das erste Ereignis aus der Liste vor.

```
'=====
' Auf CD      Buchdaten\Beispiele\09_Ereignisse\
' Dateiname  09_10_AllWorksheetsEvents.xls
' Tabelle    AllEvents
' Ereignis   Tabelle1 (AllEvents)
'=====

Private Sub Worksheet_Activate()
    MsgBox "Worksheet_Activate"
End Sub
```

302 Zellenkontextmenü verhindern

Wenn Sie mit der rechten Maustaste in eine Zelle oder einen markierten Zellenbereich auf dem Tabellenblatt klicken, wird üblicherweise das Kontextmenü der Zelle angezeigt. Dieser Rechtsklick löst vor dem Anzeigen des Kontextmenüs das Worksheet_BeforeRightClick-Ereignis aus. Beim Aufruf dieses Ereignisses wird der Parameter Cancel vom Datentyp Boolean an die Prozedur übergeben. Setzen Sie dessen Wert in der Prozedur auf True, wird nach Beendigung der Prozedur die Anzeige des Kontextmenüs deaktiviert.

Tipp

Weitere Informationen zu Kontextmenüs finden Sie in der Kategorie »Befehlsleisten«.

Grund-lagen
Allge-mein
Datu Zeit
tungen
Steuer elemen
Befehl leisten
Objekt
Dia-gramm
Ereig-nisse
User-Form
Web/Mail
Extern
Ge-misch tes
Specia

In diesem Beispiel wird die Anzeige des Kontextmenüs im Bereich A1:E25 deaktiviert.

```
'=====
' Auf CD      Buchdaten\Beispiele\09_Ereignisse\
' Dateiname  09_10_AllWorksheetsEvents.xls
' Tabelle     BeforeRightClick
' Ereignis    Tabelle2 (BeforeRightClick)
'=====

Private Sub Worksheet_BeforeRightClick _
    (ByVal Target As Range, Cancel As Boolean)

    If Not Application.Intersect(Target, Range("A1:E25")) _
        Is Nothing Then
        Cancel = True
    End If
End Sub
```

303 Auf den Spuren von Target

In einigen Ereignissen wird in der Aufrufzeile die Variable `Target` an die nachfolgende Prozedur übergeben. Neben der Adresse werden Eigenschaften, wie Zellenwert oder Formatierungen, der auslösenden Zelle oder des Bereichs an die Variable `Target` übergeben. Am Beispiel des `Worksheet_SelectionChange`-Ereignisses zeigen wir Ihnen, welche Informationen in dieser Variablen stecken und wie diese ausgewertet und geändert werden können.

```
Private Sub Worksheet_SelectionChange(ByVal Target As Range)
```

Das `Worksheet_SelectionChange`-Ereignis tritt ein, wenn durch den Anwender oder durch VBA-Code die Zellmarkierung auf dem Tabellenblatt geändert wurde. Dabei spielt es keine Rolle, ob nur eine Zelle A1, ein zusammenhängender Bereich A1:C10, eine ganze Zeile A1:IV265 oder Spalte A1:A65536 oder einzelne Zellen A1, A3, A5 mittels gedrückter `[Strg]`-Taste ausgewählt wurden.

Wird der Variablen `Target` in der Prozedur keine Eigenschaft hinzugefügt, liefert diese standardmäßig den Wert einer »einzelnen« markierten Zelle.

```
'=====
' Auf CD      Buchdaten\Beispiele\09_Ereignisse\
' Dateiname  09_11_SelectionChange.xls
' Tabelle     Target_Default
' Ereignis    Tabelle1 (Target_Default)
'=====

Private Sub Worksheet_SelectionChange(ByVal Target As Range)
    ' Wert einer einzelnen selektierten Zelle
    MsgBox Target    ' entspricht Target.Value
End Sub
```

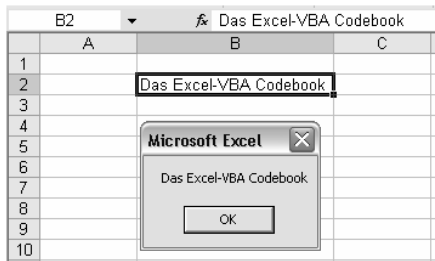


Abbildung 250: Wert einer einzelnen selektierten Zelle

Sobald mehrere Zellen selektiert wurden, verursacht obiges Makro eine Fehlermeldung, da die Variable `Target` alle markierten Zellen beinhaltet und dadurch kein eindeutiger Zellbezug hergestellt werden kann. Folgende Syntax gibt sowohl den Wert »einer« markierten Zelle als auch in einem Bereich aus mehreren Zellen den Wert der ersten Zelle zurück:

```
'=====
' Auf CD      Buchdaten\Beispiele\09_Ereignisse\
' Dateiname  09_11_SelectionChange.xls
' Tabelle     Target_Range
' Ereignis    Tabelle2 (Target_Range)
'=====

Private Sub Worksheet_SelectionChange(ByVal Target As Range)
' Wert der ersten Zelle eines Bereichs
    MsgBox Target.Cells(1, 1)
End Sub
```

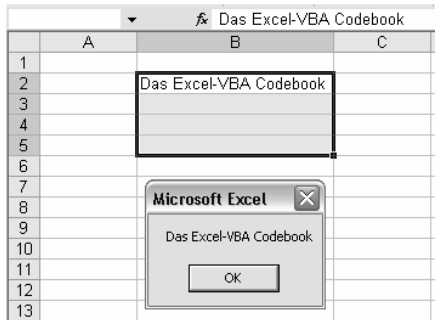


Abbildung 251: Wert der ersten Zelle eines Bereichs

Alle verfügbaren Eigenschaften und Methoden zu dem Schlüsselwort `Target` erhalten Sie, indem Sie das Wort `Target` gefolgt von einem Punkt schreiben. Wählen Sie aus der aufklappenden Intellisense-Liste das Gewünschte aus.

Die Eigenschaften der auslösenden Zelle oder des Bereichs können innerhalb der Prozedur geändert werden. Die Adresse bleibt jedoch unveränderbar.

Zu verschiedenen Tabellenblatt-Ereignissen haben wir Ihnen nachfolgend einige Beispiele aus der Praxis zusammengestellt.

Grund-lagen

Allge-mein

Datu-Zeit

tungen

Steuer-elemen

Befehl-leisten

Objekt

Dia-gramm

Ereig-nisse

User-Form

Web/Mail

Extern

Ge-mischtes

Specia

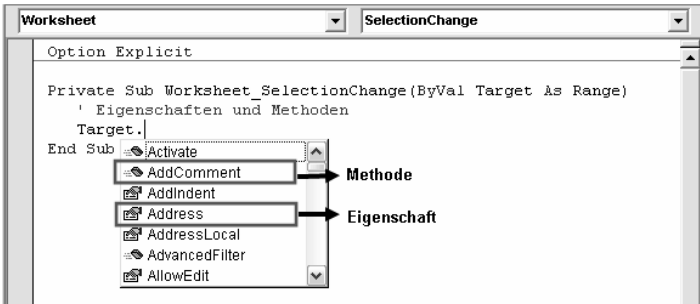


Abbildung 252: Mögliche Eigenschaften und Methoden zu Target

304 Bereichsauswahl verhindern

Auf sehr einfache Weise können Sie verhindern, dass der Anwender einen Bereich aus mehreren Zellen markieren kann. Dies ist sinnvoll, wenn Sie zum Beispiel auf einzelne Zellen in einem Tabellenblatt einen Selektierschutz gelegt haben, so dass diese Zellen nicht anwählbar sind. Durch Markierung eines Bereichs, in welchem diese geschützten Zellen enthalten sind, werden diese Zellen aber dennoch erreichbar und können kopiert werden. Mittels nachfolgender Prozedur wird auch eine Mehrfachmarkierung durch Drücken der **[Strg]**-Taste und Auswahl einzelner Zellen unterbunden.

```
' =====
' Auf CD      Buchdaten\Beispiele\09_Ereignisse\
' Dateiname   09_11_SelectionChange.xls
' Tabelle     Select one Cell
' Ereignis    Tabelle3 (Select one Cell)
' =====
```

```
Private Sub Worksheet_SelectionChange(ByVal Target As Range)
    If Target.Count = 1 Then Exit Sub

    Application.EnableEvents = False

    ' erste Zelle eines Bereichs selektieren
    Target.Cells(1, 1).Select

    Application.EnableEvents = True
End Sub
```

305 Cursorbewegungen protokollieren

Jede Änderung der Zellauswahl wird im Tabellenblatt in der ersten freien Zelle in Spalte A protokolliert. Über die Tabellen-Funktion `CountA` (Anzahl2) wird die Anzahl der nichtleeren Zellen in Spalte A ermittelt. Zu diesem Wert wird die Zahl 1 addiert, um die erste leere Zeile wiederzugeben.

```
' =====
' Auf CD      Buchdaten\Beispiele\09_Ereignisse\
' Dateiname   09_11_SelectionChange.xls
' Tabelle     CurserLog
' Ereignis    Tabelle4 (CurserLog)
' =====
```

```
Private Sub Worksheet_SelectionChange(ByVal Target As Range)
    Cells(WorksheetFunction.CountA(Colums(1)) + 1, 1) _
        .Value = Target.Address(False, False)
End Sub
```

306 Lupe vergrößert markierten Zellenbereich

Sie können in einer Tabelle die Zoom-Einstellung verkleinern, um einen größeren Bereich einzu-
sehen. Oft ist es dann nicht mehr möglich, den Zelleninhalt einer ausgewählten Zelle oder eines
Bereichs zu lesen, da die Schriftgröße zu klein ist. Folgende Worksheet_SelectionChange-Prozedur
wird ausgelöst, wenn die Zellauswahl im Tabellenblatt geändert wird. Zunächst wird die
Schriftgröße aller Zellen im benutzten Bereich UsedRange auf 10 und die Schriftfarbe auf Automa-
tik eingestellt. Diese Einstellungen entsprechen der Ausgangsformatierung der Tabelle. Allen Zel-
len im neu markierten Bereich Target wird die Schriftgröße 25 und die Schriftfarbe 5 zugewiesen.
Wenn die Grundeinstellungen der Farbpalette unverändert sind, entspricht der Farbindex 5 der
Farbe Blau. Anschließend wird über die AutoFit-Methode die Spaltenbreite aller Spalten im
Tabelleblatt automatisch geändert, um eine optimale Anpassung zu erreichen.

	A	B	C	D
1	Spalte A - Zeile 1	Spalte B - Zeile 1	Spalte C - Zeile 1	Spalte D - Zeile 1
2	Spalte A - Zeile 2	Spalte B - Zeile 2	Spalte C - Zeile 2	Spalte D - Zeile 2
3	Spalte A - Zeile 3	Spalte B - Zeile 3	Spalte C - Zeile 3	Spalte D - Zeile 3
4	Spalte A - Zeile 4	Spalte B - Zeile 4	Spalte C - Zeile 4	Spalte D - Zeile 4
5	Spalte A - Zeile 5	Spalte B - Zeile 5	Spalte C - Zeile 5	Spalte D - Zeile 5
6	Spalte A - Zeile 6	Spalte B - Zeile 6	Spalte C - Zeile 6	Spalte D - Zeile 6
7	Spalte A - Zeile 7	Spalte B - Zeile 7	Spalte C - Zeile 7	Spalte D - Zeile 7
8	Spalte A - Zeile 8	Spalte B - Zeile 8	Spalte C - Zeile 8	Spalte D - Zeile 8
9	Spalte A - Zeile 9	Spalte B - Zeile 9	Spalte C - Zeile 9	Spalte D - Zeile 9
10	Spalte A - Zeile 10	Spalte B - Zeile 10	Spalte C - Zeile 10	Spalte D - Zeile 10
11	Spalte A - Zeile 11	Spalte B - Zeile 11	Spalte C - Zeile 11	Spalte D - Zeile 11
12	Spalte A - Zeile 12	Spalte B - Zeile 12	Spalte C - Zeile 12	Spalte D - Zeile 12
13	Spalte A - Zeile 13	Spalte B - Zeile 13	Spalte C - Zeile 13	Spalte D - Zeile 13
14	Spalte A - Zeile 14	Spalte B - Zeile 14	Spalte C - Zeile 14	Spalte D - Zeile 14
15	Spalte A - Zeile 15	Spalte B - Zeile 15	Spalte C - Zeile 15	Spalte D - Zeile 15
16	Spalte A - Zeile 16	Spalte B - Zeile 16	Spalte C - Zeile 16	Spalte D - Zeile 16
17	Spalte A - Zeile 17	Spalte B - Zeile 17	Spalte C - Zeile 17	Spalte D - Zeile 17
18	Spalte A - Zeile 18	Spalte B - Zeile 18	Spalte C - Zeile 18	Spalte D - Zeile 18
19	Spalte A - Zeile 19	Spalte B - Zeile 19	Spalte C - Zeile 19	Spalte D - Zeile 19
20	Spalte A - Zeile 20	Spalte B - Zeile 20	Spalte C - Zeile 20	Spalte D - Zeile 20

Abbildung 253: Vergrößerte Ansicht der Zellauswahl

```
' =====
' Auf CD      Buchdaten\Beispiele\09_Ereignisse\
' Dateiname  09_11_SelectionChange.xls
' Tabelle    Enlargement
' Ereignis   Tabelle5 (Enlargement)
' =====
```

```
Private Sub Worksheet_SelectionChange(ByVal Target As Range)
    Application.ScreenUpdating = False

    With UsedRange
        .Font.Size = 10
        .Font.ColorIndex = xlAutomatic
    End With

    With Target
        .Font.Size = 25
        .Font.ColorIndex = 5
    End With
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

```
Columns.AutoFit
Application.ScreenUpdating = True
End Sub
```

307 Cursor in eine Eieruhr verwandeln

Über die `Cursor`-Eigenschaft können Sie das Erscheinungsbild des Mauszeigers in Microsoft Excel festlegen. Es stehen vier Mauszeiger zur Verfügung:

XlMousePointer-Konstanten	Erscheinungsbild
xlDefault	Standardmauszeiger
xlIBeam	I-Mauszeiger
xlNorthwestArrow	Nordwestpfeil-Zeiger
xlWait	Sanduhrzeiger

Tabelle 75: Verfügbare Mauszeiger

In der Beispielmappe wird über das `Worksheet_SelectionChange`-Ereignis bei Auswahl einer Zelle im Bereich D5:E25 der Mauszeiger in eine Eieruhr `xlWait` verwandelt. Klicken Sie in eine Zelle außerhalb dieses Bereichs, nimmt der Mauszeiger wieder seine Standardform `xlDefault` an.

```
'=====
' Auf CD      Buchdaten\Beispiele\09_Ereignisse\
' Dateiname  09_11_SelectionChange.xls
' Tabelle    ChangeCursor
' Ereignis   Tabelle6 (ChangeCursor)
'=====

Private Sub Worksheet_SelectionChange(ByVal Target As Range)
    If Application.Intersect(Target, Range("D5:G25")) Is Nothing Then
        Application.Cursor = xlDefault
    Else
        Application.Cursor = xlWait
    End If
End Sub
```

Achtung

Die `Cursor`-Eigenschaft wird nicht automatisch zurückgesetzt, wenn das Ausführen eines Makros beendet ist. Sie sollten den Mauszeiger auf `xlDefault` zurücksetzen, bevor Sie das Tabellenblatt oder die Arbeitsmappe verlassen.

Wenn Sie in der Beispielmappe zu einem anderen Tabellenblatt wechseln, wird im `Worksheet_Deactivate`-Ereignis der Mauszeiger auf sein Standard-Erscheinungsbild zurückgesetzt.

```
'=====
' Auf CD      Buchdaten\Beispiele\09_Ereignisse\
' Dateiname  09_11_SelectionChange.xls
' Ereignis   Tabelle6 (ChangeCursor)
```



```
'=====

Private Sub Worksheet_Deactivate()
    Application.Cursor = xlDefault
End Sub
```

308 Formeln in Kommentaren anzeigen

Ob sich ein Zelleninhalt aus einem festen Wert oder einer Formel ergibt, erfahren Sie üblicherweise nur durch einen Blick auf den Inhalt der Bearbeitungszeile. In diesem Beispiel wird bei jeder Änderung der Zellmarkierung eine vorhandene Formel im Kommentar der Zelle angezeigt. Damit beim Löschen vorhandener Kommentare das Worksheet_SelectionChange-Ereignis nicht erneut aufgerufen wird, werden vorab alle Ereignisse über die EnableEvents-Eigenschaft deaktiviert. Wurde ein Bereich aus mehreren Zellen ausgewählt, wird die Markierung über die Select-Methode auf die erste Zelle der Auswahl reduziert.

```
Target.Cells(1, 1).Select
```

Die Prozedur würde sonst im weiteren Verlauf keinen eindeutigen Zellbezug für die Erstellung eines Kommentars herstellen können und eine Fehlermeldung hervorrufen. Damit bei einer Änderung der Auswahl über die Pfeiltasten der Tastatur ebenfalls vorhandene Formeln im Kommentartext angezeigt werden, stellen wir die Visible-Eigenschaft des Comment-Objektes auf True. Die Größe des Kommentarrahmens wird automatisiert.

```
'=====
' Auf CD      Buchdaten\Beispiele\09_Ereignisse\
' Dateiname   09_11_SelectionChange.xls
' Tabelle     FormulaInfo
' Modul       Tabelle7(FormulaInfo)
'=====
```

```
Private Sub Worksheet_SelectionChange(ByVal Target As Range)
    Dim cmt As Comment

    ' Ereignisse deaktivieren
    Application.EnableEvents = False

    ' evtl. vorhandene Kommentare löschen
    If Comments.Count > 0 Then
        UsedRange.SpecialCells(xlCellTypeComments).ClearComments
    End If

    ' mit erster Zelle einer Auswahl
    With Target.Cells(1, 1)
        .Select
        If .HasFormula Then
            Set cmt = .AddComment(.FormulaLocal)
            cmt.Visible = True
            cmt.Shape.TextFrame.AutoSize = True
        End If
    End With
End Sub
```

Grundlagen

Allgemein

Datum
Zeit

Formeln

Steuerelemente

Befehle
leisten

Objekte

Diagramme

Ereignisse

User-Formen

Web/Mail

Extern

Gemischtes

Spezial

End With

' Ereignisse aktivieren

Application.EnableEvents = True

End Sub

309 Bei Blattwechsel Monatsspalten anzeigen

Im folgenden Beispiel haben wir im Tabellenblatt CALENDAR einen Terminkalender erstellt. Neben jeder Monatsspalte befindet sich eine weitere Spalte für eigene Eintragungen. Wenn die Datei geöffnet wird, wird über das Workbook_Open-Ereignis das Tabellenblatt START aktiviert.

```
' =====
' Auf CD      Buchdaten\Beispiele\09_Ereignisse\
' Dateiname   09_12_MonthColumns.xls
' Ereignis    DieseArbeitsmappe
' =====
```

```
Private Sub Workbook_Open()
    Worksheets("Start").Select
End Sub
```

Über ein KONTROLLKÄSTCHEN auf dem Tabellenblatt START können Sie per Klick angeben, ob Sie beim Wechsel zu dem Tabellenblatt CALENDAR nur die relevanten Spalten des aktuellen Monats zur Ansicht erhalten möchten oder das gesamte Tabellenblatt angezeigt werden soll. Sobald Sie von einem anderen Tabellenblatt dieser Arbeitsmappe zu dem Tabellenblatt CALENDAR wechseln, wird das Worksheet_Aktiviere-Ereignis der Tabelle ausgelöst, welches im Klassenmodul »Tabelle2 (Calendar)« hinterlegt ist.

Als Erstes wird der Zustand des Kontrollkästchens abgefragt. Ist die CheckBox1 aktiviert, beträgt dessen Wert True und der gesamte Kalender soll angezeigt werden. Folglich werden alle bedeutsamen Spalten der Tabelle A:X eingeblendet. Wenn das Kontrollkästchen deaktiviert wurde, sollen nur die Spalten des derzeitigen Monats in der Tabelle angezeigt werden. Die CheckBox1 hat in dem Fall den Wert False. Daraufhin werden die relevanten Spalten der Tabelle A:X, welche in Zeile drei jeweils Monatsnamen in den Spaltenüberschriften enthalten, ausgeblendet. Das Schlüsselwort Me bezieht sich auf das aktive Tabellenblatt.

Der aktuelle Monat wird laut Systemeinstellung in der langen Schreibweise in die Variable strMonth gespeichert, so dass über die Find-Methode die Spalte des gesuchten Monats ermittelt werden kann. Diese wird in der Variablen intCol gespeichert und anschließend mitsamt der zu dem Monat gehörenden Nebenspalte eingeblendet.

Natürlich wollen Sie die jeweilige Monatsspalte nach Aktivierung des Tabellenblattes CALENDAR auch in Ihrem Sichtbereich haben. Würden Sie jetzt lediglich die Zelle A1 mit Range("A1").Select markieren, um zum Anfang der Tabelle zu springen, bliebe der Sichtbereich jedoch weiterhin auf der ursprünglichen Zelle, obwohl die Zelle A1 markiert ist. Über die GoTo-Methode wird ein angegebener Bereich aktiviert. Für diese Methode können Sie das optionale Argument Scroll angeben, um Bildläufe im Zielfenster durchzuführen. Geben Sie den Wert True an, führt Excel die nötigen Bildläufe durch, damit die obere linke Zelle des Bereichs als obere linke Zelle des Fensters erscheint. Geben Sie für das Argument den Wert False an, führt Excel keine Bildläufe durch. Der Standardwert ist False.

```
'=====
' Auf CD      Buchdaten\Beispiele\09_Ereignisse\
' Dateiname  09_12_MonthColumns.xls
' Tabelle    Calendar
' Ereignis   Tabelle2 (Calendar)
'=====

Private Sub Worksheet_Activate()
    Dim intCol As Byte
    Dim strMonth As String

    ' Bildschirmaktualisierung ausschalten
    Application.ScreenUpdating = False

    If Worksheets(1).CheckBox1.Value = True Then
        ' CheckBox "Gesamte Kalenderansicht" wurde aktiviert
        ' Alle Spalten des Bereichs einblenden
        Me.Columns("A:X").EntireColumn.Hidden = False
    Else
        ' CheckBox "Gesamte Kalenderansicht" wurde deaktiviert
        ' Alle Spalten des Bereichs ausblenden
        Me.Columns("A:X").EntireColumn.Hidden = True

        ' aktuellen Monat in langer Schreibweise auslesen
        strMonth = Format(Date, "MMMM")

        ' Spaltennummer des aktuellen Monats auslesen
        intCol = Me.Rows(3) _
            .Find(what:=strMonth, lookat:=xlValue).Column

        ' Aktuelle Monats- und Nebenspalte einblenden
        Me.Range(Columns(intCol), Columns(intCol + 1)) _
            .EntireColumn.Hidden = False
    End If

    ' Sichtbereich festlegen
    Application.Goto Me.Range("A1"), True

    ' Bildschirmaktualisierung einschalten
    Application.ScreenUpdating = True
End Sub
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

310 Blattschutz ohne Blattschutz

Sie können in einem Tabellenblatt Zellen und Bereiche sperren, ohne dafür einzelne Zellen zu sperren oder den Blattschutz zu aktivieren. Verwenden Sie dazu das Worksheet_SelectionChange-Ereignis des Tabellenblattes.

Beim Aufruf des Ereignisses wird die Adresse der markierten Zelle oder des Bereichs über die Variable Target an die Prozedur übergeben. Mit der Intersect-Methode wird anschließend die Schnittmenge des markierten und des erlaubten Bereichs Range("E10:I25") ermittelt. Der Variablen Target wird dieser neue Bereich zugewiesen. Wurden Zellen außerhalb des erlaubten Bereichs E10:I25 markiert, werden diese dadurch in der neu zugewiesenen Target-Variablen ignoriert.

In der äußeren If-Abfrage wird der Inhalt der Variablen `Target` überprüft. Enthält dieser keinen Wert, bedeutet das, dass keine Zelle im erlaubten Bereich markiert wurde. In der folgenden If-Abfrage wird nun ermittelt, ob die statische Variable `rngLastRange` einen Wert enthält. Die `Static`-Anweisung wird verwendet, um den Wert der Variablen `rngLastRange` während der gesamten Laufzeit des Moduls zu speichern.

Die Variable `rngLastRange` enthält noch keinen Wert, wenn die Ereignis-Prozedur das erste Mal durch eine Änderung der Zellmarkierung ausgelöst wird. Demzufolge wird die erste Zelle E10 im erlaubten Bereich selektiert. Die Variable `rngLastRange` erhält einen Wert, wenn eine Zelle bzw. Bereich im erlaubten Bereich markiert wurde. Diese Zelle wird dann statt der Zelle E10 markiert.

```
'=====
' Auf CD      Buchdaten\Beispiele\09_Ereignisse\
' Dateiname  09_13_SheetProtection.xls
' Tabelle    ProtectRange
' Ereignis   Tabelle1 (ProtectRange)
'=====
```

```
Private Sub Worksheet_SelectionChange(ByVal Target As Range)
    Static rngLastRange As Range
    Set Target = Intersect(Target, Range("E10:I25"))

    Application.EnableEvents = False
    If Target Is Nothing Then
        If rngLastRange Is Nothing Then
            Range("E10").Select
        Else
            rngLastRange.Select
        End If
    Else
        Target.Select
    End If

    Set rngLastRange = Selection
    Application.EnableEvents = True
End Sub
```

Neben einem zusammenhängenden Bereich können auch einzelne Zellen zur Markierung freigegeben werden. Auf diese Art können in einem Formular auch einzelne Zellen bearbeitet werden.

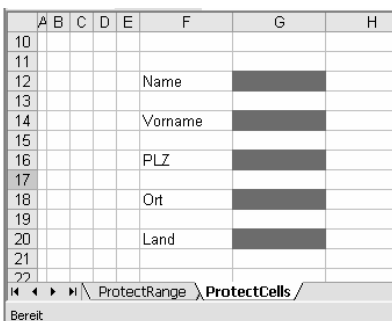


Abbildung 254: Formularzellen freigeben

```
'=====
' Auf CD      Buchdaten\Beispiele\09_Ereignisse\
' Dateiname   09_13_SheetProtection.xls
' Tabelle     ProtectCells
' Ereignis    Tabelle2 (ProtectCells)
'=====

Private Sub Worksheet_SelectionChange(ByVal Target As Range)
    Static rngLastRange As Range
    Set Target = Intersect(Target, Range("G12,G14,G16,G18,G20"))

    Application.EnableEvents = False
    If Target Is Nothing Then
        If rngLastRange Is Nothing Then
            Range("G12").Select
        Else
            rngLastRange.Select
        End If
    Else
        Target.Select
    End If

    Set rngLastRange = Selection
    Application.EnableEvents = True
End Sub
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

311 Ereignisbedingte Formatierung

In Excel ist es mit der bedingten Formatierung möglich, das Zellenformat entsprechend dem Zellenwert vorzugeben. Diese Option finden Sie im Menü **FORMAT | BEDINGTE FORMATIERUNG....** Es können jedoch nur maximal drei Bedingungen je Zelle festgelegt werden. Mit dem **Worksheet_Change**-Ereignis ist es möglich, beliebig viele Bedingungen mit entsprechenden Formatierungen für eine Zelle einzustellen.

In unserer Beispieldatei wird diese Art der bedingten Formatierung nur in einem Tabellenblatt verwendet. Aus diesem Grund schreiben wir den Code in das **Worksheet_Change**-Ereignis des Tabellen-Codeblattes. Wenn Sie dieses Beispiel in allen Tabellenblättern Ihrer Arbeitsmappe nutzen wollen, verwenden Sie das **Workbook_SheetChange**-Ereignis. Die Prozedur wird dazu in das Modul **DIESEARBEITSMAPPE** geschrieben.

In unserem Beispiel erhalten alle Zellen mit Zahlenwerten zwischen 1 und 90 einen farbigen Zellenhintergrund. Zum Beispiel werden Zellen mit Zahlenwerten zwischen 1 und 10 mit dem Farbindex 1 eingefärbt. In unserem Beispiel gehen wir davon aus, dass die Einstellungen der Farbpalette nicht geändert wurden.

Das **Worksheet_Change**-Ereignis wird ausgelöst, wenn sich der Inhalt einer oder mehrerer Zellen ändert. Der geänderte Bereich wird über die Variable **Target** an die Ereignis-Prozedur übergeben. Über die **Count**-Eigenschaft wird ermittelt, wie viele Zellen markiert wurden. Beträgt dessen Wert größer 256, wird davon ausgegangen, dass mehrere Zeilen oder Spalten markiert wurden. Der Auslöser kann zum Beispiel das Einfügen oder Löschen von Spalten oder Zeilen gewesen sein. In diesem Fall wird die Prozedur vorzeitig beendet, um einen Überlauf des Speichers und somit eine Fehlermeldung zu vermeiden.

Hinweis

Wenn Sie im MENÜ EXTRAS | OPTIONEN | FARBE die Standardeinstellungen der Farbpalette nicht verändert haben, entspricht der Farbindex 1 der Farbe Schwarz. Die Farbe Weiß hat den Standardindex 2. Anstelle der Farbindexe können Sie auch die entsprechenden RGB-Werte einsetzen. Die Farbe Schwarz hat zum Beispiel den RGB-Wert (0,0,0). Mit der RGB-Funktion stellen Sie sicher, dass die gewünschte Zellenfärbung, unabhängig von den Einstellungen der Farbpalette, angewandt wird.

Eine Übersicht der RGB-Standardfarben finden Sie im Anhang. In der Kategorie »Einstieg« finden Sie ebenfalls ein Beispiel und Erläuterungen zu RGB-Farben.

Im Tabellenblatt sollen lediglich die Zellen im Bereich A1:D20 nach ihrem jeweiligen Zahlenwert farbig formatiert werden. Die `Intersect`-Methode gibt ein `Range`-Objekt zurück, das die rechteckige Schnittmenge von zwei oder mehreren Bereichen darstellt. In unserem Beispiel setzt sich die Schnittmenge aus dem geänderten Bereich `Target` und dem zu überwachenden Bereich `Range("A1:D20")` zusammen. Wurden mehrere Zellen in einem Durchgang geändert (zum Beispiel durch `Copy/Paste`), werden nur die Zellen formatiert, welche innerhalb des angegebenen Bereichs liegen.

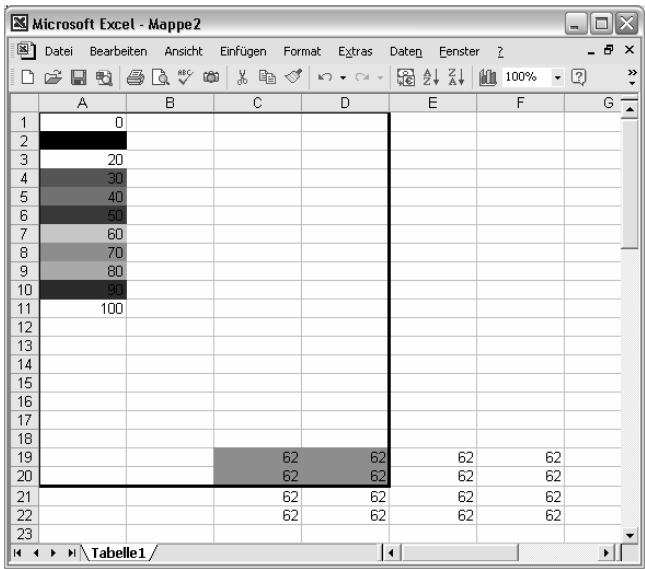


Abbildung 255: Ereignisbedingte Formatierung mit mehr als drei Bedingungen

In einer `For ... Each`-Schleife wird jede Zelle im Bereich `Target` überprüft. Wird der Wert der Zelle `rngCell` in der folgenden `Select Case`-Anweisung gefunden, wird der Farbindex an die Variable `bytColor` übergeben und die `Select Case`-Anweisung verlassen. Anschließend erhält die Zelle `rngCell` die Füllfarbe über den angegebenen Farbindex der Variablen `bytColor`. Enthält die Zelle keinen angegebenen Zahlen- oder Textwert, wird eine eventuell vorhandene Füllfarbe der Zelle gelöscht, indem der Farbindex 0 angewandt wird.

```
'=====
' Auf CD      Buchdaten\Beispiele\09_Ereignisse\
' Dateiname  09_14_EventFormatConditional.xls
' Ereignis   Tabelle1
```

```
'=====

Private Sub Worksheet_Change(ByVal Target As Range)
    Dim rngCell As Range
    Dim bytColor As Byte

    If Target.Count > 256 Then Exit Sub

    ' Bereich, der überwacht wird
    Set Target = Intersect(Target, Range("A1:D20"))
    If Target Is Nothing Then Exit Sub

    For Each rngCell In Target
        Select Case rngCell.Value
            Case 1 To 10
                bytColor = 1 ' Schwarz
            Case 11 To 20
                bytColor = 2 ' Weiss
            Case 21 To 30
                bytColor = 3 ' Rot
            Case 31 To 40
                bytColor = 4 ' Grün
            Case 41 To 50
                bytColor = 5 ' Blau
            Case 51 To 60
                bytColor = 6 ' Gelb
            Case 61 To 70
                bytColor = 7 ' Rosa
            Case 71 To 80
                bytColor = 8 ' Cyan
            Case 81 To 90
                bytColor = 9 ' Braun
            Case Else
                bytColor = 0 ' keine Farbe
        End Select

        rngCell.Interior.ColorIndex = bytColor
    Next rngCell
End Sub
```

Grundlagen
Allgemein
Datum und Zeit
Formeln
Diagramme
Steuerungselemente
Befehlsliste
Objekte
Diagramme
Ereignisse
User-Formen
Web/Mail
Extern
Gemischtes
Spezial

312 Ereignisbedingte Formatierung durch Formelbezüge

Die ereignisbedingte Formatierung aus dem vorherigen Beispiel funktioniert nur bei direkter Eingabe in dem Tabellenblatt, in dessen Codemodul die Prozedur hinterlegt ist. Werden die Werte durch Formelbezüge berechnet, wird das `Worksheet_Change`-Ereignis nur in dem Tabellenblatt ausgelöst, in welchem die Quelldaten für die Formelberechnungen geändert werden. Dementsprechend muss zusätzlich das Tabellenblatt der Quelldaten über dessen `Worksheet_Change`-Ereignis auf Änderungen überwacht werden.

In unserem Beispiel befinden sich die Quelldaten in dem Tabellenblatt `DataSource`. Die Zellen im Tabellenblatt `FormatConditional` beinhalten lediglich einen Formelbezug auf die Quelldaten. Die Abbildung 256 verdeutlicht die Situation. Sie zeigt beide Tabellenblätter der Beispieldatei neben-

einander in einer Excel-Anwendung an. Diese Anordnung erreichen Sie über das Menü FENSTER | NEUES FENSTER. Wenn Sie dann das aktive Fenster verkleinern, wird das neue Fenster der Arbeitsmappe in der Excel-Anwendung sichtbar. Wählen Sie das zweite Tabellenblatt durch Klicken auf den Registerreiter aus und ziehen Sie mit der Maus beide Fenster auf eine optimale Größe und Position. Auf diese Weise können Sie im Tabellenblatt DataSource Änderungen vornehmen und gleichzeitig im Tabellenblatt FormatConditional die Auswirkungen der ereignisbedingten Formatierung sehen.

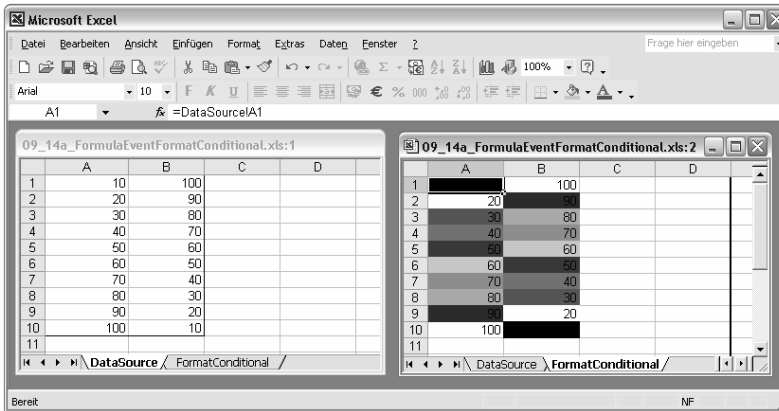


Abbildung 256: Ereignisbedingte Formatierung mit Formelbezügen

```
'=====
' Auf CD      Buchdaten\Beispiele\09_Ereignisse\
' Dateiname   09_14a_EventFormatConditional.xls
' Ereignis    Tabelle1 (DataSource)
'=====
```

```
Private Sub Worksheet_Change(ByVal Target As Range)
    Dim rngFormCond As Range
    Dim rngCell As Range
    Dim bytColor As Byte

    If Target.Count > 256 Then Exit Sub

    ' Bereich, der überwacht wird (Quelldaten für Formelbezüge)
    Set Target = Intersect(Target, Range("A1:B10"))
    If Target Is Nothing Then Exit Sub

    ' Bereich der bedingten Formatierung
    Set rngFormCond = Worksheets("FormatConditional").Range("A1:D20")

    ' Bereich, der formatiert werden soll
    For Each rngCell In rngFormCond
        Select Case rngCell.Value
            Case 1 To 10
                bytColor = 1 ' Schwarz
            Case 11 To 20
                bytColor = 2 ' Weiss
```



```
Case 21 To 30
    bytColor = 3 ' Rot
Case 31 To 40
    bytColor = 4 ' Grün
Case 41 To 50
    bytColor = 5 ' Blau
Case 51 To 60
    bytColor = 6 ' Gelb
Case 61 To 70
    bytColor = 7 ' Rosa
Case 71 To 80
    bytColor = 8 ' Cyan
Case 81 To 90
    bytColor = 9 ' Braun
Case Else
    bytColor = 0 ' keine Farbe
End Select

rngCell.Interior.ColorIndex = bytColor
Next rngCell
End Sub
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

313 Makro aus Hyperlink aufrufen

Makros lassen sich in Tabellenblättern auch über Hyperlinks aufrufen. Zeigen Sie über das Menü EINFÜGEN | HYPERLINK oder die Tastenkombination **[Strg] + [K]** den HYPERLINK-BEARBEITEN-Dialog an. Legen Sie dann den Zelltext »MyMessage«, welcher zugleich der Name der aufzurufen- den Prozedur ist, fest. Da der Hyperlink auf kein Objekt verweist, geben Sie als Hyperlink-Adresse die Zelladresse an, in welcher der Hyperlink hinterlegt ist. Bestätigen Sie Ihre Eingabe mit einem Klick auf die Schaltfläche OK.

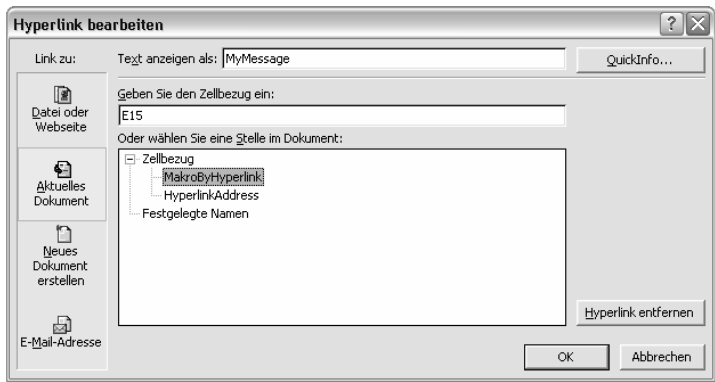


Abbildung 257: Zelltext und Hyperlinkadresse festlegen

Bei einem Klick auf einen Hyperlink wird das Worksheet_FollowHyperlink-Ereignis ausgelöst. In der folgenden Ereignis-Prozedur muss zuerst überprüft werden, ob sich der Zellbezug, auf den der Hyperlink verweist, auf die eigene Zellposition bezieht. Dadurch erfolgt die Unterscheidung, ob der Hyperlink ein Makro aufruft, oder auf einen Zellbezug verweist.

Um die Zelladresse, in welcher der Hyperlink hinterlegt ist, zu ermitteln, verwenden wir die Sub-Address-Eigenschaft. Diese gibt die Position innerhalb des mit dem Hyperlink verbundenen Dokuments zurück. Die ermittelte Zeichenkette enthält den Blattnamen mit der Zelladresse, auf den der Hyperlink verweist. Über die Excel-Tabellenfunktionen Mid und der InStr-Funktion wird die Zelladresse separiert und an die Variable strHyp übergeben.

In der folgenden If-Anweisung wird der auslösende Zellbezug mit dem Zellverweis verglichen. Sind diese beiden Zellbezüge identisch, wurde in unserem Beispiel dem Hyperlink eine Prozedur anstelle eines Verweises zugewiesen.

Die TextToDisplay-Eigenschaft gibt den angezeigten Text des auslösenden Hyperlinks zurück. Diese Zeichenfolge entspricht dem Makronamen einer in einem allgemeinen Modul hinterlegten Prozedur. Über die Run-Methode wird diese Prozedur aufgerufen. Die On Error GoTo-Anweisung verzweigt zu der Sprungmarke ERRORHANDLER, wenn die Prozedur nicht gefunden wird.

```
'=====
' Auf CD      Buchdaten\Beispiele\09_Ereignisse\
' Dateiname  09_15_FollowHyperlink.xls
' Ereignis   Tabelle1 (MakroByHyperlink)
'=====
```

```
Private Sub Worksheet_FollowHyperlink(ByVal Target As Hyperlink)
    Dim strHyp As String
```

```
    ' Zelladresse aus Zellenbezug (Blattname!Adresse)
    ' extrahieren
    strHyp = _
    Mid(Target.SubAddress, InStr(Target.SubAddress, "!") + 1, 7)
```

```
    If Target.Parent.Address(0, 0) = strHyp Then
        On Error GoTo ERRORHANDLER
        Application.Run Target.TextToDisplay
    Exit Sub
```

```
ERRORHANDLER:
    MsgBox _
        "Die Prozedur '" & Target.TextToDisplay & _
        "' wurde nicht gefunden", _
        vbExclamation
End If
End Sub
```

In der Beispieldatei wurden in drei Zellen Hyperlinks mit Makroaufrufen eingefügt. Klicken Sie auf einen dieser Hyperlinks, um die jeweils gleichnamige Prozedur wie der Zellenwert aufzurufen. Der vierte Hyperlink ruft kein Makro auf, sondern verweist auf das Tabellenblatt MAKROBYHYPERLINK.

```
'=====
' Auf CD      Buchdaten\Beispiele\09_Ereignisse\
' Dateiname  09_15_FollowHyperlink.xls
' Modul      mdl_01_HyperlinkMacros
'=====
```

```
Sub BlueColor()  
    Range("A14:I16").Interior.ColorIndex = 37  
End Sub  
  
Sub MyMessage()  
    MsgBox "Hallo " & Application.UserName  
End Sub  
  
Sub ShowTheTime()  
    MsgBox "Es ist jetzt " & Time & " Uhr"  
End Sub
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

314 Adresse zu Hyperlink ermitteln

Beim Aufruf der Worksheet_FollowHyperlink-Ereignisprozedur wird die Variable Target übergeben. Diese enthält alle Informationen, die den Hyperlink betreffen. Allerdings liefert Target nicht direkt das Objekt, aus welchem der Hyperlink aufgerufen wurde. Wenn der Hyperlink in einer Zelle eingefügt wurde, gibt die Parent-Eigenschaft das übergeordnete Range-Objekt des Target-Objekts, also die Zelle, in welcher der Hyperlink hinterlegt ist, zurück. Über die Address-Eigenschaft des Range-Objektes wird die Zelladresse ermittelt. Der Tabellenblattname wird über das Schlüsselwort Me ermittelt. Me bezieht sich auf die Klasse, in welcher der Code ausgeführt wird. Die Name-Eigenschaft gibt den Namen des Me-Objekts, also den Tabellenblattnamen zurück, in dem die Prozedur hinterlegt ist. In einem Meldungsfenster wird der Tabellenblattname mit dem auslösenden Zellbezug zu einer Zeichenkette verbunden.

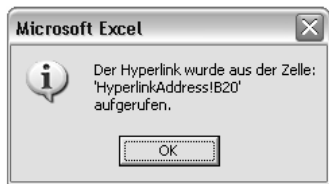


Abbildung 258: Zelladresse des auslösenden Hyperlinks

```
'=====
' Auf CD      Buchdaten\Beispiele\09_Ereignisse\
' Dateiname  09_15_FollowHyperlink.xls
' Ereignis   Tabelle2 (HyperlinkAddress)
'=====

Private Sub Worksheet_FollowHyperlink(ByVal Target As Hyperlink)

    MsgBox "Der Hyperlink wurde aus der Zelle:" & _
        vbCr & "" & _
        Me.Name & "!" & _
        Target.Parent.Address(0, 0) & _
        "" & vbCr & _
        "aufgerufen.", _
        vbInformation

End Sub
```

315 Autoprozeduren

Vielleicht haben Sie schon einmal Prozeduren gesehen, die mit

```
Sub Auto_Open()
```

oder

```
Sub Auto_Close()
```

beginnen. Dies sind so genannte »Autoprozeduren«. Seit Einführung von Ereignis-Prozeduren in Excel 97 stehen sie in neueren Excel-Versionen nur noch aus Kompatibilitätsgründen zur Verfügung. Im Gegensatz zu den Ereignis-Prozeduren werden sie nicht in einem Klassenmodul, sondern in einem beliebigen allgemeinen Modul gespeichert.

Üblicherweise werden in der derzeitigen VBA-Programmierung die analogen Ereignisse `Workbook_Open` bzw. `_Close` des `Workbook`-Objektes verwendet. Da aber noch ältere Dateien mit diesen Makros ausgestattet sind und Autoprozeduren bis nachweislich Excel 2003 funktionsfähig sind, stellen wir Ihnen hier ein Beispiel dazu vor.

In Rezept 290 haben wir Sie darauf hingewiesen, dass in einer Arbeitsdatei nur ein gleichnamiges Ereignis im Klassenmodul `DIESEARBEITSMAPPE` hinterlegt werden darf. Ebenso verhält es sich mit den Autoprozeduren. Sind zum Beispiel mehrere `Auto_Open`-Prozeduren in einer Arbeitsdatei gespeichert, erfolgt beim Öffnen der Datei eine Fehlermeldung und keine der hinterlegten Prozeduren wird ausgeführt. Dabei ist es gleichgültig, ob die Prozeduren in einem oder in verschiedenen Modulen gespeichert sind.

Wenn in einer Excel-Datei eine Prozedur mit `Auto_Open` bzw. `Auto_Close` existiert, werden diese Prozeduren automatisch beim Öffnen oder Schließen der Datei ausgeführt.

Hinweis

Wenn das Laden oder Schließen einer Excel-Datei durch VBA-Code ausgelöst wird, werden Autoprozeduren nicht automatisch ausgeführt. Führen Sie die `RunAutoMacros`-Methode für die betreffende Arbeitsmappe aus, um die Prozeduren dennoch auszuführen.

In der Beispieldatei `09_16a_AutoMacros.xls` befinden sich zwei Schaltflächen, über welche die Testdatei `09_16b_TestAutoMacros.xls` geöffnet und geschlossen werden kann. Diese Testdatei enthält ein `Auto_Open`- und ein `Auto_Close`-Ereignis.

```
'=====
' Auf CD      Buchdaten\Beispiele\09_Ereignisse\
' Dateiname  09_16b_TestAutoMacros.xls
' Modul      mdl_01_AutoMacros
'=====
```

```
Sub Auto_Open()
    MsgBox _
        "Die Datei:" & vbNewLine & _
        ThisWorkbook.Name & _
        vbNewLine & "wird geöffnet."
End Sub
```

```

Sub Auto_Close()
    MsgBox _
        "Die Datei:" & vbNewLine & _
        ThisWorkbook.Name & _
        vbNewLine & "wird geschlossen."
End Sub

```

316 Datei öffnen und Auto_Open-Prozedur aufrufen

In der Beispieldatei soll die Testdatei im Hintergrund geöffnet, also nicht angezeigt werden. Dementsprechend stellen wir die Bildschirmaktualisierung `ScreenUpdating` zunächst auf `False`. Anschließend wird die Testdatei über die `Open`-Methode geöffnet und die `Auto_Open`-Prozedur der Testdatei über die `RunAutoMacros`-Methode aufgerufen. Bevor die Bildschirmaktualisierung wieder eingeschaltet wird, muss die aufrufende Datei aktiviert werden, da sonst nach Beendigung der Prozedur die Testdatei angezeigt würde.

```

' =====
' Auf CD      Buchdaten\Beispiele\09_Ereignisse\
' Dateiname  09_16a_AutoMacros.xls
' Modul      mdl_01_WorkbookOpen
' =====

Sub WorkbookOpenWithAutoMacros()
    Dim strWb As String

    ' Pfad und Name der Testdatei
    strWb = _
        ThisWorkbook.Path & "\09_16b_TestAutoMacros.xls"

    Application.ScreenUpdating = False
    Workbooks.Open strWb

    ' Auto_Open-Prozedur der Testdatei aufrufen
    ActiveWorkbook.RunAutoMacros xlAutoOpen

    ThisWorkbook.Activate
    Application.ScreenUpdating = True
End Sub

```

317 Datei schließen und Auto_Close-Prozedur aufrufen

Um eine Datei zu schließen, muss vorab überprüft werden, ob diese überhaupt geöffnet ist, da sonst eine Fehlermeldung ausgegeben wird. In einer `For Each`-Schleife werden die Namen aller geöffneten Arbeitsmappen mit dem Namen der zu öffnenden Datei verglichen. Wird die Datei gefunden, wird die der Testdatei zugeordnete `Auto_Close`-Prozedur ausgeführt. Im Anschluss daran wird die Testdatei geschlossen und die Prozedur verlassen.

```

' =====
' Auf CD      Buchdaten\Beispiele\09_Ereignisse\
' Dateiname  09_16a_AutoMacros.xls
' Modul      mdl_02_WorkbookClose

```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

```
' =====

Sub WorkbookCloseWithAutoMacros()
    Dim strWb As String
    Dim Wb As Workbook

    ' Name der Testdatei
    strWb = "09_16b_TestAutoMacros.xls"

    Application.ScreenUpdating = False

    For Each Wb In Application.Workbooks
        If Wb.Name = strWb Then
            With Workbooks(strWb)
                .RunAutoMacros xlAutoClose
                .Close
            End With

            Exit Sub
        End If
    Next

    MsgBox _
        "Die Datei:" & vbNewLine & _
        strWb & vbNewLine & _
        "ist nicht geöffnet.", _
        vbExclamation

    Application.ScreenUpdating = True
End Sub
```

318 Begrüßung nach der Tageszeit

Wie im oberen Abschnitt bereits erwähnt, darf in einer Arbeitsdatei jeweils nur eine gleichnamige Ereignis- bzw. Autoprozedur gespeichert sein. Es ist aber durchaus möglich, in »einer« Excel-Datei ein `Workbook_Open` *und* ein `Auto_Open`-Makro zu hinterlegen. In unserer Beispieldatei haben wir beide Möglichkeiten, ein Makro bei Dateioffnung zu starten, hinterlegt.

Als Erstes wird das `Workbook_Open`-Ereignis ausgelöst. Nach dessen Abarbeitung erfolgt der Aufruf der `Auto_Open`-Prozedur. Entsprechend dieser Abfolge gehen wir in der Beschreibung nacheinander vor.

Beim Öffnen der Datei wird die aktuelle Uhrzeit aus der Systemsteuerung in einer `Select...Case`-Anweisung überprüft. Entspricht ein `Case Is`-Wert der aktuellen Uhrzeit, wird der hinterlegte Text in einem Meldungsfenster ausgegeben.

```
' =====
' Auf CD      Buchdaten\Beispiele\09_Ereignisse\
' Dateiname   09_17a_AutoOpenZoom.xls
' Ereignis    DieseArbeitsmappe
' =====
```

```
Private Sub Workbook_Open()  
    Dim strText As String  
  
    Select Case Hour(Time)  
        Case Is < 5  
            strText = "Ist da ein Nachtschwärmer unterwegs?"  
        Case Is < 8  
            strText = "Heute bist Du ja ein Frühaufsteher"  
        Case Is < 10  
            strText = "Guten Morgen"  
        Case Is < 14  
            strText = "Einen schönen Tag wünsch ich Dir"  
        Case Is < 17  
            strText = "Einen schönen Nachmittag wünsch ich Dir"  
        Case Is < 24  
            strText = "Einen schönen guten Abend"  
    End Select  
  
    MsgBox "Es ist " & Time & " Uhr" & _  
        vbNewLine & _  
        vbNewLine & _  
        strText  
End Sub
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

319 Zoomfaktor über Bildschirmauflösung

In einem größeren Unternehmen wird über ein Netzwerk oft von mehreren Anwendern auf die gleiche Datei zugegriffen. Wenn die einzelnen Benutzer unterschiedliche Bildschirmeinstellungen verwenden, wird die Tabelle teilweise zu groß oder zu klein dargestellt. Um eine individuelle Ansicht zu erhalten, stellt jeder Benutzer dieser Datei den Zoomfaktor über den Menüpunkt ANSICHT | ZOOM oder dem entsprechende Kombinationsfeld aus der Standard-Symbolleiste auf seine Bildschirmeinstellung ein.

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

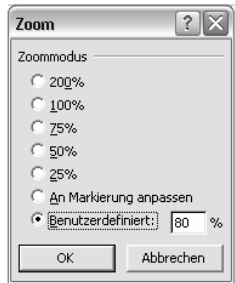


Abbildung 259: Zoom-Einstellung

Diese umständliche, manchmal auch Geduld erfordernde Handhabung können Sie sich von Excel abnehmen lassen.

Bei Dateiöffnung wird über eine Auto_Open-Prozedur die aktuelle Bildschirmeinstellung der jeweiligen Arbeitsstation, an welcher die Arbeitsmappe geöffnet wurde, aus der SYSTEMSTEUERUNG | ANZEIGE | EINSTELLUNGEN ausgelesen. Über eine Select...Case-Schleife wird entsprechend der ausgelesenen Bildschirmbreite der im Code definierte Zoomfaktor an die Variable

Zoomfactor übergeben. Nachdem der Zoomfaktor eingestellt wurde, informiert Sie ein Meldungsfenster über die aktuellen Bildschirmeinstellungen sowie den eingestellten Zoomfaktor. In der Praxis können Sie auf das Meldungsfenster verzichten. Durch ein vorangestelltes Hochkomma ' in der ersten Zeile des MsgBox-Codes wird dieser auskommentiert und beim Makrodurchlauf nicht abgearbeitet. Sie können die entsprechenden Zeilen der MsgBox auch aus dem Code löschen. Die Funktionalität der Zoom-Einstellung bleibt davon unberührt.

```
' =====
' Auf CD      Buchdaten\Beispiele\09_Ereignisse\
' Dateiname  09_17a_AutoOpenZoom.xls
' Modul      mdl_01_AutoOpenZoom
' =====

Declare Function GetSystemMetrics Lib "user32" _
    (ByVal nIndex As Long) As Long

Public Const SM_CXSCREEN = 0
Public Const SM_CYSCREEN = 1

Sub Auto_Open()
    Dim lngWidth As Long
    Dim lngHeigh As Long
    Dim Zoomfactor As Integer

    lngWidth = GetSystemMetrics(SM_CXSCREEN)
    lngHeigh = GetSystemMetrics(SM_CYSCREEN)

    Select Case lngWidth
        Case Is >= 800
            Zoomfactor = 80
        Case Is >= 1024
            Zoomfactor = 100
        Case Is >= 800
            Zoomfactor = 80
        Case Is >= 1280
            Zoomfactor = 120
    End Select

    Application.ActiveWindow.Zoom = Zoomfactor

    MsgBox "Bildschirmgröße: " & _
        lngWidth & " x " & lngHeigh & _
        vbCr & vbCr & _
        "Zoomfaktor      : " & Zoomfactor & " %"

End Sub
```

320 Zoomfaktor mit Workbook_Open

Eine praktische Verwendung der Zoomeinstellung anhand der Bildschirmeinstellung findet sich auch am heimischen PC. Hinterlegen Sie die Prozeduren aus der Beispielmappe in der Standardarbeitsmappe *Mappe.xlt*. Diese ist im Ordner XLSTART gespeichert und dient als Vorlage, wenn eine neue Datei erstellt wird. Die Datei *Mappe.xlt* ist mit der Datei *Normal.dot* aus MS Word ver-

gleichbar. Legen Sie eine neue Datei an, wird automatisch der passende Zoomfaktor zur aktuellen BildschirmEinstellung eingestellt. Wird die Datei auf einem anderen Rechner mit anderen BildschirmEinstellungen geöffnet, erfolgt ebenfalls eine Anpassung des Zoomfaktors.

Unter Windows 98 finden Sie die Datei *Mappe.xlt* normalerweise unter:

C:\Programme\Microsoft Office\Office\XLStart.

Haben Sie Excel unter Windows XP installiert, befindet sich der XLStart-Ordner im Regelfall im Speicherpfad:

C:\Dokumente und Einstellungen\Benutzername\Anwendungsdaten\Microsoft\Excel\XLStart.

Das Makro aus dem Abschnitt 319 können Sie nicht eins zu eins in das Klassenmodul DIESEARBEITSMAPPE kopieren, um es in einer Workbook_Open-Prozedur anzuwenden. Die Deklaration der im Makro verwendeten DLL-Funktion *user32* sowie die öffentlich deklarierten Konstanten *SM_CXSCREEN* und *SM_CYSCREEN* müssen in einem allgemeinen Modul gespeichert werden.

In der Beispielmappe hinterlegen wir die benötigten Deklarierungen zur Zoomeinstellung sowie das Makro *ZoomControl* in einem allgemeinen Modul. Das Makro *ZoomControl* wird im *Workbook_Open*-Ereignis aufgerufen.

Damit auch ein neu eingefügtes Tabellenblatt den korrekten Zoomfaktor erhält, wird im *Workbook_NewSheet*-Ereignis ebenfalls der Makro-Aufruf *ZoomControl* hinterlegt.

```
'=====
' Auf CD      Buchdaten\Beispiele\09_Ereignisse\
' Dateiname   09_17b_WorkbookOpenZoom.xls
' Ereignis     DieseArbeitsmappe
'=====
```

```
Private Sub Workbook_Open()
    ZoomControl
End Sub
```

```
Private Sub Workbook_NewSheet(ByVal Sh As Object)
    ZoomControl
End Sub
```

```
'=====
' Auf CD      Buchdaten\Beispiele\09_Ereignisse\
' Dateiname   09_17b_WorkbookOpenZoom.xls
' Modul       mdl_01_Zoom
'=====
```

```
Declare Function GetSystemMetrics Lib "user32" _
    (ByVal nIndex As Long) As Long
```

```
Public Const SM_CXSCREEN = 0
Public Const SM_CYSCREEN = 1
```

```
Sub ZoomControl()
    Dim lngWidth
    Dim lngHeigh
    Dim Zoomfactor
```

Grundlagen

Allgemein

Datenzeit

Funktionen

Steuerelemente

Befehle

Objekte

Diagramme

Ereignisse

User-Formen

Web/Mail

Extern

Gemischt

Spezial

```
lngWidth = GetSystemMetrics(SM_CXSCREEN)
lngHeigh = GetSystemMetrics(SM_CYSCREEN)
```

```
Select Case lngWidth
    Case Is >= 800
        Zoomfactor = 80
    Case Is >= 1024
        Zoomfactor = 100
    Case Is >= 800
        Zoomfactor = 80
    Case Is >= 1280
        Zoomfactor = 120
End Select
```

```
Application.ActiveWindow.Zoom = Zoomfactor
End Sub
```

321 Zeitgesteuerte Makros

Wollen Sie ein Makro täglich zu einer bestimmten Zeit automatisch aufrufen, verwenden Sie die `OnTime`-Methode, welche zu den Zeitereignissen zählt. Diese Methode bewirkt, dass eine Prozedur zu einem bestimmten Zeitpunkt in der Zukunft ausgeführt wird, oder nachdem eine bestimmte Zeitspanne vergangen ist.

Die Syntax der `OnTime`-Methode mit ihren verwendbaren Parametern lautet:

```
OnTime(EarliestTime, Procedure, LatestTime, Schedule)
```

Die ersten beiden Parameter sind zwingend anzugeben. Parameter drei und vier sind optional, das bedeutet, dass sie nicht angegeben werden müssen. Enthalten sie aber einen Wert, wird dieser bei der Verarbeitung der Methode angewandt.

Im ersten Beispiel wird zehn Sekunden, nachdem das Makro `OnTime1` aufgerufen wurde, das Makro `myMakro1` ausgeführt.

```
'=====
' Auf CD      Buchdaten\Beispiele\09_Ereignisse\
' Dateiname   09_18_OnTime.xls
' Modul       mdl_01_OnTimeMakros
'=====
```

```
Sub OnTime1()
    Application.OnTime Now + TimeValue("00:00:10"), "myMakro1"
End Sub
```

```
Sub myMakro1()
    MsgBox "Hallo " & Application.UserName & _
        vbNewLine & vbNewLine & _
        "hier kommt Makro1"
End Sub
```

Makros können zu einer vorgegebenen Zeit ausgeführt werden. Die Angabe eines künftigen Datums ist mit der `OnTime`-Methode nicht möglich.

```
'=====
' Auf CD      Buchdaten\Beispiele\09_Ereignisse\
' Dateiname  09_18_OnTime.xls
' Modul      mdl_01_OnTimeMakros
'=====

Sub OnTime2()
    Application.OnTime TimeValue("10:15:00"), "myMakro2"
End Sub

Sub myMakro2()
    MsgBox "Hallo " & Application.UserName & _
        vbCrLf & vbCrLf & _
        "hier kommt Makro2"
End Sub
```

Die Zeit, zu der das Makro OnTime3 ausgeführt werden soll, steht in der benannten Zelle MyTime des Tabellenblattes und wird beim Durchlaufen der Codezeilen in die Prozedur eingelesen. Dadurch muss bei einer Änderung der Aufrufzeit nicht der Code geändert werden, sondern lediglich die Zeitangabe in der Zelle.

```
'=====
' Auf CD      Buchdaten\Beispiele\09_Ereignisse\
' Dateiname  09_18_OnTime.xls
' Modul      mdl_01_OnTimeMakros
'=====

Sub OnTime3()
    Dim MyTime

    MyTime = Format(Range("MyTime").Value, "hh:mm:ss")
    Application.OnTime TimeValue(MyTime), "myMakro3"
End Sub

Sub myMakro3()
    MsgBox "Hallo " & Application.UserName & _
        vbCrLf & vbCrLf & _
        "hier kommt Makro3"
End Sub
```

322 Uhrzeit in Statuszeile zeitbegrenzt anzeigen

Bei den bisherigen Beispielen zur OnTime-Methode wurden nur die ersten beiden, zwingend anzugebenden, Parameter verwendet: die »Startzeit« und der »Name« des aufgerufenen Makros. Im vierten Parameter LatestTime kann der späteste Zeitpunkt angegeben werden, zu dem die angegebene Prozedur gestartet werden soll.

Um einen periodischen Aufruf einer Prozedur zu erreichen, setzen Sie die Startzeit des aufgerufenen Makros aus der aktuellen Uhrzeit plus einer angegebenen Zeitspanne zusammen und geben den Makronamen der eigenen Prozedur als Aufrufmakro an.

Grundlagen

Allgemein

Datum und Zeit

Funktionen

Steuerelemente

Befehle und Methoden

Objekte

Diagramme

Ereignisse

User-Formen

Web/Mail

Extern

Gemischtes

Spezial

```

Sub Test()
    MsgBox "Test"
    Application.OnTime Now + TimeValue("00:00:10"), "Test"
End Sub

```

Der vorgenannte Code würde ohne weitere Anweisungen in einer Endlosschleife enden. Sie können ihn im VBA-Editor auf die harte Tour beenden, indem Sie zum Beispiel den Namen der Prozedur ändern, was eine Fehlermeldung bewirkt, da das aufgerufene Makro nicht gefunden wird. Löschen Sie während der Laufzeit der Endlosschleife die `OnTime`-Anweisung im Code, erfolgt ebenfalls ein Abbruch. Wie Sie sehen, ist die Beendigung einer Zeitschleife nicht ganz einfach, aber nicht unmöglich. Aus diesem Grund beschreiben wir die Vorgehensweise entsprechend ausführlich.

Der späteste Zeitpunkt und somit die Beendigung der vormaligen Endlosschleife wird über den Parameter `LatestTime` angegeben. Dadurch erreichen Sie, dass die einmal angestoßene Zeitschleife nicht unendlich weiterläuft.

In der Prozedur `StatusBarTime1` wird das aktuelle Datum sowie die Uhrzeit alle zehn Sekunden als Text in der Statuszeile angezeigt. Für den spätesten Zeitpunkt speichern wir die Startzeit plus eine Minute in der Variablen `StopTime`, damit die entstandene Zeitschleife nach einer Minute beendet und keine Endlosschleife erzeugt wird.

Hier ergibt sich das Problem, dass die Variable `StopTime` jeweils zu Beginn des Makros einen neuen Wert annehmen würde, nämlich die aktuelle Systemzeit plus eine Minute und somit wären wir wieder in einer Endlosschleife. Um dieses Problem auszumerzen, verwenden wir die Variable `Bo1Time` vom Datentyp `Boolean`. Standardmäßig hat eine Variable dieses Datentyps den Wert `False`. Diese Wahrheitsprüfung setzen wir in einer `If...Then`-Anweisung ein. Da die Variable `Bo1Time` beim ersten Makrodurchlauf noch den Wert `False` hat, wird diese Schleife zunächst abgearbeitet und der späteste Zeitpunkt des Makros in die Variable `StopTime` geschrieben. Gleich im Anschluss daran wird `Bo1Time` auf `True` gesetzt, damit beim erneuten Durchlauf des Makros diese `If...Then`-Schleife nicht noch einmal durchlaufen wird.

Die Zeitangaben werden nun bei jedem Makrodurchlauf über die `StatusBar`-Eigenschaft in der Statuszeile angezeigt. Im Anschluss daran kommt die `OnTime`-Methode zum Zuge und meldet das Makro für die nächste Zeitanzeige an. Bei Erreichen der Endzeit muss die Verwaltung der Statuszeile wieder an Excel zurückgegeben werden. Die zuletzt geschriebene Zeit würde sonst in der Statuszeile stehen bleiben. Dazu wird der letztmalige Zeitpunkt des Aufrufzeitpunkts `ETime` mit der Endzeit `StopTime` verglichen. Ist die Endzeit größer, wird die Statuszeile zurückgesetzt sowie die Variable `Bo1Time` auf `False` gesetzt. Bei einem erneuten Makroaufruf wird dadurch die korrekte Zuweisung der Endzeit wieder ermöglicht.

Die verwendeten Variablen müssen vor dem `Sub`-Aufruf deklariert werden, damit sie bei erneutem Schleifendurchlauf ihren gespeicherten Wert behalten und im Makro korrekt abgearbeitet werden können.

Da die Variable `Bo1Time` sowohl in einem Modul als auch im `Workbook_Close`-Ereignis verwendet wird, muss sie durch die vorangestellte `Public`-Anweisung global deklariert werden. Sie kann dadurch aus allen Modulen des VBA-Projektes angesprochen werden.

```

'=====
' Auf CD      Buchdaten\Beispiele\09_Ereignisse\
' Dateiname   09_18_OnTime.xls
' Modul       mdl_02_StatusBar_1

```

```
'=====

Dim ETime As Double
Dim StopTime As Double
Public BolTime As Boolean

' Uhrzeit in Statuszeile 1 Minute lang anzeigen
Sub StatusBarTime1()
    ETime = Now + TimeValue("00:00:10")

    If BolTime = False Then
        StopTime = Now + TimeValue("00:01:00")
        BolTime = True
    End If

    With ThisWorkbook
        Application.StatusBar = _
            Format(Now, "DDDD DD.MM.YYYY hh:mm:ss")
    End With

    Application.OnTime _
        EarliestTime:=ETime, _
        Procedure:="StatusBarTime1", _
        LatestTime:=StopTime

    If ETime >= StopTime Then
        Application.StatusBar = False
        BolTime = False
    End If
End Sub
```

Die Uhrzeit soll nur in unserer Beispieldatei angezeigt werden. Dementsprechend wird bei Deaktivierung der Mappe die Statuszeile zurückgesetzt, damit Excel deren Verwaltung in anderen Arbeitsmappen übernehmen kann.

```
'=====
' Auf CD      Buchdaten\Beispiele\09_Ereignisse\
' Dateiname   09_17_OnTime.xls
' Ereignis    Diese Arbeitsmappe
'=====

Private Sub Workbook_Deactivate()
    Application.StatusBar = False
End Sub
```

Spätestens wenn die Arbeitsmappe geschlossen wird, muss die Variable BolTime auf False gesetzt werden, um einen Abbruch der Zeitschleife sicherzustellen.

```
'=====
' Auf CD      Buchdaten\Beispiele\09_Ereignisse\
' Dateiname   09_18_OnTime.xls
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

```
' Ereignis Diese Arbeitsmappe
```

```
' =====
```

```
Private Sub Workbook_BeforeClose(Cancel As Boolean)
```

```
    BolTime = True
```

```
End Sub
```

323 Uhrzeit in Statuszeile steuern

Die erste Variante zur Anzeige der Uhrzeit in der Statuszeile wird durch die Angabe der »Endzeit« für eine bestimmte Zeitspanne angezeigt.

Genauso gut können Sie die Anzeige mittels einer booleschen Variablen per »Knopfdruck« regeln. Im folgenden Makro nutzen wir den Standartwert `False` einer Variablen vom Datentyp `Boolean`. Dabei wird die aktualisierte Uhrzeit so lange in die Statuszeile geschrieben, bis über ein weiteres Makro der Wahrheitswert der verwendeten Variablen auf `True` gesetzt und somit die Anzeige beendet wird. Die Stopp-Prozedur `SetBolTime2` haben wir einer Schaltfläche im Tabellenblatt zugewiesen.

```
' =====
```

```
' Auf CD Buchdaten\Beispiele\09_Ereignisse\
```

```
' Dateiname 09_18_OnTime.xls
```

```
' Modul mdl_03_StatusBar_2
```

```
' =====
```

```
Dim ETime2 As Double
```

```
Public BolTime2 As Boolean
```

```
' Uhrzeit in Statuszeile bis auf Widerruf anzeigen
```

```
Sub StatusBarTime2()
```

```
    ETime2 = Now + TimeValue("00:00:03")
```

```
    If BolTime2 = False Then
```

```
        Application.StatusBar = _
```

```
            Format(Now, "DDDD DD.MM.YYYY hh:mm:ss")
```

```
        Application.OnTime _
```

```
            EarliestTime:=ETime2, _
```

```
            Procedure:="StatusBarTime2"
```

```
    Else
```

```
        Application.StatusBar = False
```

```
        BolTime2 = False
```

```
    End If
```

```
End Sub
```

```
' Uhrzeit in Statuszeile beenden
```

```
Sub SetBolTime2()
```

```
    BolTime2 = True
```

```
End Sub
```

324 Uhrzeitmeldung

Das Makro OnTime4Start veranschaulicht die Verwendung des vierten Parameters Schedule der OnTime-Methode. Der Wert True führt eine neue OnTime-Prozedur aus. False löscht eine vorher eingestellte Prozedur. Der Standardwert ist True.

Bei der folgenden Prozedur wird alle fünf Sekunden die aktuelle Uhrzeit in einem Meldungsfenster angezeigt. Um diese Schleife zu unterbrechen, setzen Sie ein weiteres Makro ein. In diesem Stopp-Makro OnTime4Stop wird die termingesetzte Prozedur OnTime4Start wieder gelöscht, indem dem vierten Parameter der OnTime-Methode der Wert False zugewiesen wird. In dieser Prozedur muss sowohl die »genaue« Zeit als auch der »genaue« Name der Prozedur angegeben werden, welche gelöscht werden soll. Wenn VBA in der Liste der vorgemerkten Prozeduren die zu löschende Prozedur nicht findet, kommt es zu einer Fehlermeldung.

```
'=====
' Auf CD      Buchdaten\Beispiele\09_Ereignisse\
' Dateiname  09_18_OnTime.xls
' Modul      mdl_01_OnTimeMakros
'=====

Sub OnTime4Start()
    ET = Now + TimeValue("00:00:05")

    MsgBox " es ist jetzt " & Format(Time, "hh:mm:ss") & " Uhr"

    Application.OnTime _
        EarliestTime:=ET, _
        Procedure:="OnTime4Start"
End Sub

Sub OnTime4Stop()
    On Error Resume Next
    Application.OnTime _
        EarliestTime:=ET, _
        Procedure:="OnTime4Start", _
        Schedule:=False
End Sub
```

325 Tastenkombinationen erstellen

Jeder Tastendruck auf Ihrer Tastatur löst eine Aktion in der gerade aktiven Anwendung aus und stellt somit ein weiteres Ereignis dar. Drücken Sie in Excel eine oder mehrere Tasten gleichzeitig, wird eine bestimmte – für diese Taste/Tastenkombination definierte – Aktion ausgeführt. Am Beispiel der Zahl 2 zeigen wir Ihnen die verschiedenen Auswirkungen von Tastenkombinationen.

Zusatztasten	Aktion
2	Die Zahl 2 wird geschrieben
⇧ + 2	Das Anführungszeichen " wird geschrieben
Strg + 2	Markierte Zellen erhalten das Format Fett

Tabelle 76: Tastenkombinationen mit der Zahl 2

Grundlagen
Allgemein
Datum und Zeit
Formeln
Diagramme
Diagramm
Ereignisse
User-Form
Web/Mail
Extern
Gemischtes
Spezial

Zusatztasten	Aktion
 + 	Markierte Zellen erhalten das Wissenschaft-Format 0,00E+00 (X197,XL2002,XL2003)
 + 	Keine Aktion
 + 	Die hochgestellte ² wird geschrieben

Tabelle 76: Tastenkombinationen mit der Zahl 2 (Forts.)

Tipp

Eine Übersicht aller in Excel zur Verfügung stehenden Tastenkombinationen finden Sie in der Online-Hilfe von Excel. Drücken Sie die Taste , um die Microsoft Excel-Hilfe aufzurufen. Falls die Registerkarten nicht angezeigt werden, klicken Sie auf das erste Symbol EINBLENDEN. In der Registerkarte ANTWORTASSISTENT geben Sie das Wort »Tastenkombinationen« ein. Aus den aufgelisteten Themen wählen Sie »Tastenkombinationen«. In der rechten Fensterhälfte werden daraufhin verschiedene Kategorien mit Tastenkombinationen angezeigt. Klicken Sie auf eine Überschrift, um zu den Tastenkombinationen mitsamt Beschreibung zu gelangen.

Diese Übersicht finden Sie ebenfalls, wenn Sie den Weg über den Officeassistenten wählen.

Mit der OnKey-Methode können Aktionen auf alle Tasten und Tastenkombinationen individuell vorgegeben werden. Die nachfolgende Übersicht zeigt die verwendeten Zusatztasten und Syntaxen zu den in Tabelle 76 genannten Tastenkombinationen.

Zusatztasten	Syntax der OnKey-Methode
 über den Schreibblock der Tastatur	Application.OnKey »2«
 + 	Application.OnKey »+2«
 + 	Application.OnKey »^2«
 +  + 	Application.OnKey »^+2«
 + 	Application.OnKey »%2«
 + 	Application.OnKey »^%2«

Tabelle 77: Zusatztasten-Übersicht

Aus den Zusatztasten laut Tabelle 77 können beliebige Kombinationen erstellt werden. Sie können einer vorhandenen Tastenkombination eine neue Aktion in Form einer Prozedur zuweisen. Ebenfalls ist es möglich, nicht verwendete Kombinationen zu belegen. Dadurch werden keine schon verfügbaren Kombinationen außer Kraft gesetzt. Nachdem die Prozedur MyOnKey2 durchlaufen wurde, wird beim Drücken der Tastenkombination  +  +  das Makro MyProcedure aufgerufen. Üblicherweise würden die markierten Zellen im Wissenschaftsformat formatiert.

```
'=====
' Auf CD      Buchdaten\Beispiele\09_Ereignisse\
' Dateiname   09_19_OnKey.xls
' Modul       mdl_01_MyOnKey
'=====
```



```
Sub MyOnKey2()  
    ' STRG + Shift + 2 Makro zuweisen  
    Application.OnKey "^+2", "MyProcedure"  
End Sub  
  
Sub MyProcedure()  
    MsgBox _  
        "Sie haben <Strg + Shift + 2> gedrückt"  
End Sub
```

Um der Tastenkombination **Strg**+**⇧**+**2** deren ursprüngliche Bedeutung (Wissenschaftsformat) wieder zuzuweisen, verwenden Sie folgende Prozedur:

```
'=====
```

Auf CD	Buchdaten\Beispiele\09_Ereignisse\
Dateiname	09_19_OnKey.xls
Modul	mdl_01_MyOnKey

```
'=====
```

```
Sub ResetOnKey()  
    ' STRG + Shift + 2 zurücksetzen  
    Application.OnKey "^A"  
End Sub
```

Die von Excel zur Verfügung gestellten Tastenkombinationen können auch ganz deaktiviert werden. Nach dem Durchlauf der Prozedur `DeaktiviereOnKey` bewirkt das Drücken der Tasten **Strg**+**⇧**+**2** keine Aktion mehr.

```
'=====
```

Auf CD	Buchdaten\Beispiele\09_Ereignisse\
Dateiname	09_19_OnKey.xls
Modul	mdl_01_MyOnKey

```
'=====
```

```
Sub DeaktiviereOnKey()  
    ' STRG + Shift + 2 deaktivieren  
    Application.OnKey "^+2", ""  
End Sub
```

Um einem der Sonderzeichen (+, ^, % usw.) eine Prozedur zuzuweisen, schließen Sie das Zeichen in geschweifte Klammern ein. Verwenden Sie für Zeichen, die beim Drücken einer Taste nicht angezeigt werden (z.B. EINGABE oder TAB), die Codes aus der Tabelle 78. Jeder Code in der Tabelle steht für eine Taste auf der Tastatur.

Taste	Code
	{BACKSPACE} oder {BS}
	{BREAK}
	{CAPSLOCK}

Tabelle 78: Tasten, die nicht gedruckt werden

Grundlagen

Allgemein

Datum
Zeit

Formeln

Steuerungselemente

Befehle
ausführen

Objekte

Diagramme

Ereignisse

User-Formen

Web/Mail

Extern

Gemischtes

Spezial

Taste	Code
	{DELETE} oder {DEL}
	{DOWN}
	{END}
(Zehnertastatur)	{ENTER}
(Schreibtastatur)	~ (Tilde)
	{ESCAPE} oder {ESC}
	{HELP}
	{HOME}
	{INSERT}
	{LEFT}
	{NUMLOCK}
	{PGDN}
	{PGUP}
	{RETURN}
	{RIGHT}
	{SCROLLLOCK}
	{TAB}
	{UP}
bis	{F1} bis {F15}

Tabelle 78: Tasten, die nicht gedruckt werden (Forts.)

Sie können bereits bei der Aufzeichnung eines Makros über den Menüpunkt EXTRAS | MAKRO | AUFZEICHNEN eine Tastenkombination zuweisen.



Abbildung 260: Tastenkombination zuweisen

Hinweis

Wenn Sie einem Makro eine bereits bestehende Tastenkombination zuweisen, wird deren ursprüngliche Bedeutung zurückgesetzt. Weisen Sie einem Makro zum Beispiel die Tastenkombination + zu, wird die ursprüngliche Bedeutung (ganzes Tabellenblatt markieren) ignoriert. Spätestens nach einem Excel-Neustart werden alle normalen Tastenkombinationen wieder hergestellt.

Geben Sie in dem Dialog aus Abbildung 260: im Feld TASTENKOMBINATION einen Buchstaben ein, um dem ausgewählten Makro eine Tastenkombination zuzuweisen. Die endgültigen Tastenkombinationen setzen sich immer folgendermaßen zusammen: **Strg**+(eingegabener Buchstabe). Wenn Sie während der Eingabe des Buchstabens die **⇧**-Taste gedrückt halten, setzt sich die endgültige Tastenkombination folgendermaßen zusammen: **Strg**+**⇧**+(eingegabener_Buchstabe).

Wollen Sie einem bestehendem Makro nachträglich eine Tastenkombination zuweisen, klicken Sie im Menü EXTRAS | MAKRO | MAKROS die Schaltfläche OPTIONEN. In dem darauf erscheinenden Fenster MAKROOPTIONEN können Sie eine neue Tastenkombination eingeben oder eine bestehende ändern.

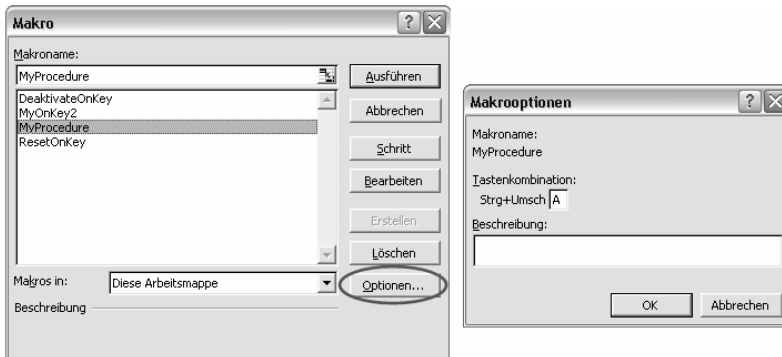


Abbildung 261: Makros nachträglich Tastenkombinationen zuweisen

Hinweis

Über die Dialog-Fenster MAKRO AUFZEICHNEN (siehe Abbildung 260) und MAKROOPTIONEN (siehe Abbildung 261) können Sie einer Prozedur nur Tastenkombinationen mit Buchstaben zuweisen. Als Zusatzstasten sind nur die Taste **Strg** und die Kombination **Strg**+**⇧** zulässig.

Über die OnKey-Methode können Sie jede beliebige Kombination aus Zusatzstasten mit Buchstaben oder Zahlen oder Befehlstasten erstellen.

326 Aufgabenbereich ein-/ausblenden

In Excel 2002 ist die Symbolleiste AUFGABENBEREICH hinzugekommen. Über den Menüpunkt ANSICHT | AUFGABENBEREICH kann die Anzeige dieser Symbolleiste aktiviert und deaktiviert werden. Sie beinhaltet folgende Registerkarten:

- ▶ Neue Arbeitsmappe
- ▶ Zwischenablage
- ▶ Suchen
- ▶ Clipart einfügen

Für die wechselnde Anzeige dieser Symbolleiste Task Pane ist in Excel 2003 die Standard-Tastenkombination **Strg**+**F1** hinzugekommen. Diese Tastenkombination ist in Excel 2002 noch nicht definiert.

Grundlagen

Allgemein

Datum
Zeit

tungen

Steuer
elemente

Befehl
leisten

Objekt

Diagramm

Ereignisse

User-
Form

Web/
Mail

Extern

Gemischtes

Spezial

Tipp

Wenn Sie die nachfolgenden Prozeduren in die Datei *PERSONL.xls* speichern, steht Ihnen diese »Excel 2003-Funktion« auch in allen Arbeitsmappen unter Excel 2002 zur Verfügung.

Beim Öffnen der Beispielmappe wird über das `Workbook_Activate`-Ereignis der Tastenkombination `[Strg] + [F1]` die Prozedur `MyTaskPane` zugewiesen. Wechseln Sie in eine andere geöffnete Excel-Datei, oder beenden Sie die Beispieldatei oder Excel, wird das `Workbook_Deactivate`-Ereignis ausgelöst. Hier wird die definierte Tastenkombination wieder zurückgesetzt. Die Tastenkombination erhält ihre normale Bedeutung zurück. Da Excel die verwendete Tastenkombination vor der Excel-Version 2003 nicht belegt hat, führt diese Tastenkombination keine Aktion mehr aus.

```
'=====
' Auf CD      Buchdaten\Beispiele\09_Ereignisse\
' Dateiname   09_20_TaskPane.xls
' Ereignis     DieseArbeitsmappe
'=====

Private Sub Workbook_Activate()
    Application.OnKey "^{F1}", "MyTaskPane"
End Sub

Private Sub Workbook_Deactivate()
    Application.OnKey "^{F1}"
End Sub
```

In der Prozedur `MyTaskPane` wird die verwendete Excel-Versionsnummer in einer `If ... Then`-Anweisung über die `Version`-Eigenschaft ausgelesen. Excel 2002 gibt die Versionsnummer 10.0 zurück. Wird eine andere Excel-Version genutzt, wird die Prozedur mit `Exit Sub` beendet. Über die `Visible`-Eigenschaft des `CommandBar`-Objektes wird der Anzeige-Zustand der Symbolleiste `Task Pane` bei jedem Drücken der Tastenkombination `[Strg] + [F1]` umgekehrt.

Achtung

Die Prozedur `MyTaskPane` ist nur in Excel 2002 funktionsfähig, da es bis Excel 2000 die Symbolleiste `AUFGABENBEREICH` nicht gibt. In Excel 2003 ist die verwendete Tastenkombination `[Strg] + [F1]` bereits für die Anzeige der Symbolleiste `AUFGABENBEREICH` integriert.

```
'=====
' Auf CD      Buchdaten\Beispiele\09_Ereignisse\
' Dateiname   09_20_TaskPane.xls
' Modul       mdl_01_TaskPane
'=====

Sub MyTaskPane()
    With Application
        If .Version <> "10.0" Then Exit Sub
        With .CommandBars("Task Pane")
            .Visible = Not .Visible
        End With
    End With
End Sub
```

Tipp

Die Prozedur `MyTaskPane` verdeutlicht an der Symbolleiste `AUFGABENBEREICH` die wechselnde Anzeige einer Befehlsleiste per Tastenkombination.

Statt der Aufgaben-Symbolleiste können Sie eine beliebige Befehlsleiste in der Prozedur angeben. Auch die Wahl der zugewiesenen Tastenkombination ist veränderbar.

Um Fehlermeldungen zu vermeiden, achten Sie darauf, in welchen Excel-Versionen die gewünschte Symbolleiste einsetzbar ist.

Weitere Informationen und Beispiele zu Symbolleisten finden Sie in der Kategorie »Befehlsleisten«.

327 Navigation mit Fadenkreuz

Um sich in größeren Tabellen leichter orientieren zu können, gibt es die Möglichkeit, die Aktionen der Pfeiltasten zu beeinflussen. Üblicherweise wird der Cursor durch Betätigen der Pfeiltasten jeweils um eine Zelle versetzt. Pfeil oben und Pfeil unten wechselt die Zeile, Pfeil links und Pfeil rechts die Spalte. In diesem Beispiel bringen wir die Zusatztaste `[Alt]` mit ins Spiel. Drücken Sie die Pfeiltasten jeweils in Verbindung mit der `[Alt]`-Taste, dann wird sowohl die ganze Zeile als auch die Spalte der aktiven Zelle aktiviert, also farblich hervorgehoben. Durch die Markierung verändert sich der Zellenhintergrund selbstständig. Es wird dabei jedoch keine Hintergrundfarbe zugewiesen, was deutlich wird, wenn die Markierung aufgehoben wird. Die aktive Zelle ist aktiviert. Dadurch wird deren Zellenhintergrund, als erste Zelle einer Markierung, weiß hinterlegt. Die normale Bedeutung der Pfeiltasten (zellenweise Cursorbewegung in einer Tabelle) wird durch dieses Beispiel nicht beeinträchtigt.

In der Beispielmappe haben wir in den Zellen des benutzten Bereichs Formeln hinterlegt (siehe Abbildung 262). Diese zeigen die Zeilen- und Spaltennummer der jeweiligen Zelle in zwei Zeilen an. Einen Zeilenumbruch in einer Zelle erzeugen Sie zum Beispiel über die Tastenkombination `[Alt] + [↵]`. Wir setzen für einen Zeilenumbruch die Funktion `ZEICHEN` ein. Das Zeichen `10` steht für einen Zeilenumbruch. In dem Dialog `ZELLEN FORMATIEREN` unter dem Menüpunkt `FORMAT | ZELLEN` Registerkarte `AUSRICHTUNG` wurde die Option `ZEILENUMBRUCH` aktiviert.

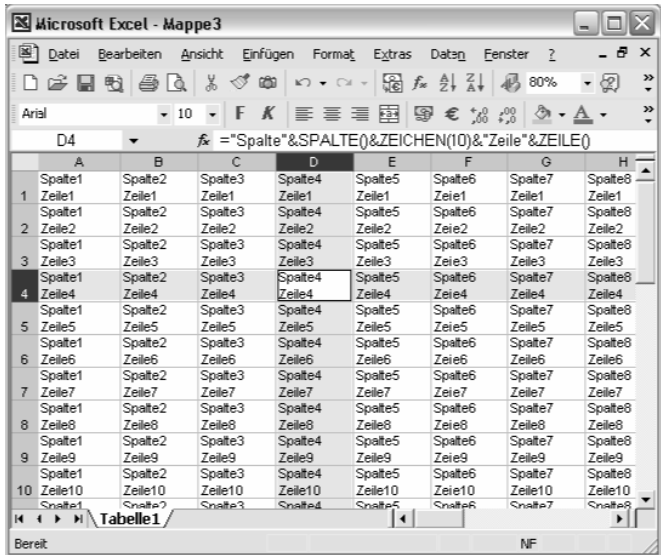


Abbildung 262: Spalte und Zeile der aktiven Zelle werden hervorgehoben

Grundlagen

Allgemein

Datum
Zeit

tungen

Steuer
elemente

Befehl
leisten

Objekt

Diagramm

Ereignisse

User-
Form

Web/
Mail

Extern

Gemischtes

Spezial

Beim Öffnen der Beispielmappe wird über das `Workbook_Activate`-Ereignis die Prozedur `ShortcutOn` aufgerufen. Wechseln Sie in eine andere geöffnete Excel-Datei, oder beenden Sie die Beispieldatei oder Excel, wird das `Workbook_Deactivate`-Ereignis ausgelöst. Hier wird die Prozedur `ShortcutOff` zum Zurücksetzen der Tastenkombinationen angegeben.

```
'=====
' Auf CD      Buchdaten\Beispiele\09_Ereignisse\
' Dateiname  09_21_Navigation.xls
' Ereignis    DieseArbeitsmappe
'=====

Private Sub Workbook_Activate()
    ShortcutOn
End Sub

Private Sub Workbook_Deactivate()
    ShortcutOff
End Sub
```

In der Prozedur `ShortcutOn` werden über die `OnKey`-Methode die einzelnen Makros zur Markierung der Zeilen und Spalten an die Tastenkombinationen `[Alt] + [Pfeiltasten]` zugewiesen. Über die `ShortcutOff`-Prozedur werden die definierten Tastenkombinationen wieder gelöscht.

```
'=====
' Auf CD      Buchdaten\Beispiele\09_Ereignisse\
' Dateiname  09_21_Navigation.xls
' Modul      mdl_01_ShortcutOnOff
'=====

Public Sub ShortcutOn()
    Application.OnKey "{UP}", "ArrowUp"
    Application.OnKey "{DOWN}", "ArrowDown"
    Application.OnKey "{LEFT}", "ArrowLeft"
    Application.OnKey "{RIGHT}", "ArrowRight"
End Sub

Public Sub ShortcutOff()
    Application.OnKey "{UP}", ""
    Application.OnKey "{DOWN}", ""
    Application.OnKey "{LEFT}", ""
    Application.OnKey "{RIGHT}", ""
End Sub
```

Die folgenden Prozeduren rufen jeweils die Prozedur `Mark` auf. Am Beispiel der Prozedur `ArrowUp` verdeutlichen wir die Übergabe der genannten Argumente an die `Mark`-Prozedur. Das Makro `ArrowUp` wird ausgelöst, wenn die Tastenkombination `[Alt] + [↑]` gedrückt wurde. Durch die Pfeiltaste `[↑]` wird normalerweise die Zellenmarkierung einer einzelnen Zelle verschoben. Durch `[↑]` wird die Zelle oberhalb der aktiven Zelle in der gleichen Spalte selektiert. Die aktive Zelle soll also um -1 Zeile und um 0 Spalten verschoben werden.

```
'=====
' Auf CD      Buchdaten\Beispiele\09_Ereignisse\
' Dateiname   09_21_Navigation.xls
' Modul       mdl_02_Arrows
'=====

Public Sub ArrowUp()
    Mark -1, 0
End Sub

Public Sub ArrowDown()
    Mark 1, 0
End Sub

Public Sub ArrowLeft()
    Mark 0, -1
End Sub

Public Sub ArrowRight()
    Mark 0, 1
End Sub
```

Am Beispiel der `ArrowUp`-Prozedur werden beim Aufruf der `Mark`-Prozedur die mitgegebenen Argumente `-1` und `0` an die Parameter `lngRow` und `lngCol` übergeben. Das erste Argument `lngRow` gibt die Zeilen-, das zweite Argument die Spaltenverschiebung zur aktiven Zelle an. Die drei weiteren `Arrow`-Prozeduren geben jeweils die entsprechenden Werte zur Zeilen- und Spaltenverschiebung an.

In der Variablen `rngCell` wird mit der `Offset`-Methode ein Verweis auf die neue Zellposition gesetzt. Die `Union`-Methode gibt die Vereinigung von mindestens zwei Bereichen zurück. Durch `Columns(rngCell.Column)` wird die ganze Spalte, durch `Rows(rngCell.Row)` die ganze Zeile der neuen Zelladresse in der `Union`-Methode zusammengebracht. Es entsteht ein Kreuz, welches als Mittelpunkt die aktive Zelle enthält (siehe Abbildung 262). Über die `Goto`-Methode werden die Zeile und Spalte, welche in der `Union`-Methode vereint wurden, markiert. Die in der Variablen `rngCell` gespeicherte Zelladresse wird aktiviert, da sonst die erste Zelle der Zeile als aktive Zelle angenommen würde.

```
'=====
' Auf CD      Buchdaten\Beispiele\09_Ereignisse\
' Dateiname   09_21_Navigation.xls
' Modul       mdl_03_Mark
'=====

Public Sub Mark(lngRow As Long, lngCol As Long)
    Dim rngCell As Range

    On Error Resume Next
    ' Zelle nach Pfeilrichtung versetzen
    Set rngCell = ActiveCell.Offset(lngRow, lngCol)

    With ActiveSheet
```

Grundlagen

Allgemein

Datum
Zeit

tungen

Steuerelemente

Befehle
leisten

Objekt

Diagramm

Ereignisse

User-Form

Web/Mail

Extern

Gemischtes

Spezial

```

' Vereinigung der aktiven Spalte und Zeile zum
' Navigationskreuz
Application.Goto _
    Union(.Columns(rngCell.Column), .Rows(rngCell.Row))
End With

' aktive Zelle aktivieren
rngCell.Activate
End Sub

```

328 Makro rückgängig machen

Nach Ausführung von VBA-Makros, welche Änderungen an der Tabelle vornehmen, ist die Funktion RÜCKGÄNGIG nicht mehr möglich. Alle vorhandenen Einträge, die beim Klick auf den Drop-Down-Pfeil des Symbols RÜCKGÄNGIG aufgelistet waren, werden gelöscht. Diese Einträge können nicht wiederhergestellt werden. Im Menü BEARBEITEN werden die Funktionen RÜCKGÄNGIG und WIEDERHOLEN ebenfalls deaktiviert.

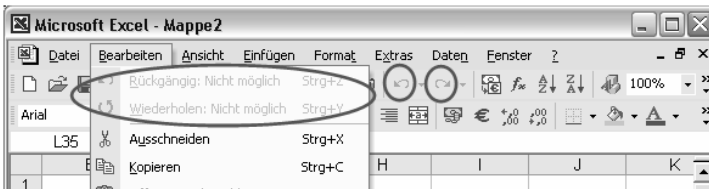


Abbildung 263: Deaktivierte Funktionen nach Makroausführung

Es ist jedoch möglich, die Befehle eines Makros rückgängig zu machen. Schreiben Sie eine Prozedur, die exakt das zurücksetzt, was in dem Makro ausgeführt wurde. Mit der `OnUndo`-Methode legen Sie den Text des Befehls RÜCKGÄNGIG und den Namen der Prozedur fest, die ausgeführt wird, wenn der Benutzer im Menü BEARBEITEN den Befehl RÜCKGÄNGIG wählt.

Mit der `OnRepeat`-Methode wird der Text des Menübefehls WIEDERHOLEN festgelegt. Geben Sie hier den Namen der Prozedur an, die ausgeführt wird, wenn der Benutzer den Befehl WIEDERHOLEN im Menü BEARBEITEN wählt. Im Gegensatz zu der `OnUndo`-Methode wird das entsprechende Icon in der Symbolleiste STANDARD nicht aktiviert.

Die Methoden `OnRepeat` und `OnUndo` müssen am Schluss einer Prozedur aufgerufen werden. Dies verhindert, dass die Prozeduren zum Wiederholen oder Rückgängigmachen durch nachfolgende Aktionen in der Prozedur überschrieben werden.

In unserem Beispiel greifen die Prozeduren `MyOnUndo` und `MyOnRepeat` beide auf die Variablen `mrngcell` und `mstrUndo` zu. Dementsprechend müssen diese Variablen auf Modulebene deklariert werden.

Hinweis

Die Variablen werden mit `Private` dimensioniert und oberhalb der ersten Prozedur deklariert. Dadurch sind sie in dem Modul, in welchem sie deklariert wurden, für alle Prozeduren sichtbar. Die Variable `mrngcell` setzt sich zum Beispiel aus dem Präfix `m` für Modul, `rng` für den Datentyp `Range` und `Cell` als Nomen zusammen. Durch diese Bezeichnungen sind Variablen im Code leichter zu identifizieren. Globale Variablen können zusätzlich mit dem Präfix `g` benannt werden.

In der Prozedur `MyOnUndo` wird in der Variablen `mrngCell` ein Verweis auf die aktive Zelle gesetzt. Anschließend schreiben wir den Text »Das Excel-VBA-Codebook, 2. Auflage« in diese Zelle. In der Variablen `mstrUndo` wird der Anzeigetext für die Befehle RÜCKGÄNGIG und WIEDERHOLEN gespeichert. Dieser setzt sich aus den ersten zehn Zeichen des Eingabetextes und der Adresse der geänderten Zelle zusammen. Der Anzeigetext `mstrUndo` sowie die Prozedur `MyOnRepeat`, welche beim Klick auf die Schaltfläche RÜCKGÄNGIG ausgeführt werden soll, werden an die `OnUndo`-Methode übergeben.

```
'=====
' Auf CD      Buchdaten\Beispiele\09_Ereignisse\
' Dateiname  09_22_OnUndoOnRepeat.xls
' Modul      mdl_01_OnUndoOnRepeat
'=====

Private mrngcell As Range
Private mstrUndo As String

Sub MyOnUndo()
    ' Verweis auf aktive Zelle
    Set mrngCell = ActiveCell

    ' Text in Zelle schreiben
    mrngCell.Value = "Das Excel-VBA-Codebook, 2. Auflage"

    ' Anzeigetext zusammensetzen
    mstrUndo = _
    Left(mrngCell.Value, 10) & " in " & mrngCell.Address(0, 0)

    ' Anzeigetext und Rückgängig-Makro übergeben
    Application.OnUndo "Undo: " & mstrUndo, "MyOnRepeat"
End Sub
```

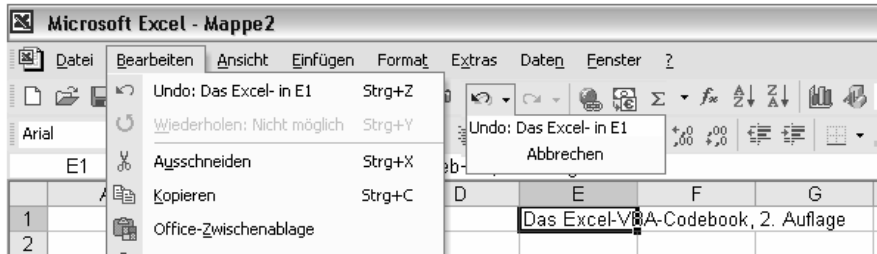


Abbildung 264: Text mit hinterlegter Prozedur im Rückgängig-Befehl

Klicken Sie nach Ausführung der Prozedur `MyOnUndo` auf das RÜCKGÄNGIG-Icon bzw. im Menü BEARBEITEN auf den RÜCKGÄNGIG-Befehl. Daraufhin wird die Prozedur `MyOnRepeat` aufgerufen. Hier wird die Eingabe des Textes »Das Excel-VBA-Codebook, 2. Auflage« rückgängig gemacht, indem der Inhalt der Zelle `mrngCell` gelöscht wird. Anschließend wird der Anzeigetext mittels der Variablen `mstrUndo` sowie der Prozedurname `MyOnUndo` an die `OnRepeat`-Methode übergeben. Nach Beendigung der Prozedur steht nun im Menü BEARBEITEN die Wiederholung der Eingabe »Das Excel-VBA-Codebook, 2. Auflage« zur Verfügung.

Grundlagen

Allgemein

Datum
Zeit

tungen

Steuer
elemente

Befehl
leisten

Objekt

Diagramm

Ereignisse

User-
Form

Web/
Mail

Extern

Gemischtes

Spezial

```
'=====
' Auf CD      Buchdaten\Beispiele\09_Ereignisse\
' Dateiname  09_22_OnUndoOnRepeat.xls
' Modul      mdl_01_OnUndoOnRepeat
'=====
```

```
Public Sub MyOnRepeat()
    ' Text aus Zelle löschen
    mrngCell.ClearContents

    ' Anzeigetext und Wiederholungs-Makro übergeben
    Application.OnRepeat "Repeat: " & mstrUndo, "MyOnUndo"
End Sub
```

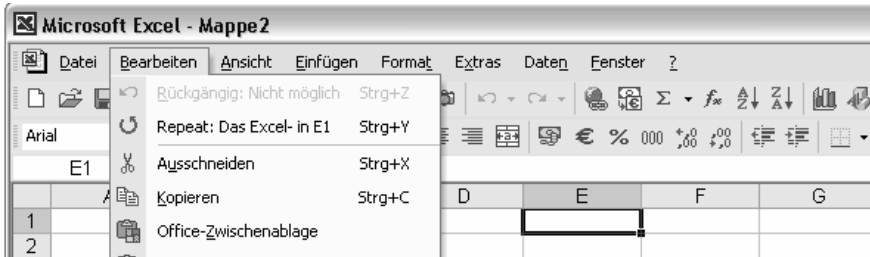


Abbildung 265: Text mit hinterlegter Prozedur im Wiederholen-Befehl

UserForm

UserForms sind Formulare, die den Benutzern die Eingabe von Daten erleichtern können. Sie sind die Alternative zur einfachen `MsgBox` und `InputBox`, welche ja beide sehr eingeschränkt in ihren Funktionen sind. Es gibt viele Möglichkeiten für den Einsatz von UserForm, wie Sie im Verlaufe des Buches noch sehen werden. Wir haben die Kategorie so aufgebaut, dass zu allen verfügbaren Steuerelementen ein Beispiel vorhanden ist. Im ersten Teil dieser Kategorie finden Sie Rezepte, die die einzelnen Steuerelemente beschreiben. Erst wenn ein Steuerelement beschrieben wurde, wird es in weiteren Beispielen zu finden sein. Nach der Beschreibung der einzelnen Steuerelemente finden Sie gemischt verschiedene Kombinationen an Steuerelementen.

Gewisse Grundkenntnisse im Umgang mit Steuerelementen setzen wir voraus. Wir behandeln in dieser Kategorie ausschließlich Steuerelemente in Bezug auf UserForm. Die Steuerelemente aus der Symbolleiste `STEUERELEMENT-TOOLBOX` werden separat in der Kategorie »Steuerelemente« behandelt.

Aus anderen Kategorien dieses Buches sind Sie es gewohnt, mit Modulen oder Ereignis-Prozeduren zu arbeiten. Um ein Formular zu erstellen, wird jedoch an Stelle eines *Moduls* ein *UserForm* erzeugt. In der VBA-Umgebung klicken Sie dazu auf den Menüpunkt `EINFÜGEN | USERFORM`. Alternativ können Sie mit der rechten Maustaste im Projektfenster auf das gewünschte Projekt klicken und aus dem Kontextmenü den Eintrag `EINFÜGEN | USERFORM` wählen. Daraufhin wird ein leeres Formular erzeugt. Wenn Sie das Formular anklicken, wird die Symbolleiste `WERKZEUGSAMMLUNG` eingeblendet. Sollte die Symbolleiste nicht angezeigt werden, dann blenden Sie diese ein, indem Sie auf den Menüpunkt `ANSICHT | WERKZEUGSAMMLUNG` klicken oder alternativ auf das entsprechende Symbol in der Symbolleiste klicken (es stellt gekreuzt einen Hammer und einen Schraubenschlüssel dar).

In der Werkzeugsammlung finden Sie verschiedene Elemente, um das UserForm zu gestalten. Aktivieren Sie per einfachen Mausklick das gewünschte Symbol und ziehen Sie im UserForm mit gedrückter linker Maustaste einen Rahmen. Das Steuerelement wird erzeugt. Das Steuerelement ist ein Objekt und kann beliebig verschoben werden. Durch Ziehen an den Eck- oder Seitenpunkten kann die Größe verändert werden.

Jedem Steuerelement können Ereignisprozeduren hinterlegt werden. Per Doppelklick auf das Steuerelement gelangen Sie ins Modul des Formulars und können einen VBA-Code hinterlegen. Um wieder zum Formular zu gelangen, klicken Sie im Projektfenster doppelt auf das UserForm (z.B. auf `UserForm1`).

Beachten Sie bei aktivem UserForm oder Steuerelement in der VBA-Umgebung das *Eigenschaften-Fenster*. Sie können hier verschiedene Einstellungen und Formatierungen vornehmen, wie z.B. Farben und Rahmen der Steuerelemente oder des UserForm selbst verändern.

Kehren wir zurück zur Werkzeugsammlung. Abbildung 266 zeigt die Symbolleiste mit den darin enthaltenen Steuerelementen. In der VBA-Codesprache werden die Elemente jeweils über den englischen Begriff angesprochen. Wenn Sie mit der Maustaste auf eins der Symbole zeigen und einen kurzen Moment warten, wird in der Quickinfo der zugehörige deutsche Begriff angezeigt. Im Fließtext dieses Buches werden wir vorzugsweise die deutschen Begriffe verwenden. Eine Übersicht über beide Begriffe können Sie der Tabelle 79 entnehmen.

Die Symbole werden (gemäß Abbildung 266) von links nach rechts beschrieben:

Grundlagen
Allgemein
Datum Zeit
tungen
Steuerelemente
Befehle leisten
Objekte
Diagramm
Ereignisse
User-Form
Web/ Mail
Extern
Gemischtes
Spezial

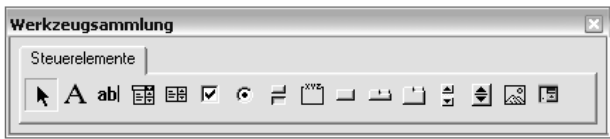


Abbildung 266: Die Werkzeugsammlung

Englische Bezeichnung	Präfix	Deutsche Bezeichnung
---	---	Objekte auswählen
Label	lbl	Bezeichnungsfeld
TextBox	txt	Textfeld
ComboBox	cmb	Kombinationsfeld
ListBox	lst	Listenfeld
CheckBox	chk	Kontrollkästchen
OptionButton	opt	Optionsfeld
ToggleButton	tgl	Umschaltfeld
Frame	frm	Rahmen
CommandButton	cmb	Befehlsschaltfläche
TapStrip	tbs	Register
MultiPage	mpg	Multiseiten
ScrollBar	vbs / hsb	Bildlaufleiste (vertikal / horizontal)
SpinButton	spn	Drehfeld
Image	img	Anzeige
RefEdit	ref	RefEdit

Tabelle 79: Englische und deutsche Bezeichnungen zu den Steuerelementen

Wenn Sie ein UserForm erzeugen, wird diesem automatisch der Name `UserForm1` zugewiesen. Weitere UserForms werden aufnummeriert `UserForm2`, `UserForm3` usw. Wenn Sie viele UserForms in einem Projekt erstellen, können Sie bei dieser Namensgebung leicht den Überblick verlieren. Wir haben uns deshalb dazu entschlossen, die UserForms aussagekräftig zu bezeichnen. Microsoft verwendet bei der Namensvergebung an erster Stelle die Buchstaben `frm` oder `fm`, woran leicht zu erkennen ist, dass es sich um ein Formular handelt. Wir haben das Präfix `frm` übernommen. Für unsere Beispiele werden wir die UserForms zudem nummerieren und mit einem Namen versehen, z. B. `frm_01_MeinName`. Um einem UserForm einen anderen Namen zu geben, muss das UserForm aktiv sein. Im Eigenschaften-Fenster finden Sie an erster Stelle den Eintrag (NAME). Überschreiben Sie im Eingabefeld rechterhand den Eintrag `USERFORM1`. Bestätigen Sie die Änderung mit . Im Projekt-Fenster können Sie nun den neuen Namen sehen. Wenn Sie den Namen in der *Titelzeile* des UserForm verändern möchten, dann überschreiben Sie den Eintrag `USERFORM1` der Eigenschaft `CAPTION`. Da die beiden Eigenschaften zu Beginn gleich heißen, kann dies anfänglich zu Verwirrungen führen. Sie werden sich jedoch schnell daran gewöhnen.

Bei den einzelnen Steuerelementen verhält es sich genauso. Hier haben wir jedoch auf eine Umbenennung der Elemente verzichtet, sofern es sich nicht um umfangreichere Projekte handelt. In kleinen Beispieldateien, in denen sich nur wenige Steuerelemente befinden, sind diese einfacher zu identifizieren, wenn deren Namen nicht geändert wird. Generell ist natürlich auch hier eine

saubere Benennung vorzuziehen, wie Sie später noch sehen werden. Verwenden Sie jeweils ein aussagekräftiges Präfix, wie z. B. `lbl` für Label, `cmb` für CommandButton usw., damit Sie trotz des Umbenennens noch erkennen können, um welchen Typ es sich handelt.

Um ein UserForm auszuführen, d. h. um das Formular auf dem Bildschirm anzuzeigen, drücken Sie in der VBA-Umgebung die Taste `[F5]`. Um ein UserForm von einem Tabellenblatt her aufzurufen, können Sie beispielsweise im Tabellenblatt eine Schaltfläche aus der Symbolleiste `STEUERELEMENT-TOOLBOX` erzeugen. Der Schaltfläche wird eine Prozedur hinterlegt, die das UserForm aufruft. Klicken Sie doppelt auf die Schaltfläche, um in die VBA-Umgebung zu gelangen. Zwischen den beiden bestehenden Codezeilen schreiben Sie den Namen des UserForm sowie die Methode `Show`:

```
Private Sub CommandButton1_Click()  
    frm_01_Label1.Show  
End Sub
```

Wechseln Sie wieder zum Tabellenblatt und deaktivieren Sie den Entwurfsmodus (die erste Schaltfläche in der Symbolleiste `STEUERELEMENT-TOOLBOX`). Wenn Sie nun auf die neu erstellte Schaltfläche klicken, wird das UserForm angezeigt.

Das Eigenschaften-Fenster

Im *Eigenschaften*-Fenster der UserForm bzw. der Steuerelemente der UserForm können verschiedene Einstellungen vorgenommen werden. Diese Einstellungen können auch mittels VBA-Code gesetzt oder geändert werden. So kann zum Beispiel die Hintergrundfarbe einer UserForm sowohl im Eigenschaften-Fenster (siehe Abbildung 267) als auch mittels Codezeilen gesteuert werden.

Per VBA können Sie ein UserForm beispielsweise beim Initialisieren, also direkt nach dem Aufrufen, rot einfärben. Die Farbe wird dann bei jedem Aufrufen des Formulars erneut zugewiesen:

```
Private Sub UserForm_Initialize()  
    UserForm1.BackColor = RGB(255, 0, 0)  
End Sub
```

Je nachdem, welches Element aktiv ist, werden im Eigenschaften-Fenster die jeweiligen Eigenschaften und Einstellungsmöglichkeiten angezeigt.

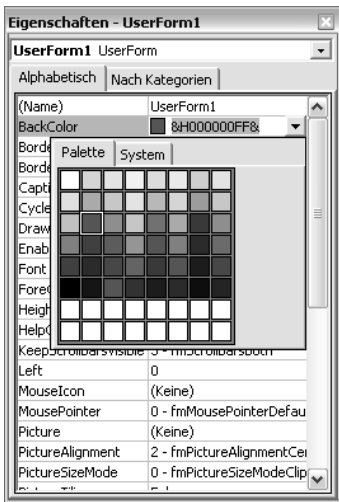


Abbildung 267: Das Eigenschaften-Fenster

Grundlagen
Allgemein
Datum und Zeit
Benutzereingaben
Steuerelemente
Befehlslisten
Objekt
Diagramm
Ereignisse
UserForm
Web/Mail
Extern
Gemischtes
Spezial

329 Bezeichnungsfelder (Label)

Bezeichnungsfelder und Textfelder werden oft verwechselt. Der Unterschied besteht darin, dass ein Bezeichnungsfeld lediglich einen Text oder einen Wert anzeigt. In einem Textfeld hingegen kann neben einer Textanzeige eine Eingabe durch den Anwender vorgenommen werden. Wie mit Textfeldern gearbeitet werden kann, werden Sie im nächsten Rezept erfahren.

Wenn Sie Ihrer UserForm ein Bezeichnungsfeld hinzufügen, trägt das Element automatisch den Namen NAME: LABEL1 und auch der Beschriftungstext CAPTION zeigt den Eintrag LABEL1 an. Sie können den Text neben CAPTION, also die Anzeige des Textes im Bezeichnungsfeld, im Eigenschaften-Fenster selbst oder per VBA verändern. Um beide Möglichkeiten zu testen, fügen wir je ein Bezeichnungsfeld in unser UserForm ein. Das erste Bezeichnungsfeld erhält den Text MEIN LABEL-TEXT AUS DEM EIGENSCHAFTEN-FENSTER. Diesen Text schreiben wir direkt ins Eigenschaften-Fenster. Den Text für das zweite Bezeichnungsfeld erzeugen wir per VBA-Code. Wir verwenden dazu das UserForm_Click-Ereignis. Dadurch wird der hinterlegte Code angezeigt, sobald das UserForm angeklickt wird.

Eine dritte Möglichkeit besteht darin, den Inhalt einer Zelle Ihres Tabellenblattes im UserForm auszugeben. Dazu fügen wir ein drittes Bezeichnungsfeld in das UserForm ein. Wir geben in die Zelle A1 den Text ein, der im LABEL3 angezeigt werden soll. Damit der Text per Klick auf das UserForm angezeigt wird, hinterlegen wir dem Code den Befehl Label3.Caption = Range("A1").Value. Wir werden zudem eine Formatierung im VBA-Code mitliefern. Die Hintergrundfarbe des Bezeichnungsfeldes soll in Rot RGB(255, 0, 0) und der Schriftschnitt in Fett .Font.Bold = True dargestellt werden. Die Schriftgröße legen wir auf 16 Punkte .Font.Size = 16 fest. Der Code sieht somit wie folgt aus:

```
' =====
' Auf CD      Buchdaten\Beispiele\10_UserForm\
' Dateiname   10_01_UserForm.xls
' UserForm    frm_01_Label
' =====

Private Sub UserForm_Click()
    Label2.Caption = "Mein Text aus dem VBA-Code"
    With Label3
        .Caption = Range("A1").Value
        .BackColor = RGB(255, 0, 0)
        .Font.Bold = True
        .Font.Size = 16
    End With
End Sub
```

Um den hinterlegten Text und die Formatierungen zu sehen, lassen Sie das UserForm auf dem Bildschirm anzeigen. Führen Sie die Ereignis-Prozedur UserForm_Click aus, indem Sie einen freien Bereich des UserForm anklicken.

330 Textfelder (TextBox)

Es gibt viele Möglichkeiten, Textfelder zu verwenden. Einige davon möchten wir Ihnen nun vorstellen. Wir erstellen ein neues UserForm, das fünf Bezeichnungsfelder sowie vier Textfelder enthält.

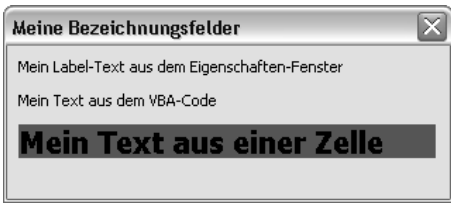


Abbildung 268: Bezeichnungsfelder

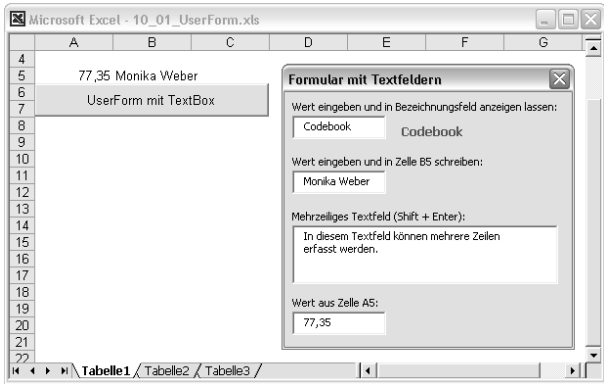


Abbildung 269: Textfelder

Im ersten Textfeld kann beliebig eine Eingabe vorgenommen werden. Die eingegebenen Zeichen werden im Bezeichnungsfeld, das sich rechts daneben befindet, angezeigt. Dazu verwenden wir das Ereignis `TextBox1_Change`. Mittels `TextBox1` wird der Name des Textfeldes angesprochen. Wenn Sie das Textfeld umbenannt haben, dann verwenden Sie den Namen, den Sie zugewiesen haben. Über das Ereignis `Change` erreichen wir, dass die eingetippten Zeichen laufend an das Bezeichnungsfeld übergeben werden.

Für die Übergabe benötigen Sie lediglich eine einzelne Codezeile. Der Beschriftung `Caption` des Bezeichnungsfeldes `Label2` wird der Wert `Value` des Textfeldes `TextBox1` übergeben.

```
'=====
' Auf CD      Buchdaten\Beispiele\10_UserForm\
' Dateiname  10_01_UserForm.xls
' UserForm   frm_02_TextBox
'=====

Private Sub TextBox1_Change()
    Label2.Caption = TextBox1.Value
End Sub
```

Das zweite Textfeld ist so aufgebaut, dass laufend während der Eingabe die Zeichen an die Zelle B5 übergeben werden. Auch hier arbeiten wir mit dem Ereignis `Change`.

```
'=====
' Auf CD      Buchdaten\Beispiele\10_UserForm\
' Dateiname  10_01_UserForm.xls
```

Grundlagen
Allgemein
Datum und Zeit
Formeln
Steuerungselemente
Befehlsliste
Objektmodell
Diagramme
Ereignisse
User-Formen
Web/Mail
Extern
Gemischtes
Spezial

```
' UserForm   frm_02_TextBox
' =====

Private Sub TextBox2_Change()
    Range("B5").Value = TextBox2.Value
End Sub
```

Wenn Sie ein einfaches Textfeld in ein UserForm einfügen, kann nur eine einzelne Zeile darin erfasst werden. Falls das Textfeld mehrere Zeilen umfassen soll, können Sie entweder im Eigenschaften-Fenster die Eigenschaft `MultiLine` auf `True` setzen, oder die Anweisung in einer Codezeile übergeben (`TextBox3.MultiLine = True`). Diese Einstellung soll in unserem Beispiel direkt beim Aufrufen des UserForm erfolgen. Wir verwenden deshalb eine Ereignis-Prozedur, die an das UserForm selbst gebunden ist. Damit der Code wie gewünscht direkt beim Aufrufen des UserForm ausgeführt wird, setzen wir das Ereignis `UserForm_Initialize` ein. Um einen manuellen Zeilenumbruch im Textfeld zu erzeugen, verwenden Sie die Tastenkombination .

In dieser Ereignis-Prozedur werden wir zudem das vierte Textfeld füllen. Der Wert, der in Zelle A5 des ersten Tabellenblattes steht, soll beim Aufrufen des UserForm in der `TextBox4` erscheinen.

```
' =====
' Auf CD     Buchdaten\Beispiele\10_UserForm\
' Dateiname  10_01_UserForm.xls
' UserForm   frm_02_TextBox
' =====

Private Sub UserForm_Initialize()
    TextBox3.MultiLine = True
    TextBox4.Value = Range("A5").Value
End Sub
```

Wenn der Wert in `TextBox4` verändert wird, sollen diese Änderungen laufend an die Zelle A5 in die TABELLE1 zurückgegeben werden. Dazu verwenden wir das Ereignis `Change`. Wenn Sie mit Zahlen arbeiten, raten wir Ihnen, den Datentyp während der Übergabe umzuwandeln. Sie stellen damit sicher, dass eine korrekte Anzeige in der Zelle erfolgt. Damit bei Eingabe eines falschen Wertes, z.B. eines Buchstabens, nicht der Debugger gestartet wird, nehmen wir in einer `If`-Entscheidung eine entsprechende Prüfung vor (`If IsNumeric(...)`).

```
' =====
' Auf CD     Buchdaten\Beispiele\10_UserForm\
' Dateiname  10_01_UserForm.xls
' UserForm   frm_02_TextBox
' =====

Private Sub TextBox4_Change()
    If IsNumeric(TextBox4.Value) Then
        Range("A5").Value = CDb1(TextBox4.Value)
    Else
        MsgBox "Bitte geben Sie eine Zahl ein."
    End If
End Sub
```

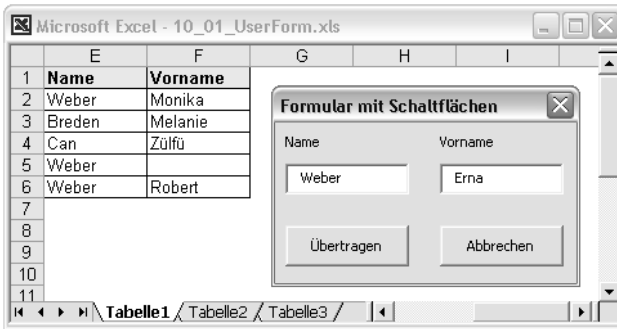


Abbildung 270: Schaltflächen auf einem UserForm

Die erste Prozedur wird an die Schaltfläche ÜBERTRAGEN (CommandButton1) gebunden. Wir verwenden das Ereignis Click. Die beiden Werte aus den Textfeldern sollen jeweils in dieselbe Zeile geschrieben werden. Die Werte werden beim Übertrag jeweils in eine neue Zeile geschrieben. Da in einem Datensatz ein Feld leer sein könnte, müssen wir in der Prozedur die Spalte ermitteln, welche am meisten Einträge enthält. Dazu verwenden wir drei Variablen. Der Variablen `intCol1` übergeben wir die Zeilennummer der letzten benutzten Zeile der Spalte 5 bzw. E. Der Variablen `intCol2` übergeben wir die Nummer der letzten benutzten Zeile der Spalte 6 bzw. F. In der If-Entscheidung wird geprüft, welche der beiden Variablen den größeren Wert enthält. Dieser Wert wird an die Variable `intBigger` übergeben. Die Variable enthält somit die Zeilennummer, welche als Nächste mit Daten gefüllt werden soll.

Für jede der beiden zu füllenden Zellen verwenden wir eine With-Anweisung, in welcher die Werte der Textfelder übergeben werden. Um das Bild abzurunden, wird jeweils ein Rahmen um die neu befüllten Zellen gezogen.

Nach den beiden With-Anweisungen wird der Inhalt der beiden Textfelder im UserForm entfernt. Dadurch kann eine neue Eingabe erfolgen, ohne dass erst der alte Text darin gelöscht werden muss. In der letzten Codezeile verwenden wir die Methode `SetFocus`, um den Cursor für eine Neueingabe wieder in das erste Textfeld zu setzen.

```
'=====
' Auf CD      Buchdaten\Beispiele\10_UserForm\
' Dateiname  10_01_UserForm.xls
' UserForm   frm_03_CommandButton
'=====
```

```
Private Sub CommandButton1_Click()
    Dim intCol1 As Integer
    Dim intCol2 As Integer
    Dim intBigger As Integer

    intCol1 = Cells(Rows.Count, 5).End(xlUp).Row
    intCol2 = Cells(Rows.Count, 6).End(xlUp).Row

    ' Prüfen, welche Spalte mehr Zeilen enthält
    If intCol1 > intCol2 Then
        intBigger = intCol1 + 1
    Else
        intBigger = intCol2 + 1
    End If
End Sub
```

<pre>End If ' Daten an die Spalte "E" übergeben und mit einem Rahmen versehen With Cells(intBigger, 5) .Value = TextBox1.Value .Borders(xlEdgeLeft).LineStyle = xlContinuous .Borders(xlEdgeTop).LineStyle = xlContinuous .Borders(xlEdgeBottom).LineStyle = xlContinuous .Borders(xlEdgeRight).LineStyle = xlContinuous End With ' Daten an die Spalte "F" übergeben und mit einem Rahmen versehen With Cells(intBigger, 6) .Value = TextBox2.Value .Borders(xlEdgeLeft).LineStyle = xlContinuous .Borders(xlEdgeTop).LineStyle = xlContinuous .Borders(xlEdgeBottom).LineStyle = xlContinuous .Borders(xlEdgeRight).LineStyle = xlContinuous End With ' Den Inhalt beider Textfelder entfernen TextBox1.Value = "" TextBox2.Value = "" ' Den Cursor, für eine neue Eingabe, wieder in das ' erste Textfeld setzen TextBox1.SetFocus End Sub</pre>	

Die Schaltfläche ABBRECHEN dient dazu, das UserForm zu verlassen. Die Anweisung `Unload Me` bewirkt, dass das angezeigte, aktive UserForm geschlossen wird.

Hinweis

Die Schaltfläche ABBRECHEN werden Sie in allen weiteren Rezepten dieser Kategorie finden. Wir werden darauf verzichten, sie erneut zu beschreiben.

<pre>'===== ' Auf CD Buchdaten\Beispiele\10_UserForm\ ' Dateiname 10_01_UserForm.xls ' UserForm frm_03_CommandButton '===== Private Sub CommandButton2_Click() Unload Me End Sub</pre>	

332 Rahmen (Frame)

Rahmen werden verwendet, um Steuerelemente zusammenzufassen. Auf einem UserForm können mehrere Rahmen verwendet werden, um beispielsweise die Steuerelemente in verschiedenen Gruppen zusammenzufassen.

Grundlagen
Allgemein
Datum und Zeit
Formen und Objekte
Steuerungselemente
Befehle
Objekte
Diagramme
Ereignisse
UserForm
Web/Mail
Extern
Gemischtes
Spezial

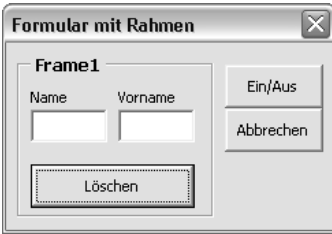


Abbildung 271: Mit aktiven und inaktiven Steuerelementen

In unserem Beispiel verwenden wir einen einzelnen Rahmen. Darin sind verschiedene Steuerelemente enthalten. Zum Ausführen der Ereignis-Prozedur verwenden wir die Schaltfläche EIN/AUS. Beim Anklicken der Schaltfläche werden alle im Rahmen enthaltenen Steuerelemente deaktiviert. Bei erneutem Anklicken der Schaltfläche werden die Steuerelemente wieder aktiviert. Dazu verwenden wir die Variable `ctrl`, die als `Typ Control` deklariert wird. In einer Schleife werden alle aktiven Steuerelemente deaktiviert. Bei erneutem Ausführen der Prozedur werden die Steuerelemente wieder aktiviert.

```
' =====
' Auf CD      Buchdaten\Beispiele\10_UserForm\
' Dateiname  10_01_UserForm.xls
' UserForm   frm_04_Frame
' =====
```

```
Private Sub CommandButton2_Click()
    Dim ctrl As Control

    For Each ctrl In Frame1.Controls
        ctrl.Enabled = Not ctrl.Enabled
    Next
End Sub
```

333 Listenfelder (ListBox)

In Listenfeldern können Sie eine Liste an Werten anzeigen lassen. Diese Werte können wahlweise selektiert und weiterverwendet werden. Unser UserForm umfasst drei Rahmen. In jedem Rahmen ist ein Listenfeld und eine Schaltfläche enthalten. Die Schaltflächen innerhalb der Rahmen dienen dazu, die markierten Einträge des Listenfeldes in einem Dialogfenster `MsgBox` auszugeben.

In Listboxen gibt es drei Selektionsmöglichkeiten, die Einzelselektion und zwei verschiedene Arten an Mehrfachselektionen. Für jede dieser Möglichkeiten haben wir auf unserem UserForm ein eigenes Listenfeld erstellt (siehe Abbildung 272). Im ersten Listenfeld kann jeweils nur ein Eintrag ausgewählt werden (`fmMultiSelectSingle`). Die Konstante wird der Eigenschaft `MultiSelect` zugewiesen. Im zweiten Listenfeld ist eine Mehrfachselektion möglich (`fmMultiSelectMulti`). Per Mausklick können Sie die gewünschten Einträge markieren oder die Markierung aufheben. Im dritten Listenfeld ist ebenfalls eine Mehrfachselektion möglich (`fmMultiSelectExtended`). Die Selektionsmöglichkeiten sind jedoch erweitert. Sie können beispielsweise mit gedrückter Maustaste mehrere aufeinander folgende Einträge auswählen. Alternativ halten Sie die `[⇧]`-Taste gedrückt, um aufeinander folgende Einträge zu markieren. Mit gedrückter `[Strg]`-Taste können Sie mehrere nicht aufeinander folgende Einträge selektieren.

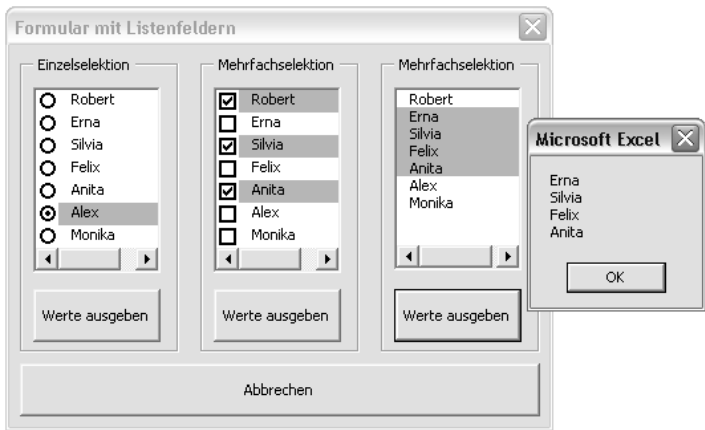


Abbildung 272: Listenfelder

Die Einträge, welche in den Listenfeldern enthalten sein sollen, finden Sie in der Beispielmappe auf dem zweiten Tabellenblatt. Die Einträge sollen beim Start des UserForm in die Listenfelder aufgenommen werden. Wir verwenden dazu das Ereignis `UserForm_Initialize`. Der Eigenschaft `RowSource` übergeben wir den Wert `"Tabelle2!A1:A7"`, welcher den Daten entspricht, die im Listenfeld angezeigt werden sollen. Alternativ zu dieser Codezeile können Sie die Einstellung im Eigenschaften-Fenster der Textfelder vornehmen. Die Eigenschaft `MultiSelect`, gefolgt von der gewünschten Konstanten, legt die Art des Listenfeldes in Bezug auf die Selektion fest.

Dem ersten und zweiten Listenfeld übergeben wir zudem die Konstante `fmListStyleOption` der Eigenschaft `ListStyle`. Abhängig von der Konstanten, die bei `MultiSelect` verwendet wurde, wird im Listenfeld entweder das Symbol für ein Optionsfeld (Kreis) oder Kontrollkästchen (Viereck) angezeigt (siehe Abbildung 272). In der letzten `With`-Anweisung verzichten wir auf die Eigenschaft `ListStyle`, da kein Symbol vor den Einträgen angezeigt werden soll.

```
'=====
' Auf CD      Buchdaten\Beispiele\10_UserForm\
' Dateiname   10_01_UserForm.xls
' UserForm    frm_05_ListBox
'=====

Private Sub UserForm_Initialize()
    With ListBox1
        .RowSource = "Tabelle2!A1:A7"
        .MultiSelect = fmMultiSelectSingle
        .ListStyle = fmListStyleOption
    End With

    With ListBox2
        .RowSource = "Tabelle2!A1:A7"
        .MultiSelect = fmMultiSelectMulti
        .ListStyle = fmListStyleOption
    End With

    With ListBox3
```

Grund-lagen
Allge-mein
DatuZeit
tungen
Steuer-elemen
Befehl-leisten
Objekt
Dia-gramm
Ereig-nisse
User-Form
Web/Mail
Extern
Ge-mischtes
Specia

```

        .RowSource = "Tabelle2!A1:A7"
        .MultiSelect = fmMultiSelectExtended
    End With
End Sub

```

Per Klick auf die Schaltfläche im ersten Rahmen wird ein Nachrichtenfenster `MsgBox` angezeigt. Der darin enthaltene Text entspricht dem selektierten Eintrag des ersten Listenfeldes. Damit der Debugger nicht gestartet wird, wenn kein Eintrag markiert wurde, verwenden wir eine `If`-Entscheidung, die prüft, ob die Auswahl leer ist. Wenn keine Auswahl getroffen wurde, wird die `MsgBox` nicht angezeigt.

```

'=====
' Auf CD      Buchdaten\Beispiele\10_UserForm\
' Dateiname   10_01_UserForm.xls
' UserForm    frm_05_ListBox
'=====

Private Sub CommandButton1_Click()
    If ListBox1.Value <> "" Then
        MsgBox ListBox1.Value
    End If
End Sub

```

Die Schaltfläche im zweiten Rahmen zeigt beim Draufklicken ebenfalls ein Meldungsfenster an. Da in diesem Listenfeld eine Mehrfachselektion möglich ist, müssen wir alle markierten Zeilen in einer `For`-Schleife auslesen. Bei Eintritt in die Schleife wird gezählt, wie viele Zeilen das Listenfeld enthält. Da der Index des ersten Listenfeldes nicht mit einer 1 beginnt, sondern mit einer 0, muss der Zähler der Schleife ebenfalls mit einer 0 beginnen. In einer `If`-Entscheidung wird geprüft, ob die Zeile markiert (`Selected`) ist. Wenn dies zutrifft, wird der Wert an die Variable `str` übergeben. Die Variable wird Zeile für Zeile erweitert. Wenn selektierte Einträge gefunden wurden, dann werden diese im Meldungsfenster angezeigt.

```

'=====
' Auf CD      Buchdaten\Beispiele\10_UserForm\
' Dateiname   10_01_UserForm.xls
' UserForm    frm_05_ListBox
'=====

Private Sub CommandButton2_Click()
    Dim i As Integer
    Dim str As String

    For i = 0 To ListBox2.ListCount - 1
        If ListBox2.Selected(i) = True Then
            str = str & ListBox2.List(i) & vbCrLf
        End If
    Next i

    If str <> "" Then
        MsgBox str
    End If
End Sub

```

```
End If
End Sub
```

Die Prozedur zur Schaltfläche im dritten Rahmen entspricht der vorangegangenen Prozedur. Der Unterschied besteht darin, dass hier das dritte Listenfeld `ListBox3` ausgelesen wird.

```
' =====
' Auf CD      Buchdaten\Beispiele\10_UserForm\
' Dateiname  10_01_UserForm.xls
' UserForm    frm_05_ListBox
' =====

Private Sub CommandButton3_Click()
    Dim i As Integer
    Dim str As String

    For i = 0 To ListBox3.ListCount - 1
        If ListBox3.Selected(i) = True Then
            str = str & ListBox3.List(i) & vbCrLf
        End If
    Next i

    If str <> "" Then
        MsgBox str
    End If
End Sub
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

334 Mehrspaltige Listenfelder

Im vorangegangenen Rezept haben wir mit einspaltigen Listenfeldern gearbeitet. Sie haben jedoch auch die Möglichkeit, mehrere Spalten und zusätzlich eine Überschrift im Listenfeld auszugeben. Das UserForm in diesem Beispiel umfasst ein Listenfeld und ein Bezeichnungsfeld für die Ausgabe des markierten Datensatzes.



Abbildung 273: Mehrspaltiges Listenfeld

Damit die gewünschte Anzahl an Spalten verfügbar ist, verwenden wir die Eigenschaft `ColumnsCount`, gefolgt von der Anzahl der Spalten. In unserem Beispiel sind es drei. Die zweite Codezeile legt die Quelle fest (`RowSource`). In der Quelle wird nur der Datenbereich, ohne Überschrift, festgelegt. Um eine Überschrift festzulegen, verwenden wir die Eigenschaft `ColumnHead`. Es wird dabei automatisch

eine Zeile oberhalb der Quelle verwendet. Wahlweise können Sie die Breite der Spalten angeben (ColumnWidths). Wenn die Optionsfelder (siehe Abbildung 273) angezeigt werden sollen, dann verwenden Sie dazu die Eigenschaft `ListStyle` gefolgt von der Konstanten `fmListStyleOption`.

```
' =====
' Auf CD      Buchdaten\Beispiele\10_UserForm\
' Dateiname  10_01_UserForm.xls
' UserForm   frm_06_ListBox
' =====

Private Sub UserForm_Initialize()
    With ListBox1
        .ColumnCount = 3
        .RowSource = "Tabelle2!E2:G7"
        .ColumnHeads = True
        .ColumnWidths = "2cm; 2cm; 2cm"
        .ListStyle = fmListStyleOption
    End With
End Sub
```

Um den markierten Datensatz in dem Bezeichnungsfeld auszugeben, verwenden wir das Ereignis `ListBox1_Change`. An drei Variablen wird je der Wert einer Zelle der selektierten Zeile übergeben, was den drei Spalten entspricht. Wir verwenden dazu die Eigenschaft `ListIndex`, gefolgt vom Index selbst. Wie bereits erwähnt beginnt die Nummerierung in Listenfeldern mit dem Wert 0. Die gefüllten Variablen werden in der letzten Codezeile miteinander verkettet. Zwischen jedem Feld wird zudem ein Bindestrich (-) eingefügt.

```
' =====
' Auf CD      Buchdaten\Beispiele\10_UserForm\
' Dateiname  10_01_UserForm.xls
' UserForm   frm_06_ListBox
' =====

Private Sub ListBox1_Change()
    Dim strCol1 As String
    Dim strCol2 As String
    Dim strCol3 As String

    With ListBox1
        strCol1 = .List(.ListIndex, 0)
        strCol2 = .List(.ListIndex, 1)
        strCol3 = .List(.ListIndex, 2)
    End With

    Label1.Caption = strCol1 & " - " & strCol2 & " - " & strCol3
End Sub
```

335 Kombinationsfelder (ComboBox)

Kombinationsfelder sind den Listenfeldern ähnlich. Beide zeigen eine Liste von Einträgen an. Der Unterschied besteht darin, dass beim Listenfeld einer oder mehrere Einträge auf einmal offen zu sehen sind. Wenn mehr Einträge vorhanden sind, als das Listenfeld anzeigen kann, erscheint eine

Bildlaufleiste am rechten Rand des Feldes. Das Kombinationsfeld hingegen zeigt lediglich eine Zeile an. Es handelt sich beim Kombinationsfeld um ein so genanntes DropDown. Erst beim Klick auf den Pfeil rechts neben dem Eingabefeld ist die gesamte Liste sichtbar.

Das Verhalten der beiden verwandten Elemente ist ansonsten gleich, das bedeutet, dass das Beispiel, das wir Ihnen an dieser Stelle aufzeigen, genauso für ein Listefeld verwendet werden kann.

Unser UserForm enthält zwei Kombinationsfelder sowie vier Schaltflächen.



Abbildung 274: UserForm mit zwei Kombinationsfeldern

Diesmal wird das Kombinationsfeld nicht mit Daten aus der Tabelle gefüllt, sondern die Werte werden innerhalb der Prozedur aufbereitet und an das Kombinationsfeld übergeben. Wir verwenden dazu das Ereignis `UserForm_Initialize`. Die Daten werden als Array an die Variable `varArray` übergeben. In einer `For`-Schleife wird das Array ausgelesen. Mittels `UBound` (Upper Boundary) und `LBound` (Lower Boundary) ermitteln wir die Ober- und Untergrenze des Arrays. So haben wir die Möglichkeit, nacheinander die Datensätze an das Kombinationsfeld zu übergeben (`AddItem`). Das erste Kombinationsfeld in unserem UserForm ist nun mit Werten gefüllt.

```
'=====
' Auf CD      Buchdaten\Beispiele\10_UserForm\
' Dateiname   10_01_UserForm.xls
' UserForm    frm_07_ComboBox
'=====
```

```
Private Sub UserForm_Initialize()
    Dim varArray As Variant
    Dim i As Integer

    varArray = Array("Robert", "Erna", "Silvia", _
                     "Felix", "Anita", "Alex", "Monika")

    For i = LBound(varArray) To UBound(varArray)
        Me.ComboBox1.AddItem varArray(i)
    Next
End Sub
```

Das zweite Kombinationsfeld ist anders aufgebaut. Die oberen drei Schaltflächen, die auf der Abbildung 274 zu sehen sind, beziehen sich alle auf das zweite Kombinationsfeld. Sie können die Liste, welche das DropDown-Feld ausmacht, selbst zusammenstellen, indem Sie Werte in das Feld eintragen.

Sie können beim Aktivieren des UserForm `UserForm_Activate` das Kombinationsfeld als geöffnet anzeigen lassen. Die Methode dazu lautet `DropDown`. Bei dieser Gelegenheit setzen wir auch gleich den Cursor `SetFocus` in das zweite Kombinationsfeld.

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

```
'=====
' Auf CD      Buchdaten\Beispiele\10_UserForm\
' Dateiname   10_01_UserForm.xls
' UserForm    frm_07_ComboBox
'=====

Private Sub UserForm_Activate()
    ComboBox1.DropDown
    ComboBox2.SetFocus
End Sub
```

Als Nächstes wollen wir den Code für die drei Schaltflächen programmieren. Per Klick auf die Schaltfläche EINTRAG HINZUFÜGEN soll ein neuer Eintrag ins zweite Kombinationsfeld aufgenommen werden. Schreiben Sie den Text, der in das Listenfeld aufgenommen werden soll, direkt in das zweite Kombinationsfeld und klicken Sie dann auf die Schaltfläche EINTRAG HINZUFÜGEN. In der Prozedur verwenden wir eine *With*-Anweisung, um die Befehle für das zweite Kombinationsfeld zusammenzufassen. Die erste Befehlszeile in der *With*-Anweisung bewirkt, dass der eingetippte Eintrag ins Kombinationsfeld aufgenommen wird. Die zweite Codezeile in der *With*-Anweisung leert das Eingabefeld des Kombinationsfeldes, so dass weitere Einträge eingetippt werden können, ohne dass der zuvor eingegebene Eintrag manuell gelöscht werden muss. Die dritte und letzte Codezeile in der *With*-Anweisung stellt sicher, dass sich der Cursor im zweiten Kombinationsfeld befindet. Damit ist das Feld für die nächste Eingabe bereit.

```
'=====
' Auf CD      Buchdaten\Beispiele\10_UserForm\
' Dateiname   10_01_UserForm.xls
' UserForm    frm_07_ComboBox
'=====

Private Sub CommandButton1_Click()
    With ComboBox2
        .AddItem .Value
        .Value = ""
        .SetFocus
    End With
End Sub
```

Per Klick auf die Schaltfläche EINTRAG ENTFERNEN wird ein einzelner Eintrag aus dem Kombinationsfeld gelöscht (*RemoveItem*). Dazu muss zuerst der Eintrag ausgewählt werden, der gelöscht werden soll.

```
'=====
' Auf CD      Buchdaten\Beispiele\10_UserForm\
' Dateiname   10_01_UserForm.xls
' UserForm    frm_07_ComboBox
'=====

Private Sub CommandButton2_Click()
    With ComboBox2
        If .ListIndex >= 0 Then
```

```
        .RemoveItem (.ListIndex)
        .Value = ""
    End If
End With
End Sub
```

Per Klick auf die Schaltfläche ALLE EINTRÄGE ENTFERNEN wird der Inhalt der gesamten Liste gelöscht. Die Methode dazu lautet Clear.

```
'=====
' Auf CD      Buchdaten\Beispiele\10_UserForm\
' Dateiname  10_01_UserForm.xls
' UserForm   frm_07_ComboBox
'=====

Private Sub CommandButton3_Click()
    ComboBox2.Clear
End Sub
```

336 Werte von Kombinationsfeld nach Listenfeld übertragen

Bei diesem Rezept wird deutlich, dass sich Kombinations- und Listenfelder in der Handhabung kaum unterscheiden. Für unser UserForm verwenden wir ein Kombinationsfeld und ein Listenfeld sowie zwei Schaltflächen. Ziel ist es, Einträge aus dem Kombinationsfeld an das Listenfeld zu übergeben.

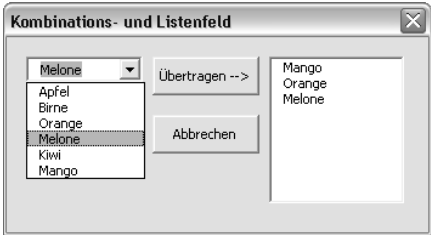


Abbildung 275: Daten aus dem Kombinationsfeld an das Listenfeld übergeben

Für die Übergabe der Werte verwenden wir hier kein Array wie im letzten Rezept, sondern fügen die Werte mittels AddItem einzeln ins Kombinationsfeld ein.

```
'=====
' Auf CD      Buchdaten\Beispiele\10_UserForm\
' Dateiname  10_01_UserForm.xls
' UserForm   frm_08_ComboToList
'=====

Private Sub UserForm_Initialize()
    With ComboBox1
        .AddItem "Apfel"
```

Grundlagen

Allgemein

Datum
Zeit

ungen

Steuer
elemente

Befehl
leisten

Objekt

Diagramm

Ereignisse

User-
Form

Web/
Mail

Extern

Gemischtes

Spezial

```

.AddItem "Birne"
.AddItem "Orange"
.AddItem "Melone"
.AddItem "Kiwi"
.AddItem "Mango"
End With
End Sub

```

Per Klick auf die Schaltfläche ÜBERTRAGEN --> wird der ausgewählte Datensatz aus dem Kombinationsfeld an das Listenfeld übergeben.

```

' =====
' Auf CD      Buchdaten\Beispiele\10_UserForm\
' Dateiname   10_01_UserForm.xls
' UserForm    frm_08_ComboToList
' =====

Private Sub CommandButton1_Click()
    If ComboBox1.ListIndex >= 0 Then
        ListBox1.AddItem ComboBox1.Value
    End If
End Sub

```

337 Optionsfelder (OptionButton)

Optionsfelder werden in der Regel in Gruppen eingesetzt. In einer solchen Gruppe kann maximal ein Optionsfeld aktiv sein. Sie haben die Möglichkeit, die Optionsfelder in einem Rahmen `Frame` zusammenzufassen, oder mit einem Gruppennamen zu arbeiten und dabei auf einen Rahmen zu verzichten. Beide Lösungen werden wir uns ansehen.

Wir verwenden insgesamt sechs Optionsfelder, zwei Bezeichnungsfelder, einen Rahmen und drei Schaltflächen.

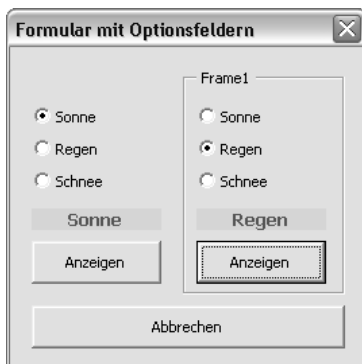


Abbildung 276: Optionsfelder

In der linken Hälfte des UserForm befinden sich drei Optionsfelder, die zusammen eine Gruppe bilden. Da sich die Optionsfelder nicht in einem Rahmen befinden, müssen wir jedem der Felder einen Gruppennamen `GroupName` zuweisen, um sie logisch miteinander zu verbinden. Diesen Vor-

gang weisen wir dem Ereignis `UserForm_Initialize` zu. Es soll außerdem das erste Optionsfeld beim Starten des `UserForm` aktiviert werden (`Value = True`).

```
'=====
' Auf CD      Buchdaten\Beispiele\10_UserForm\
' Dateiname  10_01_UserForm.xls
' UserForm   frm_09_OptionButton
'=====

Private Sub UserForm_Initialize()
    With OptionButton1
        .GroupName = "MyGroup"
        .Value = True
    End With

    OptionButton2.GroupName = "MyGroup"
    OptionButton3.GroupName = "MyGroup"
End Sub
```

Per Klick auf die Schaltfläche ANZEIGEN wird die Beschriftung des aktiven Optionsfeldes ausgelesen und in dem Bezeichnungsfeld angezeigt. Dazu verwenden wir eine `For`-Schleife. In der ersten `If`-Entscheidung innerhalb der `For`-Schleife wird geprüft, ob es sich bei dem Steuerelement um ein Optionsfeld handelt. Dies ist erforderlich, da sich in dem Rahmen auch noch andere Steuerelemente befinden könnten, die jedoch nicht mit in die Gruppe einbezogen werden sollen. Die Prüfung erfolgt, indem die linken sechs Zeichen des Namens des Steuerelementes mit der Zeichenkette »Option« verglichen werden. Dies funktioniert natürlich nur unter der Voraussetzung, dass alle entsprechenden Steuerelemente mit »Option« im Namen beginnen. In der zweiten `If`-Entscheidung wird der Gruppenname abgefragt. In der dritten `If`-Entscheidung wird geprüft, ob der Wert `True`, also Wahr für aktiv aufweist. Bei Zutreffen aller Bedingungen wird der Beschriftungstext `Caption` des ermittelten Optionsfeldes an das erste Bezeichnungsfeld übergeben.

```
'=====
' Auf CD      Buchdaten\Beispiele\10_UserForm\
' Dateiname  10_01_UserForm.xls
' UserForm   frm_09_OptionsButton
'=====

Private Sub CommandButton1_Click()
    Dim ctrl As Control

    For Each ctrl In Me.Controls
        If Left(ctrl.Name, 6) = "Option" Then
            If ctrl.GroupName = "MyGroup" Then
                If ctrl.Value = True Then
                    Label1.Caption = ctrl.Caption
                    Exit For
                End If
            End If
        End If
    Next ctrl
End Sub
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

Ähnlich verhält es sich, wenn die Optionsfelder in einem Rahmen zusammengefasst sind. Hierbei können Sie jedoch auf die Überprüfung des Gruppennamens verzichten.

```
'=====
' Auf CD      Buchdaten\Beispiele\10_UserForm\
' Dateiname  10_01_UserForm.xls
' UserForm   frm_09_OptionsButton
'=====

Private Sub CommandButton2_Click()
    Dim ctrl As Control

    For Each ctrl In Frame1.Controls
        If Left(ctrl.Name, 6) = "Option" Then
            If ctrl.Value = True Then
                Label2.Caption = ctrl.Caption
            End If
        End If
    Next ctrl
End Sub
```

338 Kontrollkästchen (CheckBox)

Kontrollkästchen sind mit den Optionsfeldern verwandt. Der Unterschied besteht erstens darin, dass sie viereckig und nicht rund sind. Dadurch können Sie eindeutig erkennen, ob es sich um ein Optionsfeld oder ein Kontrollkästchen handelt. Der zweite Unterschied besteht darin, dass bei der Verwendung von Kontrollkästchen mehrere Auswahlmöglichkeiten innerhalb einer Gruppe bestehen. Es können kein, ein oder mehrere Kontrollkästchen in einem Rahmen aktiv sein. Es ist somit eine *Mehrfachauswahl* möglich. Zum Vergleich: Bei Optionsfeldern kann, wie Sie im letzten Rezept erfahren haben, jeweils maximal eines der Felder aktiv sein. Es handelt sich dabei um eine *Entscheidung*.

In diesem Beispiel arbeiten wir mit einem Rahmen, der vier Kontrollkästchen beinhaltet sowie eine Befehlsschaltfläche.



Abbildung 277: Kontrollkästchen

Beim Aktivieren eines Kontrollkästchens ändert sich dessen Hintergrundfarbe entsprechend der Beschriftung. Wenn das Kontrollkästchen deaktiviert wird, dann wird die Hintergrundfarbe zurückgesetzt.

Für jedes Kontrollkästchen muss eine eigene Ereignis-Prozedur erstellt werden. Wir verwenden hier das Ereignis `Click`. Beim Klick auf eins der Kontrollkästchen wird dessen Wert auf `True`, falls aktiv, oder `False`, falls inaktiv gesetzt. Im Code wird in einer `Select Case`-Anweisung dieser Wert

abgefragt. Wenn das Kontrollkästchen aktiv, also `True` ist, dann wird der entsprechende RGB-Farbwert an die Eigenschaft der Hintergrundfarbe `BackColor` des Kästchens übergeben. Wenn das Kontrollkästchen inaktiv ist, wird die Farbe zurückgesetzt. Sie können sowohl mit RGB-Farbwerten als auch mit den Werten arbeiten, die das Eigenschaften-Fenster anzeigt.

Prozedur für das Kontrollkästchen ROT:

```
'=====
' Auf CD      Buchdaten\Beispiele\10_UserForm\
' Dateiname  10_01_UserForm.xls
' UserForm   frm_10_CheckBox
'=====

Private Sub CheckBox1_Click()
    Select Case CheckBox1.Value
        Case True
            CheckBox1.BackColor = RGB(255, 0, 0)
        Case False
            CheckBox1.BackColor = &H8000000F
    End Select
End Sub
```

Prozedur für das Kontrollkästchen GRÜN:

```
'=====
' Auf CD      Buchdaten\Beispiele\10_UserForm\
' Dateiname  10_01_UserForm.xls
' UserForm   frm_10_CheckBox
'=====

Private Sub CheckBox2_Click()
    Select Case CheckBox2.Value
        Case True
            CheckBox2.BackColor = RGB(0, 255, 0)
        Case False
            CheckBox2.BackColor = &H8000000F
    End Select
End Sub
```

Prozedur für das Kontrollkästchen BLAU:

```
'=====
' Auf CD      Buchdaten\Beispiele\10_UserForm\
' Dateiname  10_01_UserForm.xls
' UserForm   frm_10_CheckBox
'=====

Private Sub CheckBox3_Click()
    Select Case CheckBox3.Value
        Case True
            CheckBox3.BackColor = RGB(0, 0, 255)
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

```

        Case False
            CheckBox3.BackColor = &H8000000F
        End Select
    End Sub

```

Prozedur für das Kontrollkästchen GELB:

```

'=====
' Auf CD      Buchdaten\Beispiele\10_UserForm\
' Dateiname  10_01_UserForm.xls
' UserForm   frm_10_CheckBox
'=====

Private Sub CheckBox4_Click()
    Select Case CheckBox4.Value
        Case True
            CheckBox4.BackColor = RGB(255, 255, 0)
        Case False
            CheckBox4.BackColor = &H8000000F
    End Select
End Sub

```

339 Umschaltfelder (ToggleButton)

Umschaltfelder sind mit Befehlsschaltflächen verwandt. Der Unterschied besteht darin, dass ein Umschaltfeld, wenn es angeklickt wird, vertieft dargestellt wird. Bei erneutem Anklicken wird das Feld wieder erhöht angezeigt.

In unserem UserForm befindet sich ein Umschaltfeld, ein Bezeichnungsfeld sowie die Schaltfläche ABBRECHEN.

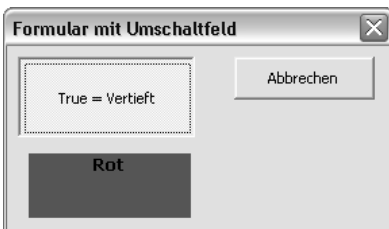


Abbildung 278: Umschaltfeld

Wenn das Umschaltfeld aktiv ist, also vertieft dargestellt wird, ist sein Wert Wahr `True`, ansonsten Falsch `False`. In unserem Beispiel fragen wir in einer `Select Case`-Anweisung diesen Wert ab. Wenn der Wert unserer Umschaltfläche `True` ist, wird dem Bezeichnungsfeld eine rote Hintergrundfarbe und die Bezeichnung `ROT` zugewiesen. Der Text auf dem Umschaltfeld verändert sich in `TRUE = VERTIEFT`. Wenn der Wert `False` ist, wird dem Bezeichnungsfeld eine blaue Hintergrundfarbe und die Bezeichnung `BLAU` zugewiesen. Der Text auf dem Umschaltfeld verändert sich in `FALSE = ERHÖHT`.


```
' =====
' Auf CD      Buchdaten\Beispiele\10_UserForm\
' Dateiname  10_01_UserForm.xls
' UserForm   frm_11_ToggleButton
' =====

Private Sub ToggleButton1_Click()
    Select Case ToggleButton1
        Case True
            Label1.BackColor = RGB(255, 0, 0)
            Label1.Caption = "Rot"
            ToggleButton1.Caption = "True = Vertieft"
        Case False
            Label1.BackColor = RGB(0, 0, 255)
            Label1.Caption = "Blau"
            ToggleButton1.Caption = "False = Erhöht"
    End Select
End Sub
```

340 Register (TabStrip)

Wenn Sie ein Register in Ihr UserForm einfügen, weist dieses standardmäßig zwei Seiten auf. Sie können weitere Seiten einfügen, indem Sie mit der rechten Maustaste im Register auf die Überschrift eines der Registernamen klicken und so das Kontextmenü aufrufen. Wählen Sie im Kontextmenü den Eintrag **NEUE SEITE**. Ebenfalls im Kontextmenü finden Sie den Eintrag **UMBENENNEN**. Sie können damit den einzelnen Registerblättern einen neuen Namen zuweisen.

Wenn Sie auf Ihrem Register ein Steuerelement einfügen, zum Beispiel eine Schaltfläche, dann wird diese Schaltfläche automatisch auf allen Registerkarten angezeigt. Alle Registerkarten zeigen somit immer denselben Inhalt an.

Auf unserem UserForm befindet sich ein Register mit drei Registerkarten. Innerhalb des Registers haben wir eine Schaltfläche eingefügt.



Abbildung 279: Register mit drei Registerkarten

Beim Wechseln der Registerkarte soll die Schaltfläche eine andere Farbe und eine andere Beschriftung annehmen. Wir verwenden dazu eine `Select Case`-Anweisung. Es wird jeweils auf den Index des selektierten Registers Bezug genommen (`SelectedItem.Index`).

```
' =====
' Auf CD      Buchdaten\Beispiele\10_UserForm\
' Dateiname  10_01_UserForm.xls
' UserForm   frm_12_TabStrip
' =====
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

```

Private Sub TabStrip1_Change()
    With CommandButton1
        Select Case TabStrip1.SelectedItem.Index
            Case 0
                .BackColor = RGB(255, 0, 0)
                .Caption = "Rot"
            Case 1
                .BackColor = RGB(0, 255, 0)
                .Caption = "Grün"
            Case 2
                .BackColor = RGB(0, 0, 255)
                .Caption = "Blau"
        End Select
    End With
End Sub

```

Beim Klick auf die farbige Schaltfläche innerhalb des Registers soll je nach Registerkarte ein anderer Text in einem Meldungsfenster angezeigt werden.

```

'=====
' Auf CD      Buchdaten\Beispiele\10_UserForm\
' Dateiname   10_01_UserForm.xls
' UserForm    frm_12_TabStrip
'=====

Private Sub CommandButton1_Click()
    Select Case TabStrip1.SelectedItem.Index
        Case 0
            MsgBox "Sie befinden sich im Register ""Rot""
        Case 1
            MsgBox "Sie befinden sich im Register ""Grün""
        Case 2
            MsgBox "Sie befinden sich im Register ""Blau""
    End Select
End Sub

```

341 Multiseiten (Multipage)

Multiseiten sind mit den Registern verwandt. Im Gegensatz zu den Registern können Sie Multiseiten unabhängig voneinander gestalten. Das bedeutet, dass jede Registerkarte einen anderen Inhalt aufweisen kann.

In unserem UserForm befinden sich eine Multiseite und vier Schaltflächen. Auf der ersten Registerkarte PAGE1 der Multiseite sind zwei Bezeichnungsfelder und zwei Textfelder enthalten. Auf der zweiten Seite PAGE2 sind drei Optionsfelder und auf der dritten Seite PAGE3 zwei Kontrollkästchen zu finden.



Abbildung 280: Multiseiten

Achten Sie bei der Abbildung 280 darauf, dass jeweils nur eine Registerkarte verfügbar ist. Die anderen sind inaktiv und somit hellgrau dargestellt. Bei den Schaltflächen ZURÜCK und WEITER verhält es sich ähnlich. Je nachdem, ob vor- oder zurückgeblättert werden kann, ist der Schaltflächentext entweder schwarz oder hellgrau dargestellt. Wenn also eine Schaltfläche nicht betätigt werden kann, z.B. die Schaltfläche ZURÜCK auf der ersten Multiseite, dann wird sie inaktiv dargestellt. Um einen solchen Effekt zu erreichen, muss ein entsprechender Programmcode hinterlegt werden.

Beim Initialisieren des UserForm stellen wir in der With-Anweisung sicher, dass nur die erste Registerkarte aktiv ist (Value = 0), wobei der Wert 0 für den Index der ersten Seite steht. Die zweite Registerkarte hat den Index 1 und die dritte den Index 2.

In den beiden weiteren Codezeilen wird sichergestellt, dass die Registerkarten 2 und 3 (Index 1 und 2) inaktiv sind (Pages(x).Enabled = False).

Nach der With-Anweisung wird festgelegt, dass die Schaltfläche ZURÜCK inaktiv ist. Die Schaltfläche WEITER jedoch wird aktiviert.

```
'=====
' Auf CD      Buchdaten\Beispiele\10_UserForm\
' Dateiname  10_01_UserForm.xls
' UserForm   frm_13_MultiPage
'=====

Private Sub UserForm_Initialize()
    With MultiPage1
        .Value = 0                ' Registerkarte 1
        .Pages(1).Enabled = False ' Registerkarte 2
        .Pages(2).Enabled = False ' Registerkarte 3
    End With

    CommandButton1.Enabled = False ' Schaltfläche "Zurück"
    CommandButton2.Enabled = True  ' Schaltfläche "Weiter"
End Sub
```

Grund-lagen

Allge-mein

Datu Zeit

tungen

Steuer elemen

Befehl leisten

Objekt

Dia-gramm

Ereig-nisse

User-Form

Web/Mail

Extern

Ge-misch tes

Specia

Als Nächstes widmen wir uns der Schaltfläche ZURÜCK. Die Schaltfläche soll nur dann als aktiv dargestellt werden, wenn sie verwendet werden kann, nämlich auf der Registerkarte 2 und 3. Beim Betätigen der Schaltfläche ZURÜCK soll jeweils die linke Registerkarte aktiviert werden. Die anderen Registerkarten werden inaktiv dargestellt, so dass immer nur eine Registerkarte aktiv ist.

Da beim Initialisieren die Einstellungen für die erste Registerkarte bereits vorgenommen werden, brauchen diese hier nicht erneut programmiert zu werden. Die erste Registerkarte wird erst in der nächsten Prozedur wieder angesprochen, nämlich dann, wenn wir den Code für die Schaltfläche WEITER programmieren.

In der `Select Case`-Anweisung wird zuerst die Registerkarte 2 aufbereitet. Das heißt, wenn der momentane Index 1 ist (Registerkarte 2), muss die linke Registerkarte mit dem Index 0 verfügbar gemacht werden (`Enabled = True`), denn beim Klick auf die Schaltfläche ZURÜCK wird in die Registerkarte 1 gewechselt (Index 0). Die Registerkarte 2, die im Moment noch aktiv ist, muss auf inaktiv gesetzt werden. Mittels `Value = 0` wird die Registerkarte 1 selektiert. Beim Wechsel auf die Registerkarte 1 muss zudem die Schaltfläche ZURÜCK deaktiviert werden.

Wenn der zweite Fall eintritt (Case 2), dann bedeutet das, dass momentan die dritte Registerkarte aktiv ist. Per Klick auf die Schaltfläche ZURÜCK würde somit in die zweite Registerkarte gewechselt. Das heißt, dass die zweite Registerkarte `Pages(1)` aktiviert werden muss. Die momentan aktive Registerkarte 3 `Pages(2)` muss deaktiviert werden. Die Registerkarte 2 `Value = 1` wird selektiert. Die Schaltfläche WEITER wird aktiviert.

```
'=====
' Auf CD      Buchdaten\Beispiele\10_UserForm\
' Dateiname   10_01_UserForm.xls
' UserForm    frm_13_MultiPage
'=====

Private Sub CommandButton1_Click()
    With MultiPage1
        Select Case .Value
            Case 1
                .Pages(0).Enabled = True      ' Bei Registerkarte = 2:
                .Pages(1).Enabled = False      ' Register 1 = aktiv
                .Value = 0                     ' Register 2 = inaktiv
                CommandButton1.Enabled = False ' Register 1 selektieren
                                                ' "Weiter" deaktivieren

            Case 2
                .Pages(1).Enabled = True      ' Bei Registerkarte = 3:
                .Pages(2).Enabled = False      ' Register 2 = aktiv
                .Value = 1                     ' Register 3 = inaktiv
                CommandButton2.Enabled = True  ' Register 2 selektieren
                                                ' "Zurück" aktivieren
        End Select
    End With
End Sub
```

In der nächsten Prozedur wird der Code für die Schaltfläche WEITER hinterlegt. Das Verfahren ist das Gleiche wie in der vorangegangenen Prozedur, mit dem Unterschied, dass wir uns diesmal per Klick auf die Schaltfläche nach rechts bewegen. Die weitere Beschreibung finden Sie direkt im Code.

```
'=====
' Auf CD      Buchdaten\Beispiele\10_UserForm\
' Dateiname   10_01_UserForm.xls
' UserForm    frm_13_MultiPage
'=====

Private Sub CommandButton2_Click()
    With MultiPage1
        Select Case .Value
            Case 0
                ' Bei Registerkarte = 1:
                .Pages(0).Enabled = False    ' Register 1 = inaktiv
                .Pages(1).Enabled = True      ' Register 2 = aktiv
                .Value = 1                    ' Register 2 selektieren
                CommandButton1.Enabled = True ' "Zurück" aktivieren

            Case 1
                ' Bei Registerkarte = 2:
                .Pages(1).Enabled = False    ' Register 2 = inaktiv
                .Pages(2).Enabled = True      ' Register 3 = aktiv
                .Value = 2                    ' Register 3 selektieren
                CommandButton2.Enabled = False ' "Weiter" deaktivieren
        End Select
    End With
End Sub
```

In einem letzten Schritt programmieren wir die Schaltfläche IN TABELLE SCHREIBEN. Die beiden Einträge, die in die Textfelder NAME und VORNAME geschrieben wurden, werden in die Zellen H1 und I1 übertragen.

Der zweite Codeblock widmet sich den Optionsfeldern aus der zweiten Registerkarte. In einer If-Entscheidung wird geprüft, welches Feld aktiviert wurde. Es wird eine entsprechende Hintergrundfarbe an den Bereich H1:I1 übergeben.

Im letzten Codeblock werden die Kontrollkästchen auf der dritten Registerkarte überprüft. Dazu verwenden wir ebenfalls eine If-Entscheidung. Da bei Kontrollkästchen auch eine Mehrfachauswahl möglich ist, müssen alle möglichen Kombinationen abgefragt werden. Je nachdem, ob die Felder aktiv sind oder nicht, wird der Schriftschnitt Fett oder/und Kursiv zugewiesen.

```
'=====
' Auf CD      Buchdaten\Beispiele\10_UserForm\
' Dateiname   10_01_UserForm.xls
' UserForm    frm_13_MultiPage
'=====

Private Sub CommandButton3_Click()
    With Worksheets("Tabelle1")
        ' Name und Vorname aus Registerkarte 1
        .Range("H1").Value = TextBox1.Value
        .Range("I1").Value = TextBox2.Value

        With .Range("H1:I1")

            ' Hintergrundfarbe aus Registerkarte 2
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

```

    If OptionButton1.Value = True Then
        .Interior.ColorIndex = 3
    ElseIf OptionButton2.Value = True Then
        .Interior.ColorIndex = 4
    ElseIf OptionButton3.Value = True Then
        .Interior.ColorIndex = 5
    End If

    ' Schriftschnitt aus Registerkarte 3
    If CheckBox1.Value = True And _
        CheckBox2.Value = False Then
        With .Font
            .Bold = True
            .Italic = False
        End With

    ElseIf CheckBox1.Value = True And _
        CheckBox2.Value = True Then
        With .Font
            .Bold = True
            .Italic = True
        End With

    ElseIf CheckBox1.Value = False And _
        CheckBox2.Value = True Then
        With .Font
            .Bold = False
            .Italic = True
        End With

    ElseIf CheckBox1.Value = False And _
        CheckBox2.Value = False Then
        With .Font
            .Bold = False
            .Italic = False
        End With
    End If
End With
End With
End Sub

```

342 Bildlaufleisten (ScrollBar)

In unserem UserForm verwenden wir drei Bildlaufleisten, sechs Bezeichnungsfelder und die Schaltfläche ABBRECHEN. Jede der Bildlaufleisten ist der Schieber für eine Farbe: Rot, Grün oder Blau. Es werden RGB-Farbwerte für die Anzeige verwendet. Beim Verschieben der Bildlaufleisten ändert sich automatisch die Farbe des zugehörigen Bezeichnungsfeldes, das sich unterhalb der Bildlaufleiste befindet. Der darin enthaltene Wert zeigt den momentan eingestellten Farbwert an.

Jeder der Bildlaufleisten hinterlegen wir ein `Change`-Ereignis. In der ersten `With`-Anweisung wird ein Minimalwert von 0 und ein Maximalwert von 255 für die Bildlaufleiste festgelegt. Dies entspricht dem Bereich 0 bis 255, den ein RGB-Wert annehmen kann.



Abbildung 281: Bildlaufleisten

In der zweiten With-Anweisung wird dem Bezeichnungsfeld der Wert der Bildlaufleiste zugewiesen. Zudem erhält das Bezeichnungsfeld die entsprechende Hintergrundfarbe.

Bildlaufleiste für die rote Farbe:

```
'=====
' Auf CD      Buchdaten\Beispiele\10_UserForm\
' Dateiname  10_01_UserForm.xls
' UserForm   frm_14_ScrollBar
'=====

Private Sub ScrollBar1_Change()
    With ScrollBar1
        .Min = 0
        .Max = 255
    End With

    With Label1
        .Caption = ScrollBar1.Value
        .BackColor = RGB(ScrollBar1.Value, 0, 0)
    End With
End Sub
```

Bildlaufleiste für die grüne Farbe:

```
'=====
' Auf CD      Buchdaten\Beispiele\10_UserForm\
' Dateiname  10_01_UserForm.xls
' UserForm   frm_14_ScrollBar
'=====

Private Sub ScrollBar2_Change()
    With ScrollBar2
        .Min = 0
        .Max = 255
    End With

    With Label2
        .Caption = ScrollBar2.Value
    End With
End Sub
```

Grund-	lagen
Allge-	mein
Datu	Zeit
tungen	
Steuer	elemen
Befehl	leisten
Objekt	
Dia-	gramm
Ereig-	nisse
User-	Form
Web/	Mail
Extern	
Ge-	misch
tes	
Specia	

```

        .BackColor = RGB(0, ScrollBar2.Value, 0)
    End With
End Sub

```

Bildlaufleiste für die blaue Farbe:

```

' =====
' Auf CD      Buchdaten\Beispiele\10_UserForm\
' Dateiname   10_01_UserForm.xls
' UserForm    frm_14_ScrollBar
' =====

Private Sub ScrollBar3_Change()
    With ScrollBar3
        .Min = 0
        .Max = 255
    End With

    With Label3
        .Caption = ScrollBar3.Value
        .BackColor = RGB(0, 0, ScrollBar3.Value)
    End With
End Sub

```

343 Drehfelder (SpinButton)

Drehfelder sind mit den Bildlaufleisten verwandt. Drehfelder zeigen jedoch lediglich zwei Pfeile an. Im Gegensatz dazu ist bei den Bildlaufleisten zwischen den Pfeilen eine Fläche mit einem Schieber vorhanden.

In unserem UserForm befindet sich ein Textfeld, ein Bezeichnungsfeld, ein Drehfeld und die Schaltfläche ABBRECHEN.



Abbildung 282: Drehfeld

Diesmal arbeiten wir mit einer Datumsanzeige. Per Klick auf den Pfeil nach oben soll das Datum um einen Tag erhöht werden. Bei Klick auf den Pfeil nach unten wird der Datumswert um 1 reduziert.

Beim Initialisieren des UserForm wird das heutige Datum in das Textfeld geladen. Damit das Datum korrekt angezeigt wird, werden wir alle Datumsangaben entsprechend umwandeln (CDate).

```

' =====
' Auf CD      Buchdaten\Beispiele\10_UserForm\
' Dateiname   10_01_UserForm.xls
' UserForm    frm_15_SpinButton

```

```
'=====
Private Sub UserForm_Initialize()
    TextBox1.Value = CDate(Date)
End Sub
```

Jedem Drehfeld-Pfeil hinterlegen wir eine eigene Ereignis-Prozedur. Für den Pfeil nach oben verwenden wir das Ereignis `SpinUp`. Das Datum wird in dieser Prozedur um den Wert 1 erhöht.

```
'=====
' Auf CD      Buchdaten\Beispiele\10_UserForm\
' Dateiname   10_01_UserForm.xls
' UserForm    frm_15_SpinButton
'=====

Private Sub SpinButton1_SpinUp()
    TextBox1.Value = CDate(TextBox1.Value) + 1
End Sub
```

Für den Pfeil nach unten verwenden wir das Ereignis `SpinDown`. Der Datumswert wird bei Klick auf den Pfeil nach unten um den Wert 1 reduziert.

```
'=====
' Auf CD      Buchdaten\Beispiele\10_UserForm\
' Dateiname   10_01_UserForm.xls
' UserForm    frm_15_SpinButton
'=====

Private Sub SpinButton1_SpinDown()
    TextBox1.Value = CDate(TextBox1.Value) - 1
End Sub
```

Dem Textfeld hinterlegen wir das Ereignis `Change`, das bestimmte Datumsangaben prüft. Wenn das aktuelle Datum im Textfeld angezeigt wird, ändert sich die Hintergrundfarbe in Rot. Zudem wird im Bezeichnungsfeld der Text `HEUTE` angezeigt. Des Weiteren haben wir einige Geburtstage hinterlegt. Falls eines der hinterlegten Daten im Textfeld angezeigt wird, ändert sich die Hintergrundfarbe des Textfeldes in Grün. Im Bezeichnungsfeld wird zudem ein entsprechender Text angezeigt. Der Variablen `intYear` wird das aktuelle Jahr übergeben. Auf diese Weise werden nur im aktuellen Jahr die Geburtstagstreffer hervorgehoben. Zu guter Letzt wird der Schweizer Nationalfeiertag im Textfeld gelb hinterlegt. Das Bezeichnungsfeld zeigt den im Code definierten Text an.

```
'=====
' Auf CD      Buchdaten\Beispiele\10_UserForm\
' Dateiname   10_01_UserForm.xls
' UserForm    frm_15_SpinButton
'=====

Private Sub TextBox1_Change()
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

```

Dim intYear As Integer

intYear = Right(CDate(Date), 4)

With TextBox1
    If .Value = CDate(Date) Then
        .BackColor = RGB(255, 0, 0)
        Label1.Caption = "Heute"
    ElseIf .Value = CDate("01.05." & intYear) Then
        .BackColor = RGB(0, 255, 0)
        Label1.Caption = "Happy Birthday Zülfü"
    ElseIf .Value = CDate("03.12." & intYear) Then
        .BackColor = RGB(0, 255, 0)
        Label1.Caption = "Happy Birthday Melanie"
    ElseIf .Value = CDate("05.09." & intYear) Then
        .BackColor = RGB(0, 255, 0)
        Label1.Caption = "Happy Birthday Monika"
    ElseIf .Value = CDate("01.08." & intYear) Then
        .BackColor = RGB(255, 255, 0)
        Label1.Caption = "Schweizer Nationalfeiertag"
    Else
        .BackColor = RGB(255, 255, 255)
        Label1.Caption = ""
    End If
End With
End Sub

```

344 Anzeige (Image)

Um eine Grafik in einem Anzeigefeld auszugeben, fügen Sie zuerst das Steuerelement ANZEIGE ein. Bei aktivem Steuerelement finden Sie im Eigenschaften-Fenster die Auswahl PICTURE. Aktivieren Sie das Feld und klicken Sie rechterhand auf die Schaltfläche mit den drei Punkten, um das Dialogfenster BILD LADEN anzuzeigen. Wählen Sie die gewünschte Grafik aus. Erlaubt sind folgende Formate:

**.bmp, *.cur, *.gif, *.ico, *.jpg und *.wmf*

In unserem UserForm befinden sich übereinander gelegt drei Anzeigefelder, ein Frame mit drei Optionsfeldern und die Schaltfläche ABBRECHEN.



Abbildung 283: Anzeige

Jedem der drei Anzeigefelder wurde eine unterschiedliche Grafik hinterlegt. Per Klick auf eins der Optionsfelder soll das entsprechende Bild angezeigt werden. Beim Initialisieren des UserForm wird das erste Bild angezeigt. Die restlichen beiden Bilder werden ausgeblendet. Das erste Optionsfeld wird aktiviert.

```
'=====
' Auf CD      Buchdaten\Beispiele\10_UserForm\
' Dateiname  10_01_UserForm.xls
' UserForm   frm_16_Image
'=====
```

```
Private Sub UserForm_Initialize()
    Image1.Visible = True
    Image2.Visible = False
    Image3.Visible = False
    OptionButton1.Value = True
End Sub
```

Nachfolgend finden Sie die Ereignis-Prozedur für das erste Optionsfeld. Als Grafik wird der Assistent »Merlin« angezeigt.

```
'=====
' Auf CD      Buchdaten\Beispiele\10_UserForm\
' Dateiname  10_01_UserForm.xls
' UserForm   frm_16_Image
'=====
```

```
Private Sub OptionButton1_Click()
    Image1.Visible = True
    Image2.Visible = False
    Image3.Visible = False
End Sub
```

Die folgende Ereignis-Prozedur steht für das zweite Optionsfeld. Als Grafik wird der Assistent »Minky« angezeigt.

```
'=====
' Auf CD      Buchdaten\Beispiele\10_UserForm\
' Dateiname  10_01_UserForm.xls
' UserForm   frm_16_Image
'=====
```

```
Private Sub OptionButton2_Click()
    Image1.Visible = False
    Image2.Visible = True
    Image3.Visible = False
End Sub
```

Die letzte Ereignis-Prozedur ist an das dritte Optionsfeld gebunden. Als Grafik wird der Assistent »Rocky« angezeigt.

```
'=====
' Auf CD      Buchdaten\Beispiele\10_UserForm\
' Dateiname  10_01_UserForm.xls
' UserForm   frm_16_Image
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

```
' =====
Private Sub OptionButton3_Click()
    Image1.Visible = False
    Image2.Visible = False
    Image3.Visible = True
End Sub
```

345 RefEdit

Das letzte Steuerelement in der Werkzeugsammlung nennt sich sowohl auf deutsch als auch auf englisch `RefEdit`. Mit dem Editor kann auf einen Bereich des Tabellenblattes referenziert werden. Das Feld ähnelt in der Anzeige einer Textbox. Bei geöffnetem Dialogfenster klicken Sie zuerst in das `RefEdit`-Feld und markieren dann den gewünschten Bereich Ihres Tabellenblattes. Im `RefEdit`-Feld wird die Adresse des markierten Bereiches angezeigt. In unserem Beispiel können Sie dem markierten Bereich eine Hintergrundfarbe zuweisen. Aktivieren Sie dazu das gewünschte Optionsfeld und klicken Sie auf die Schaltfläche `ANWENDEN`. Die Schaltfläche `ENTFERNEN` dient dazu, die Farben im markierten Bereich zu entfernen.

In unserem UserForm sind ein `RefEdit`-Feld, vier Optionsfelder und drei Schaltflächen vorhanden.



Abbildung 284: `RefEdit`

Beim Initialisieren des UserForm wird das Optionsfeld für die rote Farbe aktiviert.

```
' =====
' Auf CD      Buchdaten\Beispiele\10_UserForm\
' Dateiname  10_01_UserForm.xls
' UserForm   frm_17_RefEdit
' =====

Private Sub UserForm_Initialize()
    OptionButton1.Value = True
End Sub
```

Die Schaltfläche `ANWENDEN` soll die Farbe auf den Tabellenbereich übertragen, der mittels `RefEdit` markiert wurde. In der `If`-Entscheidung wird geprüft, ob der Wert des `RefEdit`-Feldes leer ist. Wenn dies zutrifft, wird die Prozedur verlassen. Dieser Schritt ist erforderlich, da sonst der Debugger gestartet würde, falls kein Bereich markiert wurde. Nach der `If`-Entscheidung wird der markierte Bereich referenziert.

In der `For`-Schleife wird jeder Zelle des markierten Bereiches die über das Optionsfeld ausgewählte Farbe zugewiesen. Die `If`-Entscheidung innerhalb der `For`-Schleife prüft, welches Optionsfeld aktiviert wurde.

Am Ende der Prozedur wird das RefEdit-Feld geleert, indem der im Feld enthaltene Wert entfernt wird (Value = ""). Danach wird der Cursor wieder in das RefEdit-Feld gesetzt (SetFocus).

```
'=====
' Auf CD      Buchdaten\Beispiele\10_UserForm\
' Dateiname  10_01_UserForm.xls
' UserForm   frm_17_RefEdit
'=====

Private Sub CommandButton1_Click()
    Dim rngRef As Range
    Dim c As Range

    If RefEdit1.Value = "" Then
        Exit Sub
    End If

    Set rngRef = Range(RefEdit1.Value)

    For Each c In rngRef.Rows
        If OptionButton1.Value = True Then
            rngRef.Interior.ColorIndex = 3      ' rot
        ElseIf OptionButton2.Value = True Then
            rngRef.Interior.ColorIndex = 4      ' grün
        ElseIf OptionButton3.Value = True Then
            rngRef.Interior.ColorIndex = 5      ' blau
        ElseIf OptionButton4.Value = True Then
            rngRef.Interior.ColorIndex = 6      ' gelb
        End If
    Next c

    With RefEdit1
        .Value = ""
        .SetFocus
    End With
End Sub
```

Die Prozedur, die der Schaltfläche ENTFERNEN hinterlegt ist, prüft ebenfalls zu Beginn, ob das RefEdit-Feld leer ist. Wenn ja, wird die Prozedur verlassen. Wenn der Bereich nicht leer ist, werden die darin enthaltenen Hintergrundfarben entfernt. Am Ende der Prozedur wird das RefEdit-Feld neu aufbereitet.

```
'=====
' Auf CD      Buchdaten\Beispiele\10_UserForm\
' Dateiname  10_01_UserForm.xls
' UserForm   frm_17_RefEdit
'=====

Private Sub CommandButton2_Click()
    If RefEdit1.Value = "" Then
        Exit Sub
    End If
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

```
Range(RefEdit1.Value).Interior.ColorIndex = xlNone

With RefEdit1
    .Value = ""
    .SetFocus
End With
End Sub
```

Achtung

Verwenden Sie für das Anzeigen eines UserForms (Methode Show), in welchem sich ein RefEdit-Feld befindet, nicht die Konstante. Das RefEdit-Feld selbst bedient sich bereits eines Editier-Modus, der bei geöffnetem UserForm auf die Tabelle zugreift. Ein erneutes Verwenden könnte die gesamte Anwendung blockieren.

346 Position und Größe eines UserForm

Die Größe eines UserForm wird über die Eigenschaften Width Breite und Height Höhe festgelegt. Die Position können Sie über die Eigenschaften Left Links und Top Oben bestimmen. Wenn Sie beiden Eigenschaften den Wert 0 übergeben, wird das UserForm in der linken oberen Ecke der Excel-Anwendung angezeigt. Damit das Ganze funktioniert, muss die Eigenschaft StartUpPosition auf 0 (Null) gesetzt werden. Damit stellen Sie sicher, dass Excel keine eigenen Startwerte übergibt. Weitere mögliche Werte sind:

Wert	Beschreibung
0	Es werden keine Startwerte übergeben
1	Das UserForm wird auf dem Element zentriert, zu dem es gehört
2	Bildschirmmitte
3	In der oberen linken Ecke des Bildschirms anzeigen

Tabelle 81: Werte für die Startposition

Auf unserem UserForm befindet sich lediglich ein Umschaltfeld. Wenn das Umschaltfeld angeklickt wird, verändert sich die Größe und Position des UserForm. Die Beschriftung des Umschaltfeldes zeigt je nachdem die Beschriftung VERGRÖßERN oder VERKLEINERN an.

```
'=====
' Auf CD      Buchdaten\Beispiele\10_UserForm\
' Dateiname   10_02_UserForm_Divers.xls
' UserForm    frm_01_SizeAndPosition
'=====

Private Sub ToggleButton1_Click()
    frm_01_SizeAndPosition.StartUpPosition = 0

    With frm_01_SizeAndPosition
        Select Case ToggleButton1
            Case True
                .Left = 100
```

```

.Top = 100
.Width = 400
.Height = 400
ToggleButton1.Caption = "Verkleinern"
Case False
.Left = 200
.Top = 200
.Width = 200
.Height = 200
ToggleButton1.Caption = "Vergrößern"
End Select
End With
End Sub

```

347 UserForm dynamisch in der Größe verändern

Hin und wieder trifft man auf Dialogfenster, die sich auf- und zuklappen lassen. Dies zu realisieren ist recht einfach. In unserem UserForm haben wir dazu eigens eine Schaltfläche eingefügt, die einen Pfeil aufweist. In den Abbildung 285 ist die Schaltfläche etwas dunkler als die anderen Elemente dargestellt.

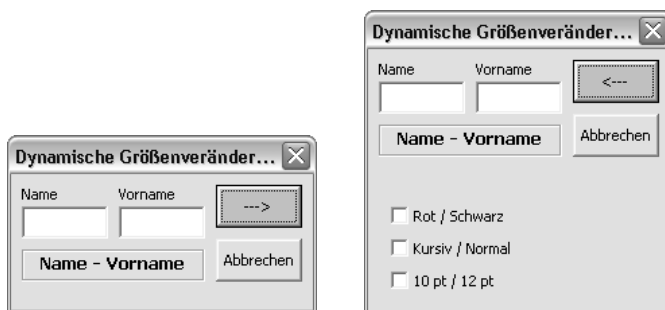


Abbildung 285: Verkleinertes und vergrößertes UserForm

Der Schaltfläche wird eine Prozedur hinterlegt, die einerseits die Beschriftung der Schaltfläche abfragt und andererseits die Beschriftung verändert. Wenn die Schaltfläche als Beschriftung einen Pfeil nach rechts ---> aufweist, soll die Prozedur bei Klick das Fenster vergrößern und die Beschriftung in einen Pfeil nach links <--- umwandeln. Bei erneutem Klick werden die Pfeile wieder vertauscht und das Fenster verkleinert.

```

' =====
' Auf CD   Buchdaten\Beispiele\10_UserForm\
' Dateiname 10_02_UserForm_Divers.xls
' UserForm  frm_02_DynamicSize
' =====

```

```

Private Sub CommandButton1_Click()
    If CommandButton1.Caption = "--->" Then
        frm_02_DynamicSize.Height = 170
        CommandButton1.Caption = "<---"
    ElseIf CommandButton1.Caption = "<---" Then

```

Grundlagen

Allgemein

Datum
Zeit

tungen

Steuer
elementeBefehl
leisten

Objekt

Diagramm

Ereignisse

User-
FormWeb/
Mail

Extern

Gemischtes

Spezial

```

    frm_02_DynamicSize.Height = 100
    CommandButton1.Caption = "--->"
End If
End Sub

```

348 UserForm aus UserForm aufrufen

In diesem Rezept geht es hauptsächlich darum, aus einem UserForm heraus ein anderes UserForm aufzurufen und dann wieder zum ersten UserForm zurückzukehren. Zusätzlich sind die beiden UserForms mit verschiedenen Steuerelementen versehen. Im ersten UserForm sind ein Frame, vier Optionsfelder, vier Anzeigen (Bilder/Image) und drei Schaltflächen enthalten. Im zweiten UserForm sind zwei Rahmen, acht Optionsfelder, ein Textfeld, eine Bildlaufleiste und zwei Schaltflächen enthalten.

Diesmal haben wir jedem Steuerelement einen selbsterklärenden Namen zugewiesen. Sie können so besser voneinander unterschieden werden. Bei vordefinierten Namen wie `OptionsButton1`, `OptionsButton2` usw. wäre nur schwer zu erkennen, welches Optionsfeld im Code wozu verwendet wird. Bei sprechenden Namen wie z.B. `opt_Red` kann davon ausgegangen werden, dass das Optionsfeld mit der Farbe Rot zu tun hat.

Die Bilder, die den Anzeigefeldern des ersten UserForm zugewiesen wurden, finden Sie auf der CD im selben Verzeichnis, wie die Beispieldatei. Die Bilder entsprechen den AutoFormen, welche in der Symbolleiste ZEICHNEN zu finden sind.



Abbildung 286: UserForm aus UserForm heraus aufrufen

Für die Schaltfläche EINFÜGEN, die sich auf dem ersten UserForm befindet, haben wir ein `Click`-Ereignis hinterlegt, das entsprechend des ausgewählten Optionsfeldes im Tabellenblatt ein entsprechendes AutoForm einfügt. Mehr zum Thema AutoFormen können Sie in der Kategorie »Objekte« nachlesen.

```

' =====
' Auf CD      Buchdaten\Beispiele\10_UserForm\
' Dateiname   10_02_UserForm_Divers.xls
' UserForm    frm_03_FirstUF
' =====

```

```
Private Sub cmb_Insert_Click()  
    Dim shp As Shape  
  
    If opt_Rectangle.Value = True Then  
        ' Viereck  
        Set shp = ActiveSheet.Shapes.AddShape _  
            (msoShapeRectangle, _  
             Left:=150, Top:=50, _  
             Width:=50, Height:=50)  
  
    ElseIf opt_Oval.Value = True Then  
        ' Kreis  
        Set shp = ActiveSheet.Shapes.AddShape _  
            (msoShapeOval, _  
             Left:=150, Top:=50, _  
             Width:=50, Height:=50)  
  
    ElseIf opt_Balloon.Value = True Then  
        ' Sprechblase  
        Set shp = ActiveSheet.Shapes.AddShape _  
            (msoShapeRoundedRectangularCallout, _  
             Left:=150, Top:=50, _  
             Width:=50, Height:=50)  
  
    ElseIf opt_Star.Value = True Then  
        ' Stern  
        Set shp = ActiveSheet.Shapes.AddShape _  
            (msoShape5pointStar, _  
             Left:=150, Top:=50, _  
             Width:=50, Height:=50)  
  
    End If  
End Sub
```

Die zweite Prozedur wurde der Schaltfläche BEARBEITEN >> zugewiesen. Sie veranlasst, dass das zweite UserForm geöffnet wird.

```
'=====
```

```
' Auf CD      Buchdaten\Beispiele\10_UserForm\  
' Dateiname  10_02_UserForm_Divers.xls  
' UserForm   frm_03_FirstUF  
'=====
```

```
Private Sub cmb_Modify_Click()  
    frm_04_SecondUF.Show  
End Sub
```

Im zweiten UserForm ist jedem der Optionsfelder ein Click-Ereignis hinterlegt. Die ersten vier Prozeduren gelten der Linienfarbe des AutoForms, das zuvor im Tabellenblatt eingefügt wurde. Nachfolgend finden Sie die Prozedur für die rote Linienfarbe. Die Prozeduren für die weiteren Linienfarben sehen genauso aus, wie diese, mit der einzigen Ausnahme, dass der RGB-Wert ein

Grundlagen
Allgemein
Datum und Zeit
Diagramme
Steuerungselemente
Befehlslisten
Objekte
Diagramme
Ereignisse
UserForm
Web/Mail
Extern
Gemischtes
Spezial

anderer ist. Wir haben darauf verzichtet, alle Prozeduren hier abzubilden, da sie, wie gesagt, nahezu identisch sind.

```
'=====
' Auf CD      Buchdaten\Beispiele\10_UserForm\
' Dateiname  10_02_UserForm_Divers.xls
' UserForm   frm_04_SecondUF
'=====

Private Sub opt_LineRed_Click()
    ActiveSheet.Shapes.Range(ActiveSheet.Shapes.Count) _
        .Line.ForeColor.RGB = RGB(255, 0, 0)
End Sub
```

Die nachfolgende Prozedur zeigt eine von insgesamt vier Prozeduren, die den Optionsfeldern für die Hintergrundfarbe zugewiesen sind. Diese Prozedur weist dem AutoForm, welches zuvor ins Tabellenblatt eingefügt wurde, eine rote Hintergrundfarbe zu. Auch hier haben wir auf die Darstellung der drei weiteren Prozeduren verzichtet, da sich ebenfalls, bis auf den RGB-Farbwert, nichts ändert.

```
'=====
' Auf CD      Buchdaten\Beispiele\10_UserForm\
' Dateiname  10_02_UserForm_Divers.xls
' UserForm   frm_04_SecondUF
'=====

Private Sub opt_FillRed_Click()
    ActiveSheet.Shapes.Range(ActiveSheet.Shapes.Count) _
        .Fill.ForeColor.RGB = RGB(255, 0, 0)
End Sub
```

Die nachfolgende Prozedur steht für die Bildlaufleiste, über welche die Transparenz der Füllfarbe des AutoForm verändert werden kann. Falls mehrere Objekte auf dem Tabellenblatt vorhanden sind, wird jeweils das zuletzt eingefügte durch die Prozedur angesprochen.

```
'=====
' Auf CD      Buchdaten\Beispiele\10_UserForm\
' Dateiname  10_02_UserForm_Divers.xls
' UserForm   frm_04_SecondUF
'=====

Private Sub scb_Transparency_Change()
    scb_Transparency.Min = 0
    scb_Transparency.Max = 100

    ' Der Wert der Bildlaufleiste wird in Prozent
    ' als Transparenz dem Shape übergeben.
    ActiveSheet.Shapes.Range(ActiveSheet.Shapes.Count) _
        .Fill.Transparency = scb_Transparency.Value / 100
```

```
' Dem Textfeld wird der aktuelle Wert der
' Bildlaufleiste übergeben
lbl_Transparency.Caption = scb_Transparency.Value & _
    "% Transparenz:"
End Sub
```

Die Prozedur, welche an die Schaltfläche << ZURÜCK des zweiten UserForm gebunden ist, bewirkt, dass das zweite UserForm verlassen und zum ersten UserForm zurückgekehrt wird.

```
' =====
' Auf CD      Buchdaten\Beispiele\10_UserForm\
' Dateiname   10_02_UserForm_Divers.xls
' UserForm    frm_04_SecondUF
' =====

Private Sub cmb_Cancel_Click()
    Unload Me
End Sub
```

Beim Klick auf die Schaltfläche ABBRECHEN werden beide UserForms verlassen. Zuerst muss das zweite und dann das erste UserForm geschlossen werden.

```
' =====
' Auf CD      Buchdaten\Beispiele\10_UserForm\
' Dateiname   10_02_UserForm_Divers.xls
' UserForm    frm_04_SecondUF
' =====

Private Sub cmb_Cancel_Click()
    Unload frm_04_SecondUF
    Unload frm_03_FirstUF
End Sub
```

349 Landkarte mit Quickinfo

In diesem Rezept wird die Landkarte der Schweiz dargestellt. Die einzelnen Kantone auf der Landkarte sind mit Abkürzungen gekennzeichnet. Wenn mit dem Mauszeiger auf eine der Abkürzungen gezeigt (nicht geklickt) wird, wird eine Quickinfo angezeigt. Die Schweiz ist ein viersprachiges Land. Der Quickinfo ist der ausgeschriebene Name des Kantons sowie die dort am häufigsten gesprochene Sprache zu entnehmen.

Sie können einem UserForm direkt ein Bild zuweisen. Aktivieren Sie das UserForm. Im Eigenschaften-Fenster finden Sie die Eigenschaft PICTURE. Klicken Sie rechts neben der Eigenschaft PICTURE auf die Schaltfläche, die drei Punkte anzeigt. Sie können nun aus dem Dialogfenster BILD LADEN die gewünschte Bild-Datei auswählen. Die Grafik für dieses Beispiel befindet sich auf der CD im selben Verzeichnis wie die Beispielmappe.

Was jetzt folgt, ist Fleißarbeit. Über jede Abkürzung, die auf der Grafik zu sehen ist, legen wir ein Bezeichnungsfeld und vergeben ihm einen Namen, wie lbl_AG, lbl_AI usw. Damit sind die Kantone eindeutig zu erkennen und es fällt uns später leichter, den Programmcode dazu zu schreiben, ohne die Kantone durcheinander zu bringen.

Grundlagen

Allgemein

Datum
Zeit

ungen

Steuer
elemente

Befehl
leisten

Objekt

Diagramm

Ereignisse

User-
Form

Web/
Mail

Extern

Gemischtes

Spezial

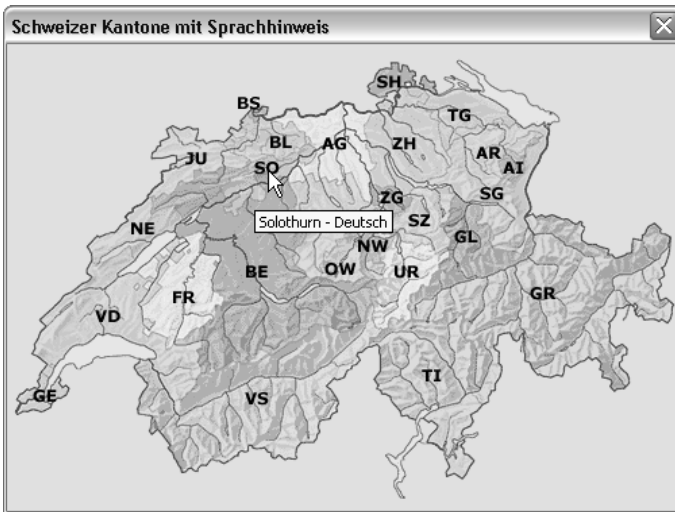


Abbildung 287: Grafik mit Quickinfos

Die Bezeichnungsfelder dürfen weder eine Beschriftung `Caption` noch eine Hintergrundfarbe oder einen Rahmen aufweisen, um den gewünschten Effekt zu erzielen. Sie sollen über der Grafik nicht sichtbar sein. Nur die Quickinfo soll angezeigt werden, wenn mit dem Mauszeiger auf einen der Kantone gezeigt wird. Am einfachsten ist es, wenn Sie zuerst ein einziges Bezeichnungsfeld einfügen. Entfernen Sie dann im Eigenschaften-Fenster den Namen `Caption`. Ändern Sie zudem die Eigenschaft `BackStyle` auf `0 - fmBackStyleTransparent`. Damit ist die darunter liegende Grafik sichtbar. Das Bezeichnungsfeld können Sie nun beliebig oft kopieren.

In der VBA-Prozedur weisen wir jedem Bezeichnungsfeld eine Quickinfo `ControlTipText`, gefolgt vom gewünschten Text, zu.

```
' =====
' Auf CD      Buchdaten\Beispiele\10_UserForm\
' Dateiname  10_02_UserForm_Divers.xls
' UserForm   frm_05_Map
' =====
```

```
Private Sub UserForm_Initialize()
    lbl_AG.ControlTipText = "Aargau - Deutsch"
    lbl_AI.ControlTipText = "Appenzell Innerrhoden - Deutsch"
    lbl_AR.ControlTipText = "Appenzell Ausserrhoden - Deutsch"
    lbl_BE.ControlTipText = "Bern - Deutsch"
    lbl_BL.ControlTipText = "Basel Land - Deutsch"
    lbl_BS.ControlTipText = "Basel Stadt - Deutsch"
    lbl_FR.ControlTipText = "Friburg - Französisch"
    lbl_GE.ControlTipText = "Genf - Französisch"
    lbl_GL.ControlTipText = "Glarus - Deutsch"
    lbl_GR.ControlTipText = "Graubünden - Deutsch / Romanisch"
    lbl_JU.ControlTipText = "Jura - Französisch"
    lbl_LU.ControlTipText = "Luzern - Deutsch"
    lbl_NE.ControlTipText = "Neuenburg - Französisch"
    lbl_NW.ControlTipText = "Nidwalden - Deutsch"
```

1bl_OW.ControlTipText = "Obwalden - Deutsch"	Grund- lagen
1bl_SG.ControlTipText = "St. Gallen - Deutsch"	
1bl_SH.ControlTipText = "Schaffhausen - Deutsch"	
1bl_SO.ControlTipText = "Solothurn - Deutsch"	
1bl_SZ.ControlTipText = "Schwyz - Deutsch"	
1bl_TG.ControlTipText = "Thurgau - Deutsch"	
1bl_TI.ControlTipText = "Tessin - Italienisch"	
1bl_UR.ControlTipText = "Uri - Deutsch"	
1bl_VD.ControlTipText = "Waadtland - Französisch"	
1bl_VS.ControlTipText = "Wallis - Deutsch"	
1bl_ZG.ControlTipText = "Zug - Deutsch"	
1bl_ZH.ControlTipText = "Zürich - Deutsch"	
End Sub	

350 Passwortheingabe

In einem UserForm ist es möglich, ein Textfeld zu erstellen, das für eine Passwortheingabe gedacht ist. Der Text, der darin eingegeben wird, kann während der Eingabe nicht entziffert werden, da beim Schreiben in das Textfeld lediglich Sternchen (*) angezeigt werden.

In unserem Beispiel wird zuerst nur ein Teil des UserForm angezeigt, nämlich der Teil, welcher die Aufforderung zur Passwortheingabe enthält. Bei korrekter Passwortheingabe wird das gesamte UserForm angezeigt. Alternativ zu einem aufklappbaren UserForm könnten Sie auch ein zweites UserForm anzeigen lassen.



Abbildung 288: Textfeld mit Passwortheingabe

Aktivieren Sie zuerst das Textfeld, um das entsprechende Eigenschaften-Fenster anzeigen zu lassen. Die Einstellung für die Anzeige der Sternchen kann im Eigenschaften-Fenster neben der Eigenschaft PasswordChar vorgenommen werden. Geben Sie in das Feld ein Sternchen ein. Sie können nach Belieben auch ein anderes Zeichen verwenden.

Wir arbeiten in unserem Beispiel mit zwei Ereignis-Prozeduren. Das erste Ereignis UserForm_Initialize stellt sicher, dass beim Start des UserForm nur der erste Teil sichtbar ist. Dazu legen wir die entsprechende Höhe Height fest.

' =====	
' Auf CD	Buchdaten\Beispiele\10_UserForm\
' Dateiname	10_02_UserForm_Divers.xls
' UserForm	frm_06_Password
' =====	

Grund- lagen
Allge- mein
Datu Zeit
tungen
Steuer elemen
Befehl leisten
Objekt
Dia- gramm
Ereig- nisse
User- Form
Web/ Mail
Extern
Ge- misch tes
Specia

```
Private Sub UserForm_Initialize()
    frm_06_Password.Height = 110
End Sub
```

In der zweiten Prozedur wird in einer If-Entscheidung das Passwort, das in das Textfeld eingegeben wurde, geprüft. Es lautet »codebook«. Die Eingabe ist casesensitiv, das bedeutet, dass auf Groß- und Kleinschreibung geachtet wird. Wenn ein falsches Passwort eingegeben wird, wird eine Fehlermeldung MsgBox angezeigt. Der Inhalt der Textbox wird entfernt und der Cursor wieder im Textfeld platziert. Bei korrekter Eingabe wird die Größe des UserForm so verändert, dass der gesamte Inhalt sichtbar wird.

```
' =====
' Auf CD      Buchdaten\Beispiele\10_UserForm\
' Dateiname  10_02_UserForm_Divers.xls
' UserForm   frm_06_Password
' =====

Private Sub CommandButton1_Click()
    If TextBox1.Value <> "codebook" Then
        MsgBox "Falsch Eingabe."
        TextBox1.Value = ""
        TextBox1.SetFocus
    Else
        frm_06_Password.Height = 150
    End If
End Sub
```

351 Automatisches Ausrichten von Elementen

Oftmals kann es sich als recht mühsam erweisen, viele Elemente in einem UserForm exakt zu positionieren. Der VBA-Editor bietet Unterstützung an. Markieren Sie die Elemente, die ausgerichtet werden sollen. Klicken Sie auf den Menüpunkt **FORMAT | AUSRICHTEN** und wählen Sie die gewünschte Ausrichtung. Im Menü **FORMAT** sind noch weitere nützliche Auswahlmöglichkeiten enthalten, die helfen, Größe, Abstand usw. genau vorzunehmen.

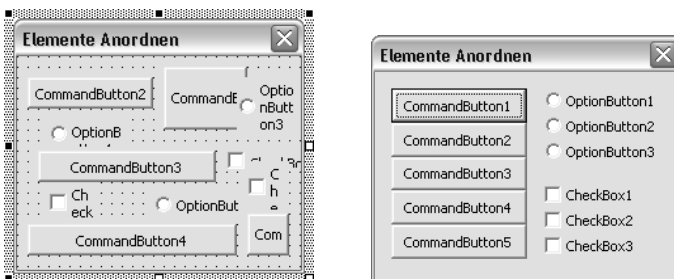


Abbildung 289: Elemente vor und nach der Ausrichtung

Oftmals wäre es jedoch einfacher, man könnte zusätzlich die Elemente automatisch gruppiert nach Typ ausrichten. In unserem Beispiel werden sämtliche Befehlsschaltflächen am linken Rand und untereinander angeordnet. Sie sollen zudem eine einheitliche Größe erhalten. Die Options-

felder und Kontrollkästchen sollen rechts neben den Befehlsschaltflächen erscheinen. Sie werden untereinander und getrennt nach Typ angeordnet. Auch hier bestimmen wir zudem die Größe (siehe Abbildung 289).

In der VBA-Prozedur verwenden wir drei Zähler, für jede Sorte an Elementen einen. In der ersten For-Schleife werden alle Steuerelemente Controls, die im UserForm enthalten sind, durchlaufen. In der If-Entscheidung wird geprüft, ob es sich bei dem Element um eine Schaltfläche oder um ein Optionsfeld handelt. Jede Schaltfläche trägt den Namen CommandButton, gefolgt von einer Nummer (sofern die Namen der Steuerelemente nicht geändert wurden). Mittels Instr vergleichen wir die ersten Zeichen des Namens der Schaltfläche auf Übereinstimmung mit der Zeichenfolge CommandButton. Bei einer Übereinstimmung werden der Abstand von oben Top und links Left sowie die Höhe Height und Breite Width zugewiesen. Damit die Elemente exakt untereinander zu liegen kommen, arbeiten wir bei Top mit einem Zähler. Der Zähler wird bei jedem Treffer um eins erhöht. Damit können wir den Abstand von oben berechnen. Der Zähler wird mit der Höhe der Schaltfläche multipliziert. Damit zwischen der ersten Schaltfläche und dem oberen Rand des UserForm ein Abstand entsteht, zählen wir den Wert 10 dazu.

Dasselbe geschieht bei den Optionsfeldern. Damit diese rechts neben den Schaltflächen erscheinen, müssen wir einen entsprechend größeren Abstand zum linken Rand festlegen.

In der zweiten For-Schleife werden die Kontrollkästchen aufbereitet. Eine zweite Schleife ist erforderlich, da die Kontrollkästchen unterhalb der Optionsfelder platziert werden sollen. Für den Abstand von oben müssen wir die Optionsfelder mit einberechnen. Die Anzahl der Optionsfelder ist bekannt, da die Zahl bereits in der Variablen intOpt gespeichert ist. Die Anzahl der Optionsfelder wird mit deren Höhe multipliziert. Der Abstand vom oberen Rand zum ersten Optionsfeld wird addiert. Der daraus berechnete Wert wird mit der Anzahl der Kontrollkästchen, welche mit deren Höhe multipliziert werden, addiert. Auch hier addieren wir den Wert 10, damit ein Abstand zwischen den Optionsfeldern und den Kontrollkästchen entsteht.

```
'=====
' Auf CD      Buchdaten\Beispiele\10_UserForm\
' Dateiname  10_02_UserForm_Divers.xls
' UserForm   frm_07_Arrange
'=====

Private Sub UserForm_Initialize()
    Dim ctrl As Control
    Dim intCmb As Integer
    Dim intOpt As Integer
    Dim intChk As Integer

    For Each ctrl In Me.Controls
        With ctrl
            ' Befehlsschaltflächen
            If Instr(ctrl.Name, "CommandButton") = 1 Then
                .Top = intCmb * 20 + 10 ' Abstand von oben
                .Left = 10              ' Abstand von links
                .Height = 20            ' Höhe
                .Width = 80             ' Breite
                intCmb = intCmb + 1     ' Zähler

                ' Optionsfelder
```

Grundlagen

Allgemein

Datum
Zeit

Diagrammen

Steuerelemente

Befehle
leisten

Objekt

Diagramm

Ereignisse

UserForm

Web/
Mail

Extern

Gemischtes

Spezial

```

    ElseIf InStr(ctrl.Name, "OptionButton") = 1 Then
        .Top = intOpt * 15 + 10
        .Left = 100
        .Height = 15
        .Width = 80
        intOpt = intOpt + 1
    End If
End With
Next ctrl

For Each ctrl In Me.Controls
    ' Kontrollkästchen
    With ctrl
        If InStr(ctrl.Name, "CheckBox") = 1 Then
            .Top = (intOpt * 15 + 10) + (intChk * 15 + 10)
            .Left = 100
            .Height = 15
            .Width = 80
            intChk = intChk + 1
        End If
    End With
Next ctrl
End Sub

```

352 Tabellenblatt in UserForm einbinden

In einer UserForm können Sie ein gesamtes Tabellenblatt abbilden. Dazu fügen Sie der WERKZEUGSAMMLUNG ein standardmäßig nicht angezeigtes Element hinzu. Klicken Sie mit der rechten Maustaste auf die WERKZEUGSAMMLUNG und wählen Sie aus dem Kontextmenü den Eintrag ZUSÄTZLICHE STEUERELEMENTE. Bewegen Sie die Bildlaufleiste nach unten und aktivieren Sie in der Liste den Eintrag MICROSOFT OFFICE SPREADSHEET 9.0 für Office 2000, 10.0 für Office XP und 11.0 für Office 2003. Bestätigen Sie Ihre Auswahl mit OK. In der WERKZEUGSAMMLUNG wird nun ein weiteres Element angezeigt, das ein Excel-Tabellenblatt darstellt (siehe Abbildung 290).

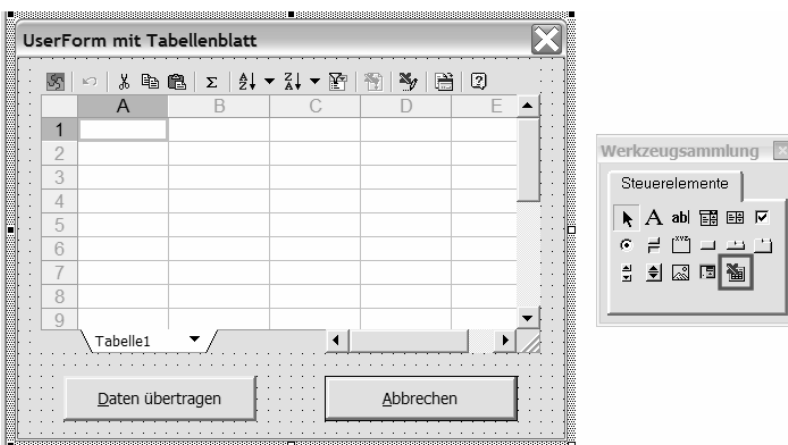


Abbildung 290: UserForm mit Tabellenblatt

Es handelt sich bei dem zusätzlichen Element (siehe Abbildung 290) um eine so genannte *Office-Web-Komponente* (Office Web Components: OWC). Es stehen neben dem Tabellenblatt auch andere Komponenten, wie Diagramme und Pivot-Tabellen, zur Verfügung.

Office-Web-Komponenten sind dazu gedacht, interaktive Webseiten zu erzeugen. Auf interaktiven Webseiten kann online gearbeitet werden. In einem interaktiven Tabellenblatt können beispielsweise online Berechnungen vorgenommen werden. OWC muss auf Ihrem System installiert sein, damit Diagramme in UserForms angezeigt werden können. Diese Komponente ist ab Office 2000 integriert, wird jedoch nicht automatisch installiert. Im Lieferumfang der Small-Business-Version ist OWC nicht enthalten. Wenn Sie eine Office-Web-Komponente auf einem anderen PC benutzen möchten, oder es per E-Mail versenden, muss OWC auch beim Empfänger installiert sein.

Seit der Excel-Version 2002 (XP) sind in einem Tabellenblatt *Spreadsheet*, das in einem UserForm angezeigt wird, 262144 Zeilen und 18278 Spalten verfügbar. Wenn Sie mittels der Tastenkombination **[Strg] + [→]** nach rechts springen, dann gelangen Sie in die Spalte ZZZ. Mit der Tastenkombination **[Strg] + [↓]** gelangen Sie zur letzten Zeile. In der Version 2000 stehen in einem Spreadsheet 702 Spalten ZZ und wie in einem Tabellenblatt einer Excel-Arbeitsmappe 65536 Zeilen zur Verfügung. Das bedeutet, dass in Excel XP einem OWC-Tabellenblatt über 285 mal mehr Zellen zur Verfügung stehen, als in einem gewöhnlichen Tabellenblatt einer Excel-Arbeitsmappe.

Hinweis

Zwischen den Excel-Versionen 2000, 2002 und 2003 gibt es geringfügige Unterschiede, was Menübefehle, Einstellungsmöglichkeiten und Ereignisse des *Spreadsheet*-Objektes angeht. In unserem Beispiel beziehen wir uns auf das *Spreadsheet*-Objekt der Excel-Version 2002.

Die Beispieldatei dieses Rezeptes wurde unter Excel 2002 programmiert und enthält das MICROSOFT OFFICE SPREADSHEET 10.0 sowie einen Verweis auf die entsprechende Bibliothek. Wenn Sie die Beispieldatei mit einer anderen Excel-Version öffnen, erhalten Sie eine Fehlermeldung. Um dennoch den hinterlegten Code anzusehen, deaktivieren Sie vor Dateiföffnung die Makros.

Sie können das verwendete *Spreadsheet* durch eine *Spreadsheet*-Komponente Ihrer Excel-Version ersetzen, um die hinterlegten Prozeduren aufzurufen. Die aufgeführte Ereignisprozedur, welche in diesem Rezept zum Einsatz kommt, ist nur in Excel 2002 lauffähig. Löschen Sie diese Prozedur, oder kommentieren Sie die Codezeilen aus, um eine Fehlermeldung zu unterdrücken.

Im Gegensatz zu einem gewöhnlichen Tabellenblatt einer Excel-Arbeitsmappe fehlen in einem *Spreadsheet* die gewohnten Befehlsleisten, die Bearbeitungszeile und die Statuszeile. Lediglich ein Kontextmenü sowie vereinzelt Funktionen in der Menüleiste stehen im *Spreadsheet*-Objekt zur Verfügung. In der Entwicklungsumgebung können Sie über das Kontextmenü der Zellen den Menüpunkt **BEFEHLE UND OPTIONEN** auswählen. In dem angezeigten Dialog (siehe Abbildung 291) befinden sich auf verschiedenen Registerkarten die Optionen der **STANDARD-** und **FORMAT-Symbolleiste** sowie weitere Funktionen für das Tabellenblatt und die Arbeitsmappe. In der Befehlsleiste finden Sie links neben der **?-Schaltfläche** ebenfalls das Menü **BEFEHLE UND OPTIONEN**. Einstellungen, die Sie in der Entwicklungsumgebung vornehmen, werden beim Aufruf der UserForm im *Spreadsheet* entsprechend angezeigt.

Die UserForm selbst kann in der Entwicklungsumgebung wie gewohnt bearbeitet werden. Das heißt, Sie können nach Belieben Elemente aus der **WERKZEUGSAMMLUNG** hinzufügen. In unserem Beispiel fügen wir die Schaltflächen **DATEN ÜBERTRAGEN** und **ABBRECHEN** ein.

Grundlagen

Allgemein

Datum
Zeit

Diagramme

Steuerungselemente

Befehlsleisten

Objekt

Diagramm

Ereignisse

UserForm

Web/Mail

Extern

Gemischtes

Spezial

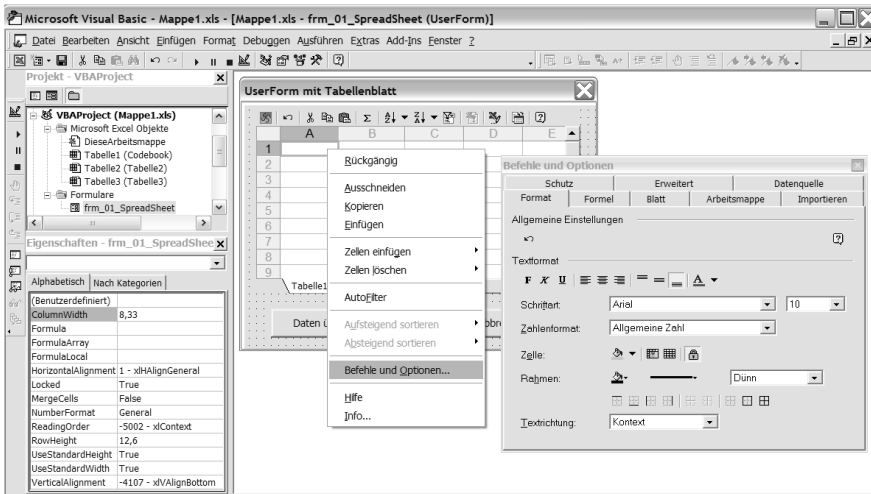


Abbildung 291: Einstellungen für das Spreadsheet festlegen

Per VBA-Code können Sie beispielsweise die Daten eines Excel-Tabellenblattes in ein OWC-Tabellenblatt auf einem Formular kopieren. In der Beispieldatei im Tabellenblatt CODEBOOK wird bei Klick auf die Schaltfläche USERFORM MIT TABELLENBLATT die hinterlegte Ereignis-Prozedur `CmdFrmShow_Click` ausgelöst. In dieser Prozedur wird die Prozedur `MySpreadsheet` aus dem Codemodul `frm_01_SpreadSheet` aufgerufen.

Standardmäßig wird in einem Spreadsheet nur ein Tabellenblatt angezeigt. Weitere Tabellenblätter können durch einen Klick auf den Registerreiter ausgewählt werden. Da die Registerbeschriftungen eines Spreadsheet-Objektes standardmäßig TABELLE1, TABELLE2 usw. lauten, wird der Name des Tabellenblattes über das Schlüsselwort `Me` als Argument an die Prozedur `MySpreadsheet` übergeben, um im Tabellenregister des Spreadsheet-Objektes angezeigt zu werden.

```
' =====
' Auf CD      Buchdaten\Beispiele\10_UserForm\
' Dateiname   10_03_UserForm_OWC_Sheet_XL2002
' Ereignis    Tabelle1 (Codebook)
' =====
```

```
Private Sub CmdFrmShow_Click()
    ' Aufruf der Prozedur »MySpreadsheet« im Codemodul
    ' »frm_01_SpreadSheet« mit Übergabe eines Argumentes
    frm_01_SpreadSheet.MySpreadsheet Me.Name
End Sub
```

Die aufgerufene Prozedur `MySpreadsheet` muss zwingend im Codemodul der UserForm `frm_01_SpreadSheet` hinterlegt und als `Public` deklariert sein, da sie in unserem Beispiel das `UserForm_Initialize`-Ereignis ersetzt. Der Einsatz dieser aufrufenden Prozedur erfolgt unabhängig von dem verwendeten Spreadsheet-Objekt. Diese Vorgehensweise kann bei allen Formularen angewandt werden. Alle Einstellungen, die üblicherweise beim Aufrufen einer UserForm in deren `UserForm_Initialize`-Ereignis vorgenommen werden, werden in der Prozedur `MySpreadsheet` durchgeführt. Wenn Sie neben einer öffentlichen Prozedur zusätzlich das `UserForm_Initialize`-Ereignis

verwenden, wird dieses zuerst und im Anschluss die aufgerufene öffentliche Prozedur ausgeführt. Die weitere Funktionsweise der Prozedur können Sie den Kommentaren im Code entnehmen.

```
' =====
' Auf CD      Buchdaten\Beispiele\10_UserForm\
' Dateiname  10_03_UserForm_OWC_Sheet_XL2002
' UserForm   frm_01_SpreadSheet
' =====

' Prozedur zum Initialisieren und Aufrufen der UserForm
' mit Übergabe von Argumenten

Public Sub MySpreadsheet(strSheetname As String)
    Dim rngCell As Range
    Dim lngRow  As Long
    Dim intCol  As Integer

    ' Verweis auf aktive Zelle des Tabellenblattes
    Set rngCell = ActiveCell

    ' oberste linke sichtbare Zelle des Tabellenblattes ermitteln
    lngRow = ActiveWindow.ScrollRow
    intCol = ActiveWindow.ScrollColumn

    ' Bildschirmaktualisierung ausschalten
    Application.ScreenUpdating = False

    ' Zellen des Tabellenblattes kopieren
    Worksheets(strSheetname).Cells.Copy

    With Me.Spreadsheet1
        ' Registername aus Argument der Aufrufprozedur übertragen
        .ActiveSheet.Name = strSheetname

        ' Daten aus Zwischenablage in Spreadsheet einfügen
        .Range("A1").Paste

        ' aktive Zelle des Tabellenblattes im Spreadsheet markieren
        .Cells(rngCell.Row, rngCell.Column).Select

        ' oberste linke Zelle des Tabellenblattes
        ' im Spreadsheet anzeigen
        .Cells(lngRow, intCol).Show
    End With

    With Application
        ' Kopiermodus im Tabellenblatt ausschalten
        .CutCopyMode = False

        ' Bildschirmaktualisierung einschalten
        .ScreenUpdating = True
    End With
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

```

' UserForm gebunden anzeigen, damit nicht gleichzeitig
' im Tabellenblatt und im UserForm gearbeitet werden kann
Me.Show vbModal
End Sub

```

Wenn eine UserForm mit einem Spreadsheet-Objekt am Bildschirm angezeigt wird, können Sie wie in der Entwicklungsumgebung den Dialog BEFEHLE UND OPTIONEN über die Menüleiste oder das Kontextmenü der Zelle aufrufen. In diesem Dialog (siehe Abbildung 292) stehen weitaus weniger Funktionen und Einstellungen als in der Entwicklungsumgebung zur Verfügung. Die angezeigten Einstellungen beziehen sich auf den markierten Zellenbereich bzw. das ausgewählte Blatt und die Arbeitsmappe. Im Register FORMEL können Sie lediglich den Inhalt der markierten Zelle ansehen. Eine Bearbeitung des Zelleninhaltes ist nur über die Zellen des Spreadsheet auf der UserForm möglich.

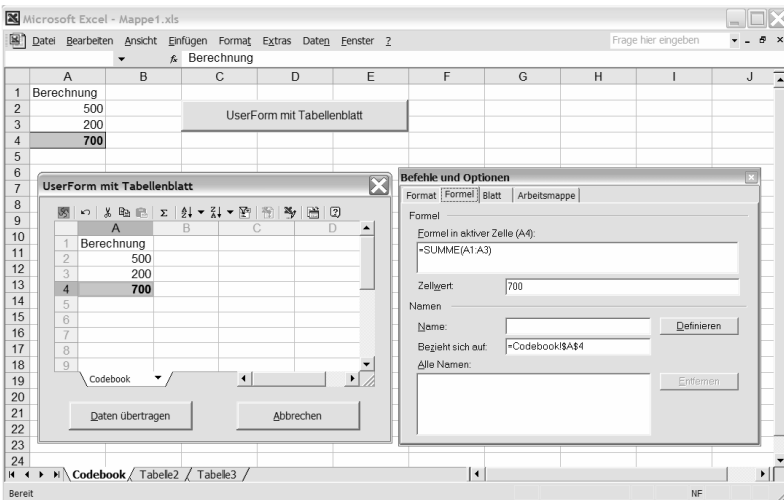


Abbildung 292: Befehle und Optionen des Spreadsheet-Objektes in der Anwendung

Mit einem Klick auf die Schaltfläche DATEN ÜBERTRAGEN der UserForm werden alle Daten und Formate des OWC-Tabellenblattes in das angegebene Tabellenblatt der Excel-Arbeitsmappe übertragen. Bestehende Daten werden dabei überschrieben. Wie Sie der Abbildung 292 entnehmen können, zeigt diese Schaltfläche den ersten Buchstaben D unterstrichen an. Der Accelerator-Eigenschaft wurde dazu im Eigenschaften-Fenster der Schaltfläche der Buchstabe D zugewiesen. Über diese Eigenschaft wird die Zugriffstaste für ein Steuerelement festgelegt. Dadurch können Sie auch mit der Tastenkombination **Alt+D** die hinterlegte Prozedur CmdCopyData_Click aufrufen. Der Schaltfläche ABBRECHEN wurde die Zugriffstaste A zugewiesen. Die Tastenkombination **Alt+A** ruft dementsprechend die hinterlegte Prozedur der Schaltfläche ABBRECHEN auf.

Um nach der Datenübertragung wieder im OWC-Tabellenblatt des UserForm weiterarbeiten zu können, setzen wir den Fokus mit der SetFocus-Methode auf das Spreadsheet-Objekt.

```
'=====
' Auf CD      Buchdaten\Beispiele\10_UserForm\
' Dateiname  10_03_UserForm_OWC_Sheet_XL2002
' UserForm   frm_01_SpreadSheet
'=====

' Daten von Spreadsheet ins Tabellenblatt übertragen

Private Sub CmdCopyData_Click()
' Bildschirmaktualisierung ausschalten
Application.ScreenUpdating = False

' Zellen in Spreadsheet kopieren
Me.Spreadsheet1.ActiveSheet.Cells.Copy

' Daten aus Spreadsheet in Tabellenblatt einfügen
ActiveSheet.Range("A1").PasteSpecial

' erforderlich, um Markierung im Tabellenblatt aufzuheben
ActiveCell.Select

' Fokus auf aktive Zelle des Spreadsheet setzen
Me.Spreadsheet1.SetFocus

' Bildschirmaktualisierung einschalten
Application.ScreenUpdating = True
End Sub
```

Wenn Sie auf die Schaltfläche ABBRECHEN der UserForm klicken oder die Tastenkombination **ALT+A** auf Ihrer Tastatur drücken, wird die hinterlegte Prozedur zum Ausblenden der UserForm aufgerufen.

```
'=====
' Auf CD      Buchdaten\Beispiele\10_UserForm\
' Dateiname  10_03_UserForm_OWC_Sheet_XL2002
' UserForm   frm_01_SpreadSheet
'=====

Private Sub CmdCancel_Click()
Unload Me
End Sub
```

Wie auch die ActiveX-Elemente in einer Excel-Arbeitsmappe weisen die Office-Web-Komponenten verschiedene Ereignisse auf. Je nach verwendeter Excel-Version unterscheidet sich diese Auflistung. Um einen Überblick über die vorhandenen Ereignisse Ihrer Excel-Version zu erhalten, wählen Sie im Codemodul der UserForm im linken DropDown-Feld OBJEKT das SPREADSHEET-Objekt aus. Daraufhin wird im rechten DropDown-Feld PROZEDUR eine Liste der verfügbaren Ereignisse angezeigt. Per Klick auf ein Ereignis wird dessen Codegerüst mit den dazugehörigen Parametern im Codemodul erstellt.

Grund-lagen

Allge-mein

Datu-Zeit

tungen

Steuer-elemen

Befehl-leisten

Objekt

Dia-gramm

Ereig-nisse

User-Form

Web/Mail

Extern

Ge-mischtes

Specia

Achtung

Sobald Sie einer UserForm ein Spreadsheet-Objekt hinzufügen, wird im Menü EXTRAS | VERWEISE ein entsprechender Verweis auf diese Bibliothek gesetzt. In Excel2000 lautet die Bibliothek: *Microsoft Office Web Components 9.0*, in Excel 2002 *Microsoft Office XP Web Components* und in Excel 2003 *Microsoft Office Web Components 11.0*. Wenn Sie keine Ereignisse des Spreadsheet-Objektes in Ihrer Datei anwenden, wird dieser Verweis nicht benötigt.

Es empfiehlt sich aber dennoch nicht, diesen Verweis zu deaktivieren, da unter Umständen die entsprechende Bibliothek anschließend nicht mehr in der Liste der verfügbaren Verweise angezeigt wird. Eine neuerlicher Verweis auf die *Microsoft Office Web Components*-Bibliotheken erweist sich anschließend als sehr aufwendig, indem Sie die entsprechende *.dll über die DURCHSUCHEN-Schaltfläche den Verweisen wieder hinzufügen müssen.

In unserem Beispiel, welches unter Excel 2002 erstellt wurde, verwenden wir das Spreadsheet1_BeforeContextMenu-Ereignis. Dieses Ereignis steht nicht in Excel 2000 zur Verfügung. In Excel 2003 werden die verwendeten Parameter von Datentypen der *Microsoft Office Web Components 11.0*-Bibliothek deklariert. Dadurch ist es nicht möglich, WOC-Elemente in unterschiedlichen Excel-Versionen einzusetzen.

Das Kontextmenü des Spreadsheet1 wird nach einem Rechtsklick auf eine Zelle im WOC-Tabellenblatt nur angezeigt, wenn die aktive Zelle im Spreadsheet1 eine Formel enthält. Es wird ein Meldungsfenster angezeigt und die Anzeige des Kontextmenüs über die Cancel-Eigenschaft deaktiviert, wenn die aktive Zelle keine Formel aufweist. Setzen Sie dazu den Cancel-Parameter des Ereignisses auf True, um die Tastatureingabe abubrechen und das Kontextmenü nicht anzuzeigen.

```
'=====
' Auf CD      Buchdaten\Beispiele\10_UserForm\
' Dateiname   10_03_UserForm_OWC_Sheet_XL2002
' UserForm    frm_01_SpreadSheet
'=====
```

```
' Verweis auf Microsoft Office XP Webcomponents-Bibliothek
' muss bei Verwendung von SpreadSheet-Ereignissen gesetzt sein
```

```
Private Sub Spreadsheet1_BeforeContextMenu( _
    ByVal x As Long, _
    ByVal y As Long, _
    ByVal Menu As OWC10.ByRef, _
    ByVal Cancel As OWC10.ByRef)
```

```
    ' Kontextmenü nur bei Formelzellen einblenden
```

```
    If Not Me.Spreadsheet1.ActiveCell.HasFormula Then
        MsgBox "Die aktive Zelle enthält keine Formel." & _
            vbCrLf & _
            "Das Kontextmenü wurde deaktiviert."
```

```
    ' Kontextmenü deaktivieren
```

```
    Cancel = True
```

```
End If
```

```
End Sub
```

353 Diagramm in UserForm

Um ein OWC-Diagramm in einem UserForm anzuzeigen, müssen Sie der Werkzeugsammlung ein weiteres Element hinzufügen, falls ein solches nicht bereits vorhanden ist. Klicken Sie mit der rechten Maustaste auf die Werkzeugsammlung und wählen Sie aus dem Kontextmenü **ZUSÄTZLICHE STEUERELEMENTE**. Bewegen Sie die Bildlaufleiste nach unten und aktivieren Sie in der Liste **MICROSOFT OFFICE CHART 9.0** für Office 2000, 10.0 für Office XP und 11.0 für Office 2003. Bestätigen Sie mit **OK**. In der Werkzeugsammlung wird nun ein weiteres Element angezeigt, das ein Diagramm darstellt.

Aktivieren Sie das Element und ziehen Sie in Ihrem UserForm einen Rahmen. Es wird ein Feld angezeigt, das ein graues Logo und einen Schriftzug darstellt (siehe Abbildung 293).

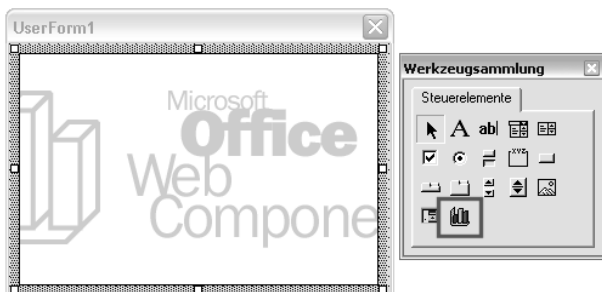


Abbildung 293: UserForm mit Diagramm und Werkzeugsammlung mit neuem Element

Wenn Sie auf das Diagramm-Objekt im UserForm klicken, wird ein Assistent gestartet. Sie haben die Möglichkeit, mittels dieses Assistenten ein Diagramm zu erzeugen oder die Daten aus einer Datenbank zu beziehen. Wir werden hier jedoch nicht mit dem Assistenten arbeiten, sondern das Diagramm mittels VBA-Code erzeugen.

Um ein OWC-Diagramm zu erstellen, arbeiten wir mit so genannten literalen Datenarrays. Die Daten für unser erstes Diagramm werden direkt im VBA-Code aufbereitet. Das nächste Rezept zeigt, wie Daten aus einem Tabellenblatt bezogen werden können.

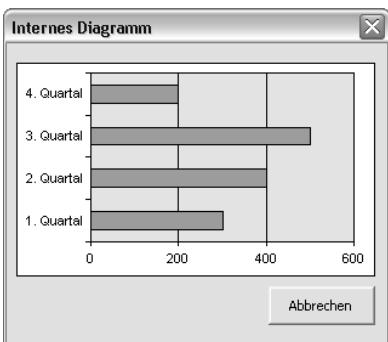


Abbildung 294: UserForm mit Diagramm

Wir verwenden das Ereignis `UserForm_Initialize`. Damit wird das Diagramm direkt beim Aufrufen des UserForm erstellt.

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

In der Variablen-Deklaration hinterlegen wir zwei Arrays, jedes mit dem Wert 4. Damit kann das Array vier Felder aufnehmen. Nach der Deklaration bereiten wir das erste Array auf. Wir übergeben die Werte für die x-Achsen-Beschriftung. Dem zweiten Array übergeben wir die Daten. Achten Sie dabei darauf, dass die Nummerierung des Index mit 0 beginnt.

Die With-Anweisung erzeugt das Diagramm Charts.Add. Innerhalb der With-Anweisung wird der Diagrammtyp bestimmt (chChartTypeBarClustered). Es handelt sich dabei um ein Balkendiagramm. Weitere Diagrammtypen können Sie dem Objekt-Katalog entnehmen. Drücken Sie in der VBA-Umgebung die Taste **[F2]**, um den Objekt-Katalog anzuzeigen. Geben Sie in das zweite Eingabefeld den Namen der Konstanten ein, beispielsweise chChartTypeBarClustered. Es werden nun sämtliche verfügbaren Konstanten für Diagrammtypen angezeigt.

Nachdem der Diagrammtyp festgelegt wurde, werden die Kategorienamen arrCategories übergeben. In der letzten Codezeile innerhalb der With-Anweisung wird die Datenreihe erzeugt (SeriesCollection(0)). Es werden die Daten aus dem Array strDataRow übergeben.

Hinweis

Weitere Informationen bzgl. Diagramme finden Sie in der Kategorie »Diagramme«. Beachten Sie jedoch, dass das Objektmodell ein anderes ist. Wenn Sie OWC auf Ihrer Festplatte installiert haben, dann finden Sie dort eine Hilfedatei. In Office 2000 trägt sie den Namen *MSOWCVBA.CHM*, in Office XP *OWCVBA10.CHM* und in Office 2003 *OWCVBA11.CHM*. Das Objektmodell ist dieser Hilfedatei zu entnehmen.

Unsere Beispiele hier beziehen sich auf die Version XP.

```
'=====
' Auf CD      Buchdaten\Beispiele\10_UserForm\
' Dateiname   10_04_UserForm_OWC_Chart
' UserForm    frm_01_ChartIntern
'=====
```

```
Private Sub UserForm_Initialize()
    Dim arrCategories(4)
    Dim arrDataRow(4)

    arrCategories(0) = "1. Quartal"
    arrCategories(1) = "2. Quartal"
    arrCategories(2) = "3. Quartal"
    arrCategories(3) = "4. Quartal"

    arrDataRow(0) = 300
    arrDataRow(1) = 400
    arrDataRow(2) = 500
    arrDataRow(3) = 200

    With ChartSpace1.Charts.Add
        .Type = chChartTypeBarClustered
        .SetData chDimCategories, chDataLiteral, arrCategories
        .SeriesCollection(0).SetData chDimValues, chDataLiteral,
            arrDataRow
    End With
End Sub
```


354 Diagramm in UserForm mit Daten aus Tabelle

Im vorangegangenen Rezept wurden die Daten für das Diagramm direkt im Code aufbereitet. Alternativ dazu können Sie die Daten auch aus einem Tabellenblatt beziehen. Erstellen Sie dazu wie gewohnt Ihre Tabelle, aus welcher ein Diagramm erzeugt werden soll.

Genau wie im vorangegangenen Rezept erstellen wir ein UserForm mit einer Web-Komponente in Form eines Diagramms.

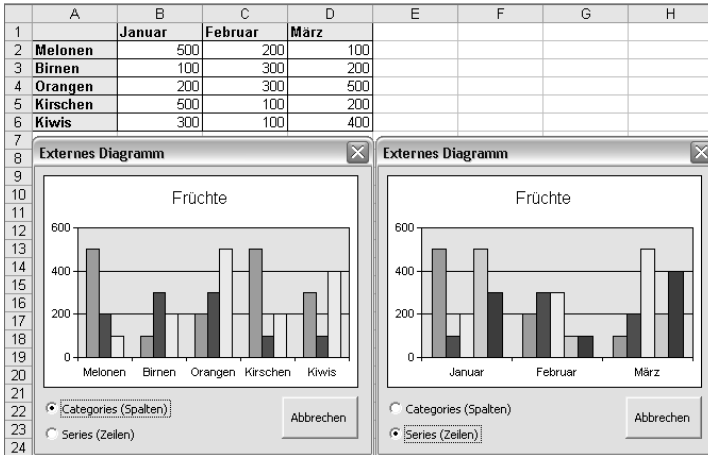


Abbildung 295: UserForm mit Datenquelle aus Tabelle

Im UserForm befinden sich zwei Optionsfelder. Wenn das Erste aktiviert ist, sollen die Datenreihen in Spalten angezeigt werden, wenn das Zweite aktiviert ist, sollen die Datenreihen in Zeilen ausgegeben werden (siehe Abbildung 295). Dazu verwenden wir zwei Ereignis-Prozeduren, die an je eines der Optionsfelder gebunden sind.

In der ersten Prozedur werden die ersten drei Zeilen aus dem Tabellenblatt ausgelesen. Aus jeder einzelnen Zeile soll eine Datenreihe entstehen. Die Daten für jede einzelne Reihe müssen an eine Variable vom Typ Array übergeben werden, denn Web-Komponenten erkennen Objekte wie z.B. `Range` oder `Cells` nicht.

Nach der Variablendeklaration wird ein allfällig vorhandenes Diagramm entfernt. Dies muss geschehen, damit nicht ein zweites Diagramm im UserForm erzeugt wird, wenn eines der Optionsfelder angeklickt wird.

In der ersten `For`-Schleife werden die Variablen für die Datenreihen sowie die Bezeichnungen für die Zeilen aufbereitet. In der zweiten `For`-Schleife werden die Spaltenüberschriften an eine Variable übergeben. Damit befinden sich alle benötigten Werte in den einzelnen Arrays.

In der ersten `With`-Anweisung wird ein Säulendiagramm erzeugt und ein Titel hinzugefügt. In der zweiten `With`-Anweisung werden die Daten an das zuvor erstellte Diagramm `Charts(0)` übergeben.

```

=====
' Auf CD      Buchdaten\Beispiele\10_UserForm\
' Dateiname  10_04_UserForm_OWC_Chart
' UserForm   frm_02_ChartExtern
=====

```

```

Private Sub opt_Categories_Click()
    Dim DataCol1(1 To 6)
    Dim DataCol2(1 To 6)
    Dim DataCol3(1 To 6)
    Dim arrCategories(1 To 6)
    Dim i As Integer

    ' Altes Diagramm entfernen
    ChartSpace1.Clear

    For i = 6 To 2 Step -1
        ' Daten aus Spalten an Array übergeben
        DataCol1(i) = Cells(i, 2)
        DataCol2(i) = Cells(i, 3)
        DataCol3(i) = Cells(i, 4)

        ' Bezeichnungen für Zeilen
        arrCategories(i) = Cells(i, 1)
    Next i

    ' Säulen-Diagramm erstellen
    With ChartSpace1.Charts.Add
        .HasTitle = True
        .Title.Caption = "Früchte"
        .Type = chChartTypeColumnClustered
    End With

    ' Datenreihen erzeugen
    With ChartSpace1.Charts(0)
        With .SeriesCollection
            .Add.SetData chDimValues, chDataLiteral, DataCol1
            .Add.SetData chDimValues, chDataLiteral, DataCol2
            .Add.SetData chDimValues, chDataLiteral, DataCol3
        End With
        .SetData chDimCategories, chDataLiteral, arrCategories
    End With
End Sub

```

In der zweiten Prozedur wird das Optionsfeld für Zeilen programmiert. Hier geschieht im Grunde genommen dasselbe wie in der ersten Prozedur. Der Unterschied besteht lediglich darin, dass diesmal nicht die Zeilen, sondern die Spalten aus dem Tabellenblatt ausgelesen werden.

```

' =====
' Auf CD      Buchdaten\Beispiele\10_UserForm\
' Dateiname   10_04_UserForm_OWC_Chart
' UserForm    frm_02_ChartExtern
' =====

Private Sub opt_Series_Click()
    Dim DataRow1(1 To 4)
    Dim DataRow2(1 To 4)

```

```
Dim DataRow3(1 To 4)
Dim DataRow4(1 To 4)
Dim DataRow5(1 To 4)
Dim arrSeries(1 To 4)
Dim i As Integer

' Altes Diagramm entfernen
ChartSpace1.Clear

For i = 4 To 2 Step -1
    ' Daten aus Zeilen an Array übergeben
    DataRow1(i) = Cells(2, i)
    DataRow2(i) = Cells(3, i)
    DataRow3(i) = Cells(4, i)
    DataRow4(i) = Cells(5, i)
    DataRow5(i) = Cells(6, i)

    ' Bezeichnungen für Spalten
    arrSeries(i) = Cells(1, i)
Next i

' Säulen-Diagramm erstellen
With ChartSpace1.Charts.Add
    .HasTitle = True
    .Title.Caption = "Früchte"
    .Type = chChartTypeColumnClustered
End With

' Datenreihen erzeugen
With ChartSpace1.Charts(0)
    With .SeriesCollection
        .Add.SetData chDimValues, chDataLiteral, DataRow1
        .Add.SetData chDimValues, chDataLiteral, DataRow2
        .Add.SetData chDimValues, chDataLiteral, DataRow3
        .Add.SetData chDimValues, chDataLiteral, DataRow4
        .Add.SetData chDimValues, chDataLiteral, DataRow5
    End With
    .SetData chDimCategories, chDataLiteral, arrSeries
End With
End Sub
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

355 Mappenbaum in UserForm (TreeView)

In einem UserForm können Sie einen »Mappenbaum« anzeigen lassen. Es werden dabei alle geöffneten Mappen und die darin enthaltenen Tabellenblätter dargestellt. Sie können auf diese Weise, ähnlich wie im Explorer, zwischen den einzelnen Mappen und Tabellenblättern wechseln.

Um das zu realisieren, müssen Sie Ihrer Werkzeugsammlung ein weiteres Element hinzufügen. Klicken Sie mit der rechten Maustaste auf die Werkzeugsammlung und wählen Sie aus dem Kontextmenü den Eintrag ZUSÄTZLICHE STEUERELEMENTE. Im Dialogfenster, das sich daraufhin öffnet, aktivieren Sie den Eintrag Microsoft TREEVIEW CONTROL, VERSION 6.0. Fügen Sie das Element in Ihr UserForm ein (siehe Abbildung 296).

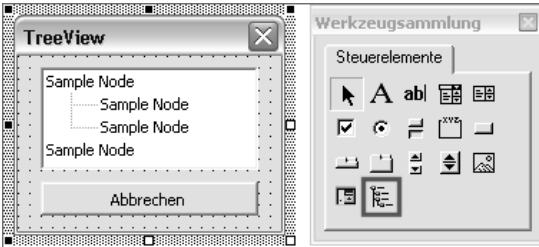


Abbildung 296: TreeView in UserForm

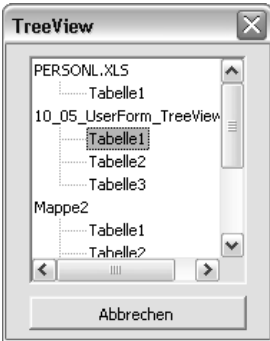


Abbildung 297: Mappenbaum

Wir benötigen zwei Ereignis-Prozeduren. Die eine dient dazu, den Mappenbaum beim Start des UserForm `UserForm_Initialize` anzuzeigen. Innerhalb der Ereignisprozedur wird mit zwei `For`-Schleifen gearbeitet. Die äußere `For`-Schleife erstellt die Mappeneinträge `Workbooks.Nodes.Add`. Die innere Schleife erstellt die Untereinträge, also die Einträge für die Tabellenblätter `Worksheet.Nodes.Add`.

Damit die Mappeneinträge sowie die Tabellenblätter geöffnet angezeigt werden, setzen wir die Eigenschaft `Expanded` auf `True`.

```
'=====
' Auf CD      Buchdaten\Beispiele\10_UserForm\
' Dateiname  10_05_UserForm_TreeView
' UserForm   frm_01_TreeView
'=====
```

```
Private Sub UserForm_Initialize()
    Dim wb As Workbook
    Dim ws As Worksheet
    Dim ndeMain As Node
    Dim ndeSub As Node

    With TreeView1
        For Each wb In Workbooks
            Set ndeMain = .Nodes.Add(Text:=wb.Name)
            For Each ws In wb.Worksheets
                Set ndeSub = .Nodes.Add _
```

```
                (Relative:=ndeMain, _  
                Relationship:=twvChild, _  
                Text:=ws.Name)  
        Next ws  
  
        ndeMain.Expanded = True  
    Next wb  
End With  
End Sub
```

Die zweite Ereignis-Prozedur wird dem Mappenbaum hinterlegt (TreeView1). Es wird beim Anklicken des TreeView geprüft, ob darin ein Haupt- oder ein Untereintrag selektiert wurde. Dies wird mittels einer If-Entscheidung geprüft. Es wird abgefragt, ob es sich bei dem selektierten Eintrag um ein Kindelement, also einen untergeordneten Eintrag handelt. Entsprechend wird darauf reagiert.

```
' =====  
' Auf CD      Buchdaten\Beispiele\10_UserForm\  
' Dateiname  10_05_UserForm_TreeView  
' UserForm   frm_01_TreeView  
' =====  
  
Private Sub TreeView1_Click()  
    With TreeView1.SelectedItem  
        If .Children = False Then  
            Workbooks(.Parent.Text).Worksheets(.Text).Activate  
        Else  
            Workbooks(TreeView1.SelectedItem.Text).Activate  
        End If  
    End With  
End Sub
```

356 Animierte Grafik in UserForm

Wenn Sie eine animierte Grafik z.B. vom Typ *.gif in ein UserForm einfügen, dann steht die Grafik still. Das heißt, die Animation wird nicht ausgeführt. Es wird von all den Bildfolgen, die eine Animation ausmachen, nur das erste Bild angezeigt. Wenn die Grafik animiert werden soll, können Sie ein Browser-Fenster in Ihr UserForm einbinden. Voraussetzung dafür ist, dass auf Ihrem Dateisystem der Internet Explorer von Microsoft installiert ist.

Um Ihrer Werkzeugsammlung das Element für den Webbrowser hinzuzufügen, klicken Sie mit der rechten Maustaste auf die Werkzeugsammlung. Wählen Sie aus dem Kontextmenü den Eintrag ZUSÄTZLICHE STEUERELEMENTE. In dem Dialogfenster, das sich nun öffnet, aktivieren Sie den Eintrag MICROSOFT WEBBROWSER. Als Symbol wird eine Weltkugel in der Werkzeugsammlung angezeigt. Fügen Sie ein Webbrowser-Element in Ihrem UserForm ein. Es wird dabei ein schwarzes Fenster mit dem Windows-Zeichen angezeigt (siehe Abbildung 298).

Als Ereignis-Prozedur verwenden wir UserForm_Initialize. Die Animation wird damit direkt bei Start des UserForm geladen und ausgeführt. Wenn sich die Datei auf Ihrer Festplatte befindet, können Sie an Stelle des URL auch einen Pfad angeben. Wenn Sie mehrere Webbrowser einfügen, werden diese jeweils über die aufsteigende Nummer angesprochen, WebBrowser1, WebBrowser2 usw.

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia



Abbildung 298: Einen Webbrowser einbinden

```
' =====
' Auf CD      Buchdaten\Beispiele\10_UserForm\
' Dateiname  10_06_UserForm_Animation
' UserForm   frm_01_Animation
' =====
```

```
Private Sub UserForm_Initialize()
    frm_01_Animation.WebBrowser1.Navigate _
        "http://www.jumper.ch/Logos/Moni_Compi.gif"
End Sub
```

Hinweis

In der Kategorie »Objekte« finden Sie ein Beispiel, das zeigt, wie ein Browser-Fenster direkt in ein Tabellenblatt integriert werden kann.

Wichtig: Eine Verbindung zum Internet muss bestehen, wenn Sie auf eine Datei aus dem Web zugreifen möchten.

357 Hyperlink in UserForm

Für gewöhnlich können in einem UserForm keine Hyperlinks angezeigt werden. Mit einer Trickkiste kann jedoch auch das realisiert werden. Erstellen Sie ein UserForm mit einem Bezeichnungsfeld. Beschriften Sie das Bezeichnungsfeld mit der Webadresse. Damit das Ganze wie ein Hyperlink aussieht, formatieren Sie den Text in blauer Farbe und unterstrichen.

Nun fehlt noch der Mauszeiger, denn dieser wird beim Überfahren eines Hyperlinks für gewöhnlich als Händchen dargestellt. Im Eigenschaften-Fenster zum Bezeichnungsfeld können Sie die gewünschten Einstellungen vornehmen. Dazu benötigen Sie allerdings ein entsprechendes Cursor-Symbol. Sie finden solche Symbole auf Ihrem Dateisystem. Um Ihnen die Suche danach zu ersparen, stellen wir Ihnen auf der CD die Datei *Hand.cur* zur Verfügung. Sie befindet sich im selben Verzeichnis wie die nachfolgende Beispieldatei. Klicken Sie im Eigenschaften-Fenster neben der Einstellung **MOUSEICON** auf die Schaltfläche mit den drei Punkten, um das Dialogfenster **BILD LADEN** zu öffnen. Wählen Sie die Datei *Hand.cur* aus. Bei der Eigenschaft **MOUSEPOINTER** wählen Sie den Eintrag **99 – FM_MOUSEPOINTER_CUSTOM**.



Abbildung 299: UserForm mit Hyperlink

Nachdem alle nötigen Formatierungen vorgenommen wurden, um einen Hyperlink zu simulieren, können Sie dem Bezeichnungsfeld den VBA-Code zuweisen. Der Text im Bezeichnungsfeld ist immer noch kein Hyperlink und wird auch nie einer sein, da UserForms dies nicht unterstützen. VBA bietet jedoch die Methode `FollowHyperlink` an. Diese Methode ist direkt der Mappe zugeordnet. Somit haben wir die Möglichkeit, als Adresse den Textinhalt des Bezeichnungsfeldes mitzuliefern. `NewWindow:=True` bedeutet, dass die Webseite in einem eigenen Fenster geöffnet wird.

Für den Fall, dass die URL nicht oder nicht mehr existieren sollte, verwenden wir eine Fehlerbehandlung. Damit verhindern wir das Starten des Debuggers, falls ein Fehler auftreten sollte. Eine entsprechende Meldung `MsgBox` wird angezeigt.

```
'=====
' Auf CD      Buchdaten\Beispiele\10_UserForm\
' Dateiname  10_07_UserForm_Hyperlink
' UserForm   frm_01_Hyperlink
'=====

Private Sub Label1_Click()
    On Error GoTo ErrorHandler
    ActiveWorkbook.FollowHyperlink _
        Address:=Label1.Caption, _
        NewWindow:=True
    Unload Me

Exit Sub
ErrorHandler: MsgBox "Ungültiger Hyperlink"
End Sub
```

Achtung

Stellen Sie vor dem Ausführen des Codes sicher, dass die Verbindung zum Internet besteht.

358 Eine eigene Werkzeugsammlung zusammenstellen

In diesem Rezept wird zwar kein VBA-Code verwendet, wir zeigen Ihnen jedoch einen nützlichen Trick, den Sie vielleicht noch nicht kennen. Sie können in der Werkzeugsammlung eine weitere Registerkarte hinzufügen und dort nach Belieben Ihre Steuerelemente zusammenstellen.

Um eine neue Registerkarte in der Werkzeugsammlung einzufügen, klicken Sie mit der rechten Maustaste auf die Beschriftung der bestehenden Registerkarte `STEUERELEMENTE`. Wählen Sie aus dem Kontextmenü den Eintrag `NEUE SEITE`. Eine zweite Registerkarte wird angezeigt. Wenn Sie den Namen des Registers ändern möchten, klicken Sie mit der rechten Maustaste auf die neue Registerbeschriftung und wählen Sie aus dem Kontextmenü den Eintrag `Umbenennen`. Geben Sie den gewünschten Namen ein.

Erstellen Sie ein UserForm und fügen Sie aus der Standard-Werkzeugsammlung eine Schaltfläche in Ihr UserForm ein. Vergeben Sie der Schaltfläche eine gewünschte Farbe. Sie können im Eigenschaften-Fenster auch noch andere Einstellungen zur Formatierung vornehmen, wie z.B. zur Schrift. Ziehen Sie nun die formatierte Schaltfläche auf Ihre selbst erstellte Registerkarte. Ein Sym-

Grundlagen
Allgemein
Datum und Zeit
Formeln
Diagramme
Objektmodelle
Diagramme
Ereignisse
UserForm
Web/Mail
Extern
Gemischtes
Spezial

bol wird hinzugefügt. Wenn Sie nun mit diesem Symbol weitere Schaltflächen in Ihr UserForm einfügen, dann werden diese genauso formatiert dargestellt.

Sie können noch einen Schritt weiter gehen. Fügen Sie in Ihrem UserForm drei Optionsfelder ein. Markieren Sie alle Optionsfelder und ziehen Sie die ganze Gruppe auf die neu erstellte Registerkarte. Die gesamte Gruppe lässt sich nun auf Ihren UserForms einfügen.

Die neuen Symbole lassen sich wieder entfernen, indem Sie sie mit der rechten Maustaste anklicken und aus dem Kontextmenü den Eintrag LÖSCHEN wählen.

In dieser Kategorie dreht sich alles rund um das Internet. Sie werden hier erfahren, wie Webabfragen erstellt und wie E-Mail verschiedener Art verwendet werden können. Sie lernen alles rund um Hyperlinks, werden in Smart Tags und Webabfragen eingeführt. XML ist ebenfalls ein Bestandteil dieser Kategorie. Die Themen Smart Tags und XML sind erst seit der Excel-Version 2002 (XP) verfügbar.

359 Eine einfache E-Mail versenden

Sie haben in Excel zwei Möglichkeiten, eine E-Mail manuell zu versenden. Entweder Sie binden die Mappe in Ihre E-Mail ein, oder Sie versenden sie als Anlage.

- Menü: DATEI | SENDEN AN | E-MAIL-EMPFÄNGER
- Menü: DATEI | SENDEN AN | E-MAIL-EMPFÄNGER (ALS ANLAGE)

Seit der Excel-Version 2002 (XP) haben Sie die Möglichkeit, einzelne Tabellenblätter oder nur einzelne Bereiche zu versenden. Wählen Sie dazu die erste der beiden oben erwähnten Möglichkeiten. Markieren Sie vor dem Senden den Zellbereich oder aktivieren Sie das Tabellenblatt, das versendet werden soll. Die Schaltfläche SENDEN zeigt je nach Auswahl einen entsprechenden Text an. Für einen markierten Zellbereich lautet der Text AKTUELLE AUSWAHL SENDEN, für ein Tabellenblatt DIESES BLATT SENDEN, für ein Diagramm AKTUELLES CHART SENDEN usw. Wenn Sie eine Mail auf diese Weise versenden, hat der Empfänger jedoch nicht die Möglichkeit, den Inhalt der Mail als Excel-Tabelle zu bearbeiten. Er muss den Inhalt zuerst in eine Mappe kopieren. Ein Chart wird dabei in eine Grafik umgewandelt.

Wenn Sie per VBA eine Mail aus Excel heraus versenden, greifen Sie in der Regel auf Outlook oder Outlook Express zu, es sei denn, Sie arbeiten mit einer anderen Software zum Versenden Ihrer Nachrichten. Das bedeutet, dass wir es hier, von Excel aus gesehen, mit einer externen Applikation zu tun haben (mehr zum Thema »Extern« erfahren Sie in der entsprechenden Kategorie). Die Applikation, auf welche Sie zugreifen wollen, muss entsprechend lauffähig installiert sein. Wenn Sie in Excel einen Programmcode schreiben, der auf Outlook zugreift, müssen Sie entweder den entsprechenden Verweis auf die Bibliothek von Outlook setzen (»Early Binding«), oder Sie erzeugen das Objekt während der Laufzeit (»Late Binding«).

Wenn Sie das Objekt während der Laufzeit erzeugen möchten, bedienen Sie sich einer entsprechenden Variablendeklaration und Referenzierung.

```
Dim objOutlook As Object
Dim objMail As Object
Set objOutlook = CreateObject("Outlook.Application")
Set objMail = objOutlook.CreateItem(0)
```

Wenn Sie die Bibliothek von Outlook gleich im Vornherein aktivieren möchten, setzen Sie im VBA-Editor den Verweis auf die Bibliothek MICROSOFT OUTLOOK 10.0 OBJECT LIBRARY. Sie finden die Bibliotheken im VBA-Editor unter dem Menüpunkt EXTRAS | VERWEISE.

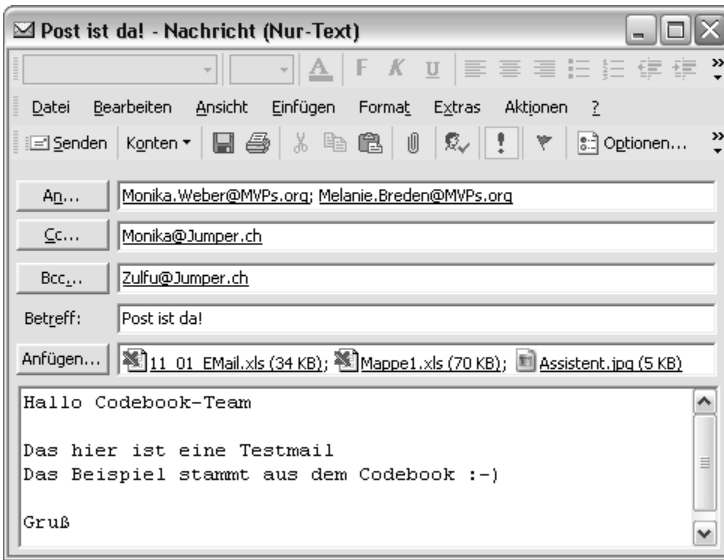


Abbildung 300: Eine E-Mail erzeugen

Wir wollen uns im ersten Rezept ansehen, wie eine Mail aus Excel heraus erzeugt werden kann. Die Mail soll an verschiedene Empfänger (To) verschickt werden. Die einzelnen E-Mail-Adressen werden dabei durch ein Semikolon (;) getrennt. Es soll eine Kopie an einen weiteren Empfänger verschickt werden (CC). CC ist die Abkürzung von »Carbon Copy«; dieser Begriff stammt aus alten Zeiten, wo noch Kohlepapier verwendet wurde, um Durchschläge, sprich Kopien, zu erstellen. Ein weiterer Empfänger soll eine Blindkopie erhalten (BCC (Blind Carbon Copy)). Eine Blindkopie zu verschicken bedeutet, dass die Empfänger, welche in To und CC angeschrieben wurden, nicht sehen können, dass eine Kopie an einen weiteren Empfänger verschickt wurde. Diese Einstellung ist mit Vorsicht zu genießen, denn beim Versand über das Internet kann es vorkommen, dass in BCC enthaltene Einträge zu CC konvertiert werden. Im Zweifelsfalle wird die Mail lieber ein zweites Mal separat an den Blindempfänger geschickt.

Die Mail soll einen Betreff Subject und einen Nachrichtentext Body haben. Drei Dateien werden an die Mail angehängt. Die aktive Mappe selbst (ThisWorkbook.FullName), eine weitere Excel-Mappe sowie eine Grafik. Die beiden zuletzt genannten Dateien (Mappe1.xls und Assistent.jpg) müssen sich im Laufwerk C:\ befinden. Beide Dateien finden Sie im selben Verzeichnis wie die Beispielmappe. Kopieren Sie diese ins Laufwerk C:\, bevor Sie den Code ausführen. Es wird eine Empfangsbestätigung angefordert (ReadReceiptRequested = True). Zu guter Letzt wird die Mail angezeigt (Display). Sie können die Mail alternativ auch direkt verschicken, ohne dass sie zuvor angezeigt wird. Ersetzen Sie dazu die Methode Display durch die Methode Send. Am Ende der Prozedur zeigen wir in der Statuszeile von Excel an, dass im Moment eine Mail versendet wird (StatusBar).

```
'=====
' Auf CD      Buchdaten\Beispiele\11_Web_Mail\
' Dateiname  11_01_EMail.xls
' Tabelle    11_WebMail_EMail
' Modul      mdl_01_SendSimpleMail
'=====
```

```
Sub SendSimpleMail()  
    Dim objOutlook As Object  
    Dim objMail As Object  
  
    Set objOutlook = CreateObject("Outlook.Application")  
    Set objMail = objOutlook.CreateItem(0)  
  
    With objMail  
        .To = "Monika.Weber@MVPs.org; Melanie.Breden@MVPs.org"  
        .CC = "Monika@Jumper.ch"  
        .BCC = "Zulfu@Jumper.ch"  
        .Subject = "Post ist da!"  
        .Body = "Hallo Codebook-Team" & vbCrLf & vbCrLf & _  
                "Das hier ist eine Testmail" & vbCrLf & _  
                "Das Beispiel stammt aus dem Codebook :-)" & _  
                vbCrLf & vbCrLf & "Gruß"  
        With .Attachments  
            .Add ThisWorkbook.FullName  
            .Add "C:\Mappel.xls"  
            .Add "C:\Assistent.jpg"  
        End With  
        .ReadReceiptRequested = True  
        .Display      ' oder direkt Send  
    End With  
  
    Application.StatusBar = "Eine Mail wird versendet!"  
  
    Set objOutlook = Nothing  
    Set objMail = Nothing  
End Sub
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Anweisung	Beschreibung
To	An
CC	Kopie
BCC	Blindkopie
Subject	Betreff
Body	Nachrichtentext im Format Text
HTMLBody	Nachrichtentext im Format HTML
Attachments	Anhang
ReadReceiptRequested = True	Empfangsbestätigung anfordern
Display	Mail vor dem Versenden anzeigen
Send	Mail direkt versenden

Extern

Ge-
misch
tes

Specia

Tabelle 82: Die wichtigsten E-Mail-Optionen

360 Eine E-Mail mit HTML-Formatierungen

Wenn Sie sich beim Versenden Ihrer E-Mail für das HTML-Format HTMLBody entscheiden, können Sie mit HTML-Tags arbeiten. Das bedeutet, dass Ihnen die gleichen Formatierungen zur Verfügung stehen, wie beim Erstellen von Webseiten. Da E-Mails im HTML-Format jedoch Viren gefährdet sind, raten wir Ihnen generell von der Benutzung ab.



Abbildung 301: E-Mail mit HTML-Formatierungen

Nichtsdestotrotz können Sie in Ihrem VBA-Code mit HTML-Tags arbeiten. Gewisse Grundkenntnisse mit dem Umgang der Seitenbeschreibungssprache HTML sind dabei erforderlich. Wichtig beim Benutzen von HTML-Tags in einem VBA-Code ist, dass diese in Anführungs- und Schlusszeichen (»«) gesetzt werden. Wenn innerhalb der Tags Anführungs- und Schlusszeichen erforderlich sind, müssen diese jeweils doppelt geschrieben werden.

Zur Formatierung unserer E-Mail wollen wir eine Überschrift der Ebene 2 <h2> erstellen. Der Text soll in roter Farbe geschrieben werden. Dazu verwenden wir das Tag , gefolgt vom Attribut color und der entsprechenden Farbcodierung. Für Rot lautet diese #FF0000. Wir fügen zudem eine Grafik in unsere E-Mail ein (). Die Quelle dieser Grafik liegt auf einer Webseite <http://www.jumper.ch/blume.gif>.

Hinweis

Um die Blume (siehe Abbildung 301) sehen zu können, müssen Sie mit dem Internet verbunden sein, da die Grafik auf einem Webserver im Internet liegt.

```
'=====
' Auf CD      Buchdaten\Beispiele\11_Web_Mail\
' Dateiname   11_01_EMail.xls
' Tabelle     11_WebMail_EMail
' Modul       mdl_02_SendHTMLMail
'=====
```

```
Sub SendHTMLMail()
    Dim objOutlook As Object
    Dim objMail As Object

    Set objOutlook = CreateObject("Outlook.Application")
```

```
Set objMail = objOutlook.CreateItem(0)

With objMail
    .To = "Monika.Weber@MVPs.org"
    .Subject = "Post ist da!"
    .HTMLBody = _
        "<html>" & _
        "<body>" & _
        "<h2>" & _
        "Hallo Codebook-Team" & _
        "</h2>" & _
        vbCrLf & _
        "<p>" & _
        "<font color=""#FF0000"">" & _
        "Das hier ist eine Testmail" & _
        "</font>" & _
        "</p>" & _
        vbCrLf & _
        "<p>" & _
        "<img border=""0"" & _
        "src=""http://www.jumper.ch/blume.gif"" & _
        "width=""31"" & _
        "height=""31"">" & _
        "</p>" & _
        "</body>" & _
        "</html>"
    .Display
End With

Set objOutlook = Nothing
Set objMail = Nothing
End Sub
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

361 Ein Tabellenblatt als E-Mail versenden

Wie bereits erwähnt, ist es seit der Version 2002 (XP) möglich, einzelne Tabellenblätter als E-Mail zu versenden. Der Nachteil dabei ist, dass der Empfänger nicht wirkliche Excel-Daten erhält, sondern die Daten werden in die E-Mail selbst eingebunden. Das bedeutet, dass der Empfänger diese unter Umständen nicht ohne größeren Aufwand weiter verarbeiten kann. Von daher kann es immer noch sehr nützlich sein, einen VBA-Code zu erstellen, mit dem man jederzeit das aktive Tabellenblatt als Anhang in einer Excel-Mappe verschicken kann. Vor der Version 2002 war dies ohnehin unerlässlich.

Manuell sieht der Vorgang wie folgt aus: Das aktive Tabellenblatt soll als Anhang versendet werden. Wir kopieren somit das aktive Tabellenblatt in eine neue Excel-Mappe und verschicken diese neu erstellte Mappe als Anlage. Nach dem Versenden kann diese Mappe wieder gelöscht werden. Genau diese Schritte werden wir in unserer VBA-Prozedur automatisieren.

Unser VBA-Code muss diesmal anders aufgebaut werden, als wir es bisher gewohnt sind. Der Grund liegt darin, dass mit der herkömmlichen Methode, also mit dem Zugriff auf Outlook, die aktive Mappe nicht übergeben werden kann, wenn diese nicht zuvor unter einem Namen abgespeichert worden ist. Da wir genau dies nicht tun möchten, werden wir mit Excel-eigenen Befehlen arbeiten. Zuerst wird das aktive Blatt kopiert (ActiveSheet.Copy). Damit wird auch gleich eine

neue Mappe erzeugt. Die neue Mappe enthält nur dieses eine Blatt, das wir kopiert haben. Um das E-Mail-Dialogfenster zu öffnen, lassen wir das entsprechende Dialogfenster anzeigen (Show). Der Nachteil an diesem Dialogfenster ist, dass lediglich drei Argumente übergeben werden können (recipients, subject, return_receipt). Auf einen Text im Nachrichtenfenster, der per VBA erzeugt wird, muss somit verzichtet werden. Am Ende der Prozedur wird die neu erstellte Mappe ohne diese zu speichern geschlossen (Close Savechanges:=False).

```
' =====
' Auf CD      Buchdaten\Beispiele\11_Web_Mail\
' Dateiname   11_01_EMail.xls
' Tabelle     11_WebMail_EMail
' Modul       mdl_03_SendActiveSheetV1
' =====

Sub SendActiveSheetV1()
    ActiveSheet.Copy

    With Application
        .Dialogs(xlDialogSendMail).Show _
            "Monika.Weber@MVPs.org", _
            "Post ist da!"
    End With

    ActiveWorkbook.Close Savechanges:=False
End Sub
```

Falls Sie wie gewohnt alle Optionen, wie CC, Body usw. verwenden möchten, ist es unerlässlich, auf Outlook zuzugreifen und die neu erstellte Mappe zuerst abzuspeichern, denn es muss beim Versenden ein Pfad und ein Dateiname FullName vorhanden sein. In diesem Beispiel speichern wir die neue Mappe im Laufwerk C:\ unter dem Namen *Test.xls* ab. Sollte bereits eine Datei mit diesem Namen in dem Verzeichnis vorhanden sein, wird ein entsprechendes Meldungsfenster angezeigt, das rückfragt, ob die Datei überschrieben werden soll. Am Ende der Prozedur wird die Mappe geschlossen (Close) und anschließend gelöscht (Kill).

```
' =====
' Auf CD      Buchdaten\Beispiele\11_Web_Mail\
' Dateiname   11_01_EMail.xls
' Tabelle     11_WebMail_EMail
' Modul       mdl_04_SendActiveSheetV2
' =====

Sub SendActiveSheetV2()
    Dim objOutlook As Object
    Dim objMail As Object

    Set objOutlook = CreateObject("Outlook.Application")
    Set objMail = objOutlook.CreateItem(0)

    ActiveSheet.Copy
    ActiveWorkbook.SaveAs "C:\Test.xls"
```

```

With objMail
    .To = "Monika.Weber@MVPs.org"
    .Subject = "Post ist da!"
    .Body = "Hallo Codebook-Team"
    .Attachments.Add ActiveWorkbook.FullName
    .Display
End With

ActiveWorkbook.Close
Kill ("C:\Test.xls")

Set objOutlook = Nothing
Set objMail = Nothing
End Sub

```

362 Zellbereich per E-Mail versenden

Im Rezept 361 haben wir ein einzelnes Tabellenblatt verschickt. Sie können auch nur einen bestimmten Zellbereich versenden. Das Verfahren dabei ist ähnlich. Da wir jedoch nicht das gesamte Blatt kopieren können, kopieren wir zuerst den Bereich Copy. Dann erzeugen wir eine neue Mappe Workbooks.Add. Der kopierte Bereich, der sich noch in der Zwischenablage befindet, wird in der neuen Mappe in das aktive Tabellenblatt eingefügt (Paste). Auch diesmal muss die Mappe gespeichert werden. Die Mappe kann danach versendet, geschlossen und gelöscht werden.

```

'=====
' Auf CD      Buchdaten\Beispiele\11_Web_Mail\
' Dateiname  11_01_EMail.xls
' Tabelle    11_WebMail_EMail
' Modul      mdl_05_SendRange
'=====

```

```

Sub SendRange()
    Dim objOutlook As Object
    Dim objMail As Object

    Set objOutlook = CreateObject("Outlook.Application")
    Set objMail = objOutlook.CreateItem(0)

    ActiveSheet.Range("F2:G4").Copy
    Workbooks.Add
    ActiveSheet.Paste
    ActiveWorkbook.SaveAs "C:\Test.xls"

    With objMail
        .To = "Monika.Weber@MVPs.org"
        .Subject = "Post ist da!"
        .Body = "Hallo Codebook-Team"
        .Attachments.Add ActiveWorkbook.FullName
        .Display
    End With

    ActiveWorkbook.Close

```

Grund-
lagenAllge-
meinDatu
Zeit

tungen

Steuer
elemenBefehl
leisten

Objekt

Dia-
grammEreig-
nisseUser-
FormWeb/
Mail

Extern

Ge-
misch
tes

Specia

```

Kill ("C:\Test.xls")

Set objOutlook = Nothing
Set objMail = Nothing
End Sub

```

363 Serien-Mail mit einheitlichem Inhalt

Ziel dieses Rezeptes ist es, eine einzige E-Mail mit mehreren Empfängern, jedoch gleichem Inhalt zu erzeugen. Der Inhalt Ihrer E-Mail wird diesmal aus Excel-Zellen bezogen. Das Problem in diesem Fall besteht darin, die E-Mail-Adressen zu ermitteln, denn die Anzahl der Adressen kann sich ändern und auch die Adressen selbst. Wir werden deshalb mit einem benannten Bereich arbeiten. Die Bereiche werden direkt im VBA-Code benannt.



Abbildung 302: Serien-Mail mit benanntem Bereich

Nach der Dimensionierung und Referenzierung wird als Erstes der Bereich, in dem sich die E-Mail-Adressen befinden, benannt (`Names.Add`). Dem `Add` folgt der Name, den der Bereich erhalten soll: `Name:=Empfänger`. Danach wird der Bereich festgelegt: `RefersTo:=`. Es muss der Name des Tabellenblattes angegeben werden (`11_WebMail_MultiMailV1`) sowie der Bereich selbst (`$A$7:$A$8`). Um den Bereich dynamisch zu halten, werden wir die letzte benutzte Zeile mit der Anweisung `Cells(Rows.Count, 1).End(xlUp).Row` ermitteln. Der benannte Bereich ist nun festgelegt.

Jetzt müssen die im benannten Bereich `Range("Empfänger")` enthaltenen E-Mail-Adressen an eine Variable `strRecipients` übergeben werden. Dazu verwenden wir eine `For`-Schleife, die die Anzahl Zeilen `Rows.Count` im benannten Bereich durchläuft.

Nach dem Ermitteln der E-Mail-Adressen kann die E-Mail selbst aufbereitet werden. Als Empfänger `To` verwenden wir die zuvor gefüllte Variable `strRecipients`. Die restlichen Informationen für den Betreff, Text und Anhang beziehen wir aus den entsprechenden Zellen.

```

' =====
' Auf CD   Buchdaten\Beispiele\11_Web_Mail\
' Dateiname 11_01_EMail.xls
' Tabelle   11_WebMail_MultiMailV1
' Modul     mdl_06_SendMultiMailV1
' =====

```

```

Sub SendMultiMailV1()
    Dim objOutlook As Object
    Dim objMail As Object

```



```
Dim i As Integer
Dim strRecipients As String

Set objOutlook = CreateObject("Outlook.Application")
Set objMail = objOutlook.CreateItem(0)

' Bereich der E-Mail-Adressen benennen
ActiveWorkbook.Names.Add Name:="Empfänger", _
    RefersTo:="=11_WebMail_MultiMailV1!$A$7:$A$" & _
    Cells(Rows.Count, 1).End(xlUp).Row

' Empfänger-Adressen ermitteln und an Variable übergeben
For i = 1 To Range("Empfänger").Rows.Count
    strRecipients = strRecipients & _
        Range("Empfänger").Rows(i).Value & ";"
Next i

' Die E-Mails werden erzeugt
With objMail
    .To = strRecipients
    .Subject = Range("B1").Value
    .Body = Range("B2").Value
    .Attachments.Add Range("B3").Value
    .Attachments.Add Range("B4").Value
    .Display
End With

Set objOutlook = Nothing
Set objMail = Nothing
End Sub
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

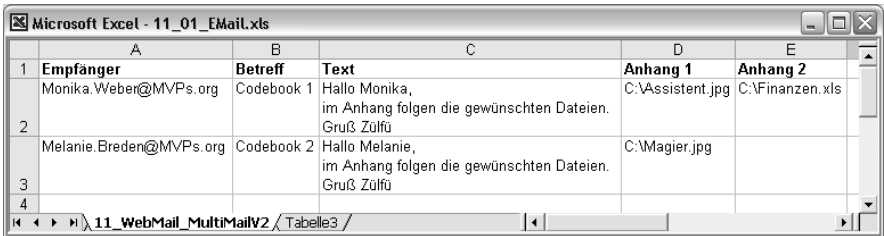
Ge-
misch
tes

Specia

364 Serien-Mail mit unterschiedlichem Inhalt

Im Rezept 363 wurde eine Serien-Mail erstellt, die immer denselben Inhalt hat. Sie können jedoch auch eine Serien-Mail erzeugen, die für jeden Empfänger einen individuellen Inhalt bereitstellt, wie z.B. eine persönliche Anrede und verschiedene Dateianhänge. Dabei werden mehrere E-Mails erzeugt, nämlich pro Empfänger eine.

Die Excel-Daten müssen entsprechend anders aufgebaut werden. Der Abbildung 303 können Sie entnehmen, dass jede Zeile (außer der Titelzeile) für eine eigene E-Mail steht. In diesem Beispiel sollen also zwei E-Mails erzeugt werden.



	A	B	C	D	E
1	Empfänger	Betreff	Text	Anhang 1	Anhang 2
	Monika.Weber@MVPs.org	Codebook 1	Hallo Monika, im Anhang folgen die gewünschten Dateien. Gruß Zülfü	C:\Assistent.jpg	C:\Finanzen.xls
2	Melanie.Breden@MVPs.org	Codebook 2	Hallo Melanie, im Anhang folgen die gewünschten Dateien. Gruß Zülfü	C:\Magier.jpg	
3					
4					

Abbildung 303: Individuelle Serien-Mail

Um mehrere E-Mails zu erzeugen, müssen wir mit einer Schleife arbeiten. Wir verwenden hier eine For-Schleife. In dieser Schleife wird ab der zweiten Zeile ermittelt, wie viele Zeilen in dem Tabellenblatt vorhanden sind. Eine entsprechende Anzahl an E-Mails wird erzeugt. Der Unterschied zu den vorangegangenen Rezepten besteht unter anderem darin, dass die Mails erst in der Schleife erzeugt werden. Eine Referenzzeile fällt somit weg. Wenn wir diese Referenzzeile auch hier verwenden würden, würde bereits vor der Schleife eine E-Mail erzeugt. Genau das wollen wir ja nicht.

Der restliche Aufbau der E-Mail ist ähnlich dem vorangegangenen Rezept. Der Inhalt der Zellen wird an die Outlook-Felder übergeben. Beim Anhang prüfen wird diesmal zusätzlich auf mögliche Fehler. Der Debugger könnte sich melden, wenn eine Anhang-Zelle leer wäre oder eine falsche Pfad- oder Dateiangabe enthält. Vor dem Einfügen der Dateianhänge `Attachments.Add` werden wir deshalb entsprechende Prüfungen vornehmen. Um das Ganze sauber umzusetzen, werden die Zellinhalte bzw. Pfadangaben an eine Variable `strPath` übergeben. Die If-Entscheidung nimmt die Prüfung vor. Wenn die Angaben in korrekter Form vorliegen, werden die Dateien angehängt.

```
'=====
' Auf CD      Buchdaten\Beispiele\11_Web_Mail\
' Dateiname   11_01_EMail.xls
' Tabelle     11_WebMail_MultiMailV2
' Modul       mdl_07_SendMultiMailV2
'=====
```

```
Sub SendMultiMailV2()
    Dim objOutlook As Object
    Dim objMail As Object
    Dim strPath As String
    Dim i As Integer

    Set objOutlook = CreateObject("Outlook.Application")

    For i = 2 To ActiveSheet.Cells(Rows.Count, 1).End(xlUp).Row

        ' Die E-Mails werden erzeugt
        With objOutlook.CreateItem(0)
            .To = Cells(i, 1)
            .Subject = Cells(i, 2)
            .Body = Cells(i, 3)

            ' Anhang: Prüfen, ob ein Eintrag vorhanden ist
            ' und ob die Pfadangabe korrekt ist
            strPath = Cells(i, 4)
            If strPath <> "" And Dir(strPath) <> "" Then
                .Attachments.Add strPath
            End If

            strPath = Cells(i, 5)
            If strPath <> "" And Dir(strPath) <> "" Then
                .Attachments.Add strPath
            End If

            .Display
        End With
    End For
End Sub
```

```
Next i

Set objOutlook = Nothing
End Sub
```

365 Interne Hyperlinks

Unter internen Hyperlinks verstehen sich Hyperlinks, die innerhalb der Mappe gesetzt werden. Sie können beispielsweise Hyperlinks erstellen, die zu anderen Zellen, Tabellenblättern oder benannten Bereichen führen.

Manuell erzeugen Sie einen Hyperlink, indem Sie den Menüpunkte EINFÜGEN | HYPERLINK wählen. Alternativ können Sie auch die Tastenkombination `[Strg] + [k]` drücken. In dem Dialogfenster, das sich nun öffnet, finden Sie linkerhand eine Leiste. Dort können Sie festlegen, wohin der Hyperlink führen soll. Wenn der Hyperlink innerhalb des Dokumentes gesetzt werden soll, klicken Sie auf AKTUELLES DOKUMENT (siehe Abbildung 304). Im ersten Feld TEXT ANZEIGEN ALS können Sie den Text eingeben, den die Zelle anzeigen soll. Im zweiten Feld geben Sie die Zielzelle an, auf welche der Hyperlink zeigen soll. Im dritten Feld können Sie die Zieltabelle oder einen benannten Bereich auswählen, zu dem der Hyperlink führen soll. Benannte Bereiche werden natürlich nur angezeigt, wenn in Ihrem Tabellenblatt welche enthalten sind. Sobald Sie Ihre Auswahl in dem Dialogfenster getroffen haben, bestätigen Sie dies, indem Sie die Schaltfläche OK anklicken. In der aktiven Zelle können Sie nun den Hyperlink sehen. Hyperlinks sind an der blauen Textfarbe und an der Unterstreichung zu erkennen, sofern diese Grundeinstellungen nicht verändert wurden. Wenn Sie mit dem Mauszeiger über den Hyperlink fahren, verwandelt sich dieser in ein Händchen. Ein einfacher Klick auf den Hyperlink führt Sie zu dem hinterlegten Ziel.

Wenn Sie den Hyperlink verändern möchten, aktivieren Sie die Zelle, die den Hyperlink enthält, und betätigen wiederum die Tastenkombination `[Strg] + [k]`, um das Dialogfenster wieder zu öffnen. Sie können dann nach Belieben Änderungen vornehmen.

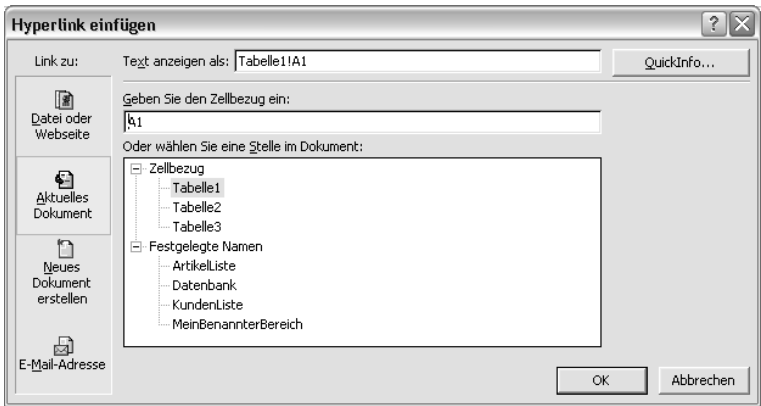


Abbildung 304: Dialogfenster für Hyperlinks

Um die VBA-Befehle für Hyperlinks zu erfahren, verwenden Sie am besten den Makro-Rekorder und zeichnen die gewünschten Schritte auf. Etwas übersichtlich geordnet könnte der Code wie folgt aussehen:

```

' =====
' Auf CD      Buchdaten\Beispiele\11_Web_Mail\
' Dateiname   11_02_Hyperlinks.xls
' Modul       mdl_01_AddHyperlink
' =====

Sub AddHyperlink()
    ActiveSheet.Hyperlinks.Add _
        Anchor:=Selection, _
        Address:="", _
        SubAddress:="Tabelle2!B2", _
        TextToDisplay:="Zum Tabellenblatt 2"
End Sub

```

Mittels `Hyperlinks.Add` wird ein Hyperlink erstellt. In unserem Beispiel werden vier Argumente übergeben. `Anchor` bedeutet übersetzt »Anker«. Hinter `Anchor` wird die Zelle eingegeben, welche den Hyperlink erhalten soll. In unserem Beispiel `Selection`, also die selektierte Zelle. Sie können alternativ auch einen Zellbezug angeben wie z.B. `Range("A1")`. `Address` ist in diesem Beispiel leer, wir werden später bei den externen Hyperlinks wieder darauf zurück kommen. Hinter `SubAddress` folgt die Zieltabelle und Zielzelle. Das letzte Argument nennt sich `TextToDisplay`. Hier steht der Text, der in der Zelle angezeigt wird.

366 Hyperlink mit Quickinfo

Eine Quickinfo ist das hellgelbe Fenster, das sich öffnet, wenn Sie mit der Maustaste auf den Hyperlink zeigen, ohne diesen anzuklicken. Sie können den darin enthaltenen Text auch selbst bestimmen. Die zu verwendende Eigenschaft nennt sich `ScreenTip`.

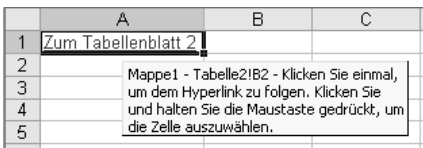


Abbildung 305: Quickinfo

Mit einer `For`-Schleife können Sie allen im Tabellenblatt enthaltenen Hyperlinks automatisch eine neue Quickinfo zuweisen. In unserem Beispiel wird allen Hyperlinks ein Text sowie der eigene Index zugewiesen. Das bedeutet, wenn Sie nach dem Ausführen des Codes über einen der Hyperlinks fahren, zeigt Ihnen die Quickinfo den selbst definierten Text sowie den Index des Hyperlinks an.

```

' =====
' Auf CD      Buchdaten\Beispiele\11_Web_Mail\
' Dateiname   11_02_Hyperlinks.xls
' Modul       mdl_02_Quickinfo
' =====

Sub QuickInfo()
    Dim i As Integer

    For i = 1 To ActiveSheet.Hyperlinks.Count

```

```
ActiveSheet.Hyperlinks(i).ScreenTip = _
    "Dies ist der Hyperlink mit dem Index " & i
Next i
End Sub
```

367 Mittels Hyperlinks ein Inhaltsverzeichnis erstellen

Mittels VBA haben Sie die Möglichkeit, auf sehr schnelle Weise ein Inhaltsverzeichnis in Ihrer Mappe zu erstellen. Das Inhaltsverzeichnis wird in unserem Beispiel mit Hyperlinks versehen. Jeder Hyperlink führt zu dem gewünschten Tabellenblatt. Jedes der Zieltabellenblätter wird zudem mit einem Hyperlink zurück zum Inhaltsverzeichnis versehen. Sie haben so die Möglichkeit, jederzeit per Klick wieder zurück zum Inhaltsverzeichnis zu navigieren. Lohnenswert ist dies vor allem bei Mappen, die viele Tabellenblätter enthalten.



Abbildung 306: Tabellenblätter mit Inhaltsverzeichnis und Navigation

Der Code ist so aufgebaut, dass keine bestehenden Daten überschrieben werden. Für das Inhaltsverzeichnis wird ein eigenes Tabellenblatt mit dem Namen INHALTSVERZEICHNIS erstellt. Zu Beginn der Prozedur wird geprüft, ob bereits ein Tabellenblatt mit dem Namen INHALTSVERZEICHNIS besteht. Falls dies der Fall ist, wird eine Dialogbox MsgBox angezeigt. Die Dialogbox enthält die Schaltflächen JA und NEIN. Wenn die Schaltfläche JA geklickt wird, wird das bestehende Tabellenblatt mit dem Namen INHALTSVERZEICHNIS gelöscht und die Schleife wird verlassen. Wenn auf NEIN geklickt wird, wird die Prozedur verlassen und eine entsprechende Meldung ausgegeben.

Im zweiten Teil der Prozedur wird das Tabellenblatt mit dem Namen INHALTSVERZEICHNIS erzeugt (Worksheets.Add). Das neue Tabellenblatt soll das erste in der Mappe sein. Wir ergänzen daher das Argument: Before:=Worksheets(1).

Nachdem das Tabellenblatt für das Inhaltsverzeichnis erstellt wurde, können wir damit beginnen, die Hyperlinks zu erstellen. Wir verwenden dazu eine For-Schleife. Im ersten Block wird das Inhaltsverzeichnis auf dem ersten Tabellenblatt erstellt.

Im zweiten Codeblock, innerhalb der For-Schleife, werden die Hyperlinks auf den weiteren Tabellenblättern erzeugt. Sie dienen dazu, zum Inhaltsverzeichnis zurückzunavigieren. Die Hyperlinks werden jeweils in die Zelle A1 geschrieben. Damit auch hier keine Daten überschrieben werden, wird jeweils eine Zeile vor die erste bestehende Zeile eingefügt Worksheets(i).Rows(1).Insert. Noch vor dieser Codezeile wird geprüft, ob bereits eine alte Navigationszeile vorhanden ist. Wenn dies der Fall sein sollte, wird sie gelöscht. Wenn wir dies nicht tun würden, würde bei jedem Ausführen des Codes eine weitere Navigationszeile hinzugefügt und es wären pro Blatt mehrere enthalten. Das wäre natürlich kein schöner Effekt.

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

```

' =====
' Auf CD      Buchdaten\Beispiele\11_Web_Mail\
' Dateiname  11_03_TableOfContents.xls
' Modul      mdl_01_TableOfContents
' =====

Sub TableOfContents()
    Dim i As Integer
    Dim strTOC As String
    Dim strNavigation As String
    Dim strQuestion As String

    strTOC = "Inhaltsverzeichnis"
    strNavigation = "Zurück zum Inhaltsverzeichnis"

    ' Prüfen, ob bereits ein Tabellenblatt mit dem Namen
    ' "Inhaltsverzeichnis" existiert
    For i = 1 To Worksheets.Count
        If Worksheets(i).Name = strTOC Then
            strQuestion = MsgBox _
                ("Ein Tabellenblatt mit den Namen: " & _
                & strTOC & _
                " existiert bereits. " & _
                "Wollen Sie es löschen?", _
                vbCritical + vbYesNo)

            If strQuestion = vbYes Then
                Application.DisplayAlerts = False
                Worksheets("Inhaltsverzeichnis").Delete
                Application.DisplayAlerts = True
                Exit For
            Else
                MsgBox "Die Prozedur wurde abgebrochen"
                Exit Sub
            End If
        End If
    Next i

    ' Ein neues Tabellenblatt für das Inhaltsverzeichnis
    ' erzeugen, damit keine Daten überschrieben werden
    Worksheets.Add Before:=Worksheets(1)
    With Worksheets(1)
        .Name = strTOC
        .Range("A1").Value = strTOC
    End With

    ' Hyperlinks erzeugen
    For i = Worksheets.Count To 2 Step -1

        ' Inhaltsverzeichnis erstellen
        Worksheets(1).Hyperlinks.Add _
            Anchor:=Worksheets(1).Cells(i, 1), _
            Address:="", _
            SubAddress:="" & Worksheets(i).Name & "!A1", _

```

```
TextToDisplay:=Worksheets(i).Name

' Navigation zurück zum Inhaltsverzeichnis
If i <> 1 Then
    With Worksheets(i)

        ' Alte Navigationszeile löschen, falls vorhanden
        If .Range("A1").Value = strNavigation Then
            .Rows(1).Delete
        End If

        ' Neue Navigationszeile einfügen
        .Rows(1).Insert

        .Hyperlinks.Add _
            Anchor:=Worksheets(i).Range("A1"), _
            Address:="", _
            SubAddress:="" & Worksheets(1).Name & "!A1", _
            TextToDisplay:=strNavigation
    End With
End If

Next i
End Sub
```

368 Hyperlinks und Zelladressen (Suchen/Weitersuchen)

Dieses Rezept stellt eine Suchmaschine dar. In ein Eingabefeld `InputBox` kann ein Suchbegriff eingegeben werden. Die Suchmaschine sucht in dem aktiven Tabellenblatt nach übereinstimmenden Zeichenketten. Auf einem separaten Tabellenblatt, das während der Laufzeit der Prozedur erstellt wird, werden die gefundenen Zelladressen ausgegeben und mit Hyperlinks versehen. Die Hyperlinks führen zurück zu den Zellen, die die Zeichenketten enthalten.



Abbildung 307: Suchmaschine mit Zell- und Hyperlinkausgabe

Nach der Variablendeklaration deaktivieren wir die Bildschirmaktualisierung, um die Geschwindigkeit der Prozedur zu erhöhen und das Bildflackern zu unterdrücken. Am Ende der Prozedur wird die Einstellung wieder aktiviert.

Danach übergeben wir der Variablen `strActSh` den Namen des aktiven Blattes. Wir merken uns diesen Namen, um später das richtige Blatt wieder aktivieren zu können.

Grundlagen

Allgemein

Datum
Zeit

Formeln

Steuerelemente

Befehle
leisten

Objekte

Diagramme

Ereignisse

User-Formen

Web/
Mail

Extern

Gemischtes

Spezial

Es wird dann geprüft, ob aus einer vorherigen Suche bereits ein Tabellenblatt mit dem Namen `AUSGABE` in der Mappe enthalten ist. Wenn dies der Fall ist, wird das Blatt gelöscht. Danach wird ein Blatt mit dem Namen `AUSGABE` neu erstellt. Um keine Meldung zu erhalten, ob das Blatt wirklich gelöscht werden soll, deaktivieren wir die Systemmeldungen `DisplayAlerts`. Nach dem Löschen wird sie wieder aktiviert.

Da nun das *neu* erstellte Blatt aktiv ist, werden wir das Blatt, in welchem die Suche stattfinden soll, wieder aktivieren. Dazu verwenden wir die zuvor erstellte Variable `strActSh` (`Worksheets(strActSh).Activate`).

Mittels eines Eingabefensters `InputBox` fragen wir nach dem Suchbegriff. Der gewünschte Begriff kann in dieses Eingabefenster eingetippt werden. Als Standardwert `Default` haben wir den Begriff »Codebook« hinterlegt. Damit muss der Begriff nicht jedes Mal eingegeben werden.

Vor der Sprungmarke übergeben wir unserem Zähler `i` den Wert 1. Der Zähler wird später verwendet, um die Zelladressen untereinander ins neue Tabellenblatt zu schreiben.

Die Sprungmarke benötigen wir, damit die Suche so lange fortgeführt werden kann, bis wir am Ende der Suche angelangt sind. Nach der Sprungmarke wird die Suchmaschine gestartet. Die Suche wird referenziert, damit wir die Möglichkeit haben, zu prüfen, ob der Suchbegriff im Tabellenblatt überhaupt vorhanden ist. Ein Errorhandling kann damit umgangen werden. Dies geschieht in der `If`-Entscheidung, welche dem Suchblock folgt. Wenn der Suchbegriff nicht gefunden wird, wird die Prozedur verlassen.

Wenn der Suchbegriff gefunden wird, wird die Zelladresse an die Variable `strAdr` übergeben. Dies dient zur besseren Übersicht für spätere Verwendungen.

In einem nächsten Schritt kann der Hyperlink an das Tabellenblatt `AUSGABE` übergeben werden. Wie der Hyperlink angezeigt wird, können Sie der Abbildung 307 entnehmen.

Damit die Prozedur nicht endlos ausgeführt wird, müssen wir das Ende der Suche abfragen. Dies geschieht in der `With`-Anweisung. Die letzte Zelle darf nicht `A1` sein, sonst würde die Prozedur schon nach dem ersten Durchlauf abgebrochen. Eine entsprechende Prüfung wird in der ersten `If`-Entscheidung vorgenommen. In der zweiten `If`-Entscheidung wird zudem geprüft, ob der letzte Eintrag mit dem Inhalt der Zelle `A1` übereinstimmt. Bei Zutreffen bedeutet das, dass die Suche wieder von vorne begonnen hat. An dieser Stelle wird die Prozedur verlassen. Zuvor wird jedoch der letzte Eintrag gelöscht, denn dieser ist ja mit dem ersten identisch. Die Spalte `A` wird zudem automatisch in der Breite angepasst. Dies ist nicht unbedingt erforderlich, sondern dient lediglich der Kosmetik.

Der Zähler `i` wird nun um den Wert 1 erhöht. Wenn die Suche noch nicht am Ende angelangt ist, werden mittels der Sprungmarke bestimmte Befehle der Prozedur wiederholt.

```
'=====
' Auf CD      Buchdaten\Beispiele\11_Web_Mail\
' Dateiname  11_04_Search.xls
' Modul      mdl_01_SearchFurther
'=====
```

```
Sub SearchFurther()
    Dim strActSh As String
    Dim ws As Worksheet
    Dim wsOutput As Worksheet
    Dim strAdr As String
    Dim i As Byte
    Dim strInput As String
```

```

Dim objFind As Object

Application.ScreenUpdating = False

' Ausgangsblatt merken
strActSh = ActiveSheet.Name

' Prüfen, ob ein Tabellenblatt mit dem Namen
' "Ausgabe" vorhanden ist, wenn ja, dann löschen
For Each ws In Worksheets
    If ws.Name = "Ausgabe" Then
        Application.DisplayAlerts = False
        ws.Delete
        Application.DisplayAlerts = True
    End If
Next ws

' Tabellenblatt "Ausgabe" erstellen
Worksheets.Add Before:=Worksheets(1)
Set wsOutput = Worksheets(1)
wsOutput.Name = "Ausgabe"

' Ausgangsblatt wieder aktivieren
Worksheets(strActSh).Activate

' Suchbegriff abfragen
strInput = Application.InputBox _
    ("Bitte den gewünschten Suchbegriff eingeben", _
    Default:="Codebook")

i = 1

' Suche starten
Sprungmarke:
Set objFind = ActiveSheet.Cells.Find(What:=strInput, _
    After:=ActiveCell, _
    LookIn:=xlFormulas, _
    LookAt:=xlPart, _
    SearchOrder:=xlByRows, _
    SearchDirection:=xlNext, _
    MatchCase:=False)

If objFind Is Nothing Then
    MsgBox "Der Suchbegriff wurde nicht gefunden"
    Exit Sub
Else: objFind.Activate
End If

' Zelladresse an Variable übergeben
strAdr = ActiveCell.Address(False, False)

' Hyperlink hinzufügen
ActiveSheet.Hyperlinks.Add _
    Anchor:=wsOutput.Cells(i, 1), _

```

Grund-
lagenAllge-
meinDatum
Zeit

tungen

Steuer
elemenBefehl
leisten

Objekt

Dia-
grammEreig-
nisseUser-
FormWeb/
Mail

Extern

Ge-
misch
tes

Specia

```

    Address:="", _
    SubAddress:="" & Worksheets(strActSh).Name & "!" & strAdr, _
    TextToDisplay:=strAdr & " ... " & strActSh

    ' Wenn wieder die erste Zelle der Suche erreicht wird,
    ' dann die Prozedur verlassen. Die Suche ist damit beendet.
With wsOutput
    If .Cells(i, 1).Address <> "$A$1" Then
        If .Range("A1") = .Cells(i, 1) Then
            .Cells(i, 1).Clear
            .Columns(1).EntireColumn.AutoFit
            Exit Sub
        End If
    End If
End With

    ' Den Zähler erhöhen
    i = i + 1

    ' Suche wiederholen
    GoTo Sprungmarke

Application.ScreenUpdating = True

Set wsOutput = Nothing
Set objFind = Nothing
End Sub

```

369 Ein neues Dokument erstellen

Die Methode `CreateNewDocument` kann dazu verwendet werden, ein neues Dokument zu erstellen, das mit dem angegebenen Hyperlink verknüpft wird.

Die Prozedur erstellt zuerst einen Hyperlink in der Zelle A1. Beim Erstellen referenzieren wir direkt auf die Variable `objHLink`, welche im zweiten Codeblock wieder verwendet wird. Im zweiten Teil der Prozedur wird im Verzeichnis C:/ eine Mappe mit dem Namen `Mappe1.xls` erzeugt. Das Argument `EditNow` steht für »jetzt bearbeiten«. Setzen Sie den Wert auf `True`, wenn Sie die Datei direkt nach dem Erstellen bearbeiten möchten. Das Argument `Overwrite` steht für »überschreiben«. Setzen Sie den Wert auf `False`, wenn Sie eine Systemmeldung erhalten möchten, sofern schon eine Datei mit diesem Namen in dem angegebenen Laufwerk existiert.

```

' =====
' Auf CD      Buchdaten\Beispiele\11_Web_Mail\
' Dateiname   11_05_HyperlinksExtern.xls
' Modul       mdl_01_CreateNewFile
' =====

Sub CreateNewFile()
    Dim objHLink As Object

    With Worksheets(1)
        Set objHLink = .Hyperlinks.Add _
            (Anchor:=Range("A1"), _

```

```
Address:="C:\Mappe1.xls", _  
ScreenTip:="Mein neues Dokument", _  
TextToDisplay:="Mein neues Dokument")  
  
objHLink.CreateNewDocument _  
    Filename:="C:\Mappe1.xls", _  
    EditNow:=False, _  
    Overwrite:=True  
End With  
End Sub
```

370 Externe Hyperlinks

Externe Hyperlinks zeigen auf Dokumente außerhalb der aktiven Mappe. Zum Beispiel auf Webseiten oder auf Dokumente, die sich auf der Festplatte befinden.

Wenn Sie in eine Zelle einen *URL* (Unified Resource Locator) zu einer Webseite eingeben, wie z. B. *http://www.jumper.ch*, dann wird dieser durch Excel automatisch erkannt. Excel stellt den Hyperlink automatisch blau und unterstrichen dar.

Hinweis

Ab der Version 2002 (XP) kann das automatische Erkennen von externen Hyperlinks deaktiviert werden, indem im Menü EXTRAS | AUTOKORREKTUR-OPTION auf der Registerkarte AUTOFORMAT WÄHREND DER EINGABE das entsprechende Kontrollkästchen deaktiviert wird. Bei deaktiviertem Zustand werden Hyperlinks nicht mehr automatisch erzeugt.

Um manuell einen Hyperlink zu erstellen, wählen Sie den Menüpunkt EINFÜGEN | HYPERLINK oder betätigen die Tastenkombination `[Strg] + [k]`. Sie arbeiten also mit demselben Fenster, wie bei den internen Hyperlinks. Diesmal wählen Sie linkerhand jedoch nicht AKTUELLES DOKUMENT, sondern DATEI ODER WEBSEITE. Das rechte Fenster zeigt nun eine andere Auswahl an:

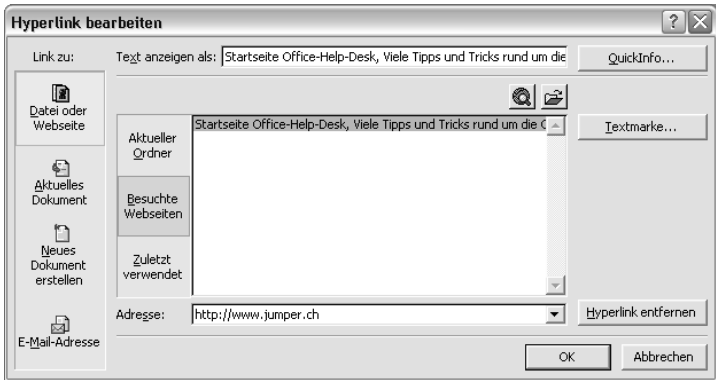


Abbildung 308: Hyperlink zu Datei oder Webseite

Die Syntax ist beinahe dieselbe wie die von internen Hyperlinks. Die *SubAddress*, welche nur für interne Zwecke verwendet wird, fällt weg. Dafür verwenden wir erstmals *Address*. Hier wird die Webadresse (URL) eingetragen.

Grundlagen

Allgemein

Datum
Zeit

tungen

Steuer
elemente

Befehl
leisten

Objekt

Diagramm

Ereignisse

User-
Form

Web/
Mail

Extern

Gemischtes

Spezial

```
'=====
' Auf CD      Buchdaten\Beispiele\11_Web_Mail\
' Dateiname   11_05_HyperlinksExtern.xls
' Modul       mdl_02_Website
'=====

Sub HyperlinkWebsite()
    ActiveSheet.Hyperlinks.Add _
        Anchor:=Range("A2"), _
        Address:="http://www.jumper.ch", _
        TextToDisplay:="Office-Help-Desk"
End Sub
```

Beim Anklicken des Hyperlinks wird der installierte Webbrowser gestartet und dem Hyperlink gefolgt. Voraussetzung dafür ist, dass ein Browser installiert ist und Sie mit dem Internet verbunden sind.

371 Hyperlink auf lokale Datei

Die Prozedur, um einen Hyperlink auf eine lokale Datei zu setzen, unterscheidet sich nur darin, dass keine Webadresse, sondern Datei- und Pfadangabe hinter *Address* eingegeben werden. Um zu verhindern, dass der Debugger gestartet wird, wenn die Datei- oder Pfadangaben nicht stimmen, verwenden wir in diesem Beispiel eine Fehlerprüfung. Den Pfad übergeben wir an die Variable *strFile*. In einer If-Entscheidung prüfen wir, ob der Pfad leer (" ") ist. Wenn dies zutrifft, wird eine Meldung *MsgBox* auf dem Bildschirm angezeigt. Die Prozedur wird verlassen (*Exit Sub*).

Wenn der Pfad korrekt ist, wird der Hyperlink in die Zelle A3 eingefügt.

```
'=====
' Auf CD      Buchdaten\Beispiele\11_Web_Mail\
' Dateiname   11_05_HyperlinksExtern.xls
' Modul       mdl_03_Path
'=====

Sub HyperlinkPath()
    Dim strFile

    strFile = "C:\Mappel.xls"

    If Dir(strFile) = "" Then
        MsgBox "Die Datei: " & strFile & " existiert nicht."
        Exit Sub
    End If

    ActiveSheet.Hyperlinks.Add _
        Anchor:=Range("A3"), _
        Address:=strFile, _
        TextToDisplay:="Übersetzungsliste DE/EN"
End Sub
```

372 Hyperlinks zu einer Textmarke

Wie Sie per Hyperlink auf andere Dateien verweisen, haben Sie im Rezept 371 erfahren. Sie haben zudem die Möglichkeit, einen Link zu erstellen, der zu einem bestimmten Tabellenblatt bzw. Zelle in

der anderen Mappe zeigt. Am einfachsten ist es, wenn Sie dort die Zielzelle benennen (EINFÜGEN | NAMEN | DEFINIEREN). Sie können den Hyperlink so direkt auf diese Textmarke zeigen lassen.

In unserem Beispiel haben wir eine Mappe mit dem Namen *Mappe1.xls* im Laufwerk C:\ erstellt. In dieser Mappe haben wir auf dem Tabellenblatt TABELLE2 der Zelle C7 den Namen *MeinBereich* zugewiesen.

Um auf diese Textmarke zu verweisen, verwenden wir das Argument *SubAddress*, gefolgt vom Namen des Bereiches *MeinBereich*.

Die Zielmappe wird beim Anklicken des Hyperlinks geöffnet und es wird direkt zu der benannten Zelle gesprungen.

```
'=====
' Auf CD      Buchdaten\Beispiele\11_Web_Mail\
' Dateiname  11_05_HyperlinksExtern.xls
' Modul      mdl_04_Bookmark
'=====

Sub HyperlinkBookmark()
    Dim strFile

    strFile = "C:\Mappel.xls"

    If Dir(strFile) = "" Then
        MsgBox "Die Datei: " & strFile & " existiert nicht."
        Exit Sub
    End If

    ActiveSheet.Hyperlinks.Add _
        Anchor:=Range("A4"), _
        Address:=strFile, _
        SubAddress:="MeinBereich", _
        TextToDisplay:="Mappel: MeinBereich"
End Sub
```

373 Hyperlinks zu Zellen/Tabellen einer anderen Mappe

Im vorangegangenen Rezept haben wir in der Zielmappe zuerst den Bereich benannt, der angesprungen werden soll. Dies ist nicht unbedingt erforderlich. Sie können auch den Namen des Tabellenblattes und der Zelle angeben. Im nachfolgenden Code verweisen wir auf die *Mappe1.xls* im Laufwerk C:\, welche wir zuvor angelegt haben. Innerhalb dieser Mappe soll direkt auf die Zelle C7 im Tabellenblatt TABELLE2 gesprungen werden. Wir übergeben dazu dem Argument *SubAddress* den Namen des Tabellenblattes sowie die gewünschte Zelle. Beachten Sie dabei, dass die beiden Einträge durch ein Ausrufezeichen (!) getrennt werden müssen: *SubAddress:="Tabelle2!C7"*.

```
'=====
' Auf CD      Buchdaten\Beispiele\11_Web_Mail\
' Dateiname  11_05_HyperlinksExtern.xls
' Modul      mdl_05_SheetCell
'=====

Sub HyperlinkSheetCell()
    Dim strFile
```

Grundlagen

Allgemein

Datum
Zeit

tungen

Steuer
elemente

Befehl
leisten

Objekt

Diagramm

Ereignisse

User-
Form

Web/
Mail

Extern

Gemischtes

Spezial

```

strFile = "C:\Mappel.xls"

If Dir(strFile) = "" Then
    MsgBox "Die Datei: " & strFile & " existiert nicht."
    Exit Sub
End If

ActiveSheet.Hyperlinks.Add _
    Anchor:=Range("A5"), _
    Address:=strFile, _
    SubAddress:="Tabelle2!C7", _
    TextToDisplay:=strFile & ", " & "Tabelle2, Zelle C7"

End Sub

```

374 E-Mail-Adressen

Excel erkennt E-Mail-Adressen automatisch und wandelt sie in Hyperlinks um. Beim Anklicken des Hyperlinks wird die Anwendung Outlook geöffnet. Voraussetzung dazu ist einerseits, dass unter dem Menüpunkt EXTRAS | AUTOKORREKTUR, Registerkarte AUTOFORMAT WÄHREND DER EINGABE das entsprechende Kontrollkästchen aktiviert ist, und andererseits, dass Outlook installiert ist.

Warum erkennt Excel, dass es sich dabei um eine E-Mail-Adresse und nicht um einen gewöhnlichen Hyperlink zu einer Webseite oder Datei handelt? Excel erkennt dies am Zeichen @ (das Zeichen wird als »at« ausgesprochen). Der Ausdruck »at« stammt aus dem Englischen. Das Wörterbuch verrät uns, dass die Bedeutung zu Deutsch »in«, »an«, »bei« oder »auf« ist. Ein @ wird durch die Autokorrektur als E-Mail-Adresse gewertet, sobald ein Zeichen vorausgeht. Wenn nur ein @ in die Zelle eingetippt wird, wird dieses nicht in einen Hyperlink umgewandelt. Wenn dem @ kein Zeichen vorangeht, jedoch ein Zeichen folgt, wird dies durch die Autokorrektur als Fehler gedeutet. Excel gibt eine entsprechende Meldung aus. Excel kann nicht erkennen, ob es sich um eine gültige und existierende E-Mail-Adresse handelt. Wenn Sie z.B. a@a in eine Zelle eingeben, wird diesem Eintrag ein E-Mail-Hyperlink zugeordnet, auch wenn eine solche E-Mail-Adresse gar nicht existieren kann.

Wie bereits erwähnt erkennt Excel am @, dass es sich um eine E-Mail-Adresse handelt. Deutlich wird dies, wenn Sie das Dialogfenster HYPERLINK BEARBEITEN anzeigen lassen ([Strg]+[k]). Vor der E-Mail-Adresse steht `mailto:`.

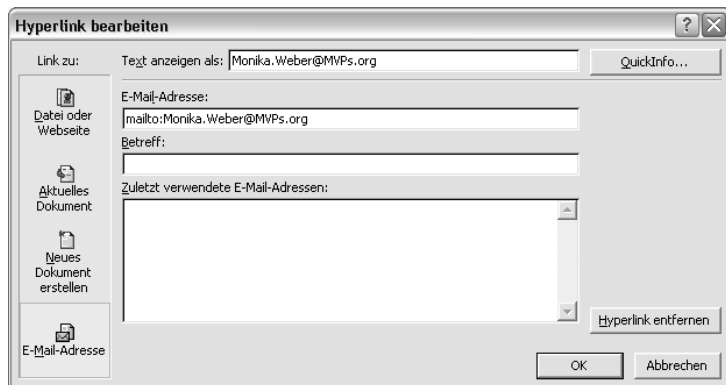


Abbildung 309: E-Mail-Hyperlink

Der VBA-Code ist genauso aufgebaut, wie gewöhnliche Hyperlinks. Bei dem Argument Address muss jedoch das mailto: als String mit übergeben werden. Wenn Sie zusätzlich noch einen Betreff mitliefern möchten, ergänzen Sie den String ?subject, gefolgt vom gewünschten Text.

```
' =====
' Auf CD      Buchdaten\Beispiele\11_Web_Mail\
' Dateiname  11_05_HyperlinksExtern.xls
' Modul      mdl_06_EMail
' =====

Sub EMail()
    ActiveSheet.Hyperlinks.Add _
        Anchor:=Range("A6"), _
        Address:="mailto:Monika.Weber@MVPs.org" & _
            "?subject=Eine Mail aus dem Codebook!", _
        ScreenTip:="E-Mail-Adresse", _
        TextToDisplay:="Monika Weber"
End Sub
```

375 Eine Schaltfläche mit Hyperlink

Sie können nicht nur Zellen, sondern auch andere Elemente, wie z.B. eine Schaltfläche aus der Symbolleiste STEUERELEMENT-TOOLBOX, mit einem Hyperlink versehen.

Da die Schaltfläche an das Tabellenblatt gebunden ist und die Prozedur auf Anklicken der Schaltfläche reagieren soll, verwenden wir eine Ereignisprozedur. Die Ereignisprozedur befindet sich hinter dem Tabellenblatt, in dem die Schaltfläche eingefügt wurde. Mehr zum Thema Ereignisprozeduren erfahren Sie in der Kategorie »Ereignisse«.

In unserer Prozedur übergeben wir die Webadresse an die Variable strHLink. Danach formatieren wir die Schaltfläche CommandButton1, damit der Text wie ein Hyperlink aussieht, also unterstrichen und mit blauer Textfarbe.

Der letzte Teil der Prozedur stellt den Hyperlink dar. Es wird die Methode FollowHyperlink verwendet.

```
' =====
' Auf CD      Buchdaten\Beispiele\11_Web_Mail\
' Dateiname  11_05_HyperlinksExtern.xls
' Ereignis    Tabelle1(11_05_HyperlinksExtern.xls)
' =====

Private Sub CommandButton1_Click()
    Dim strHLink As String

    strHLink = "http://www.jumper.ch"

    ' Formatierung der Schaltfläche (Blau/Unterstrichen)
    With CommandButton1
        .Caption = strHLink
        .ForeColor = RGB(0, 0, 256)
        .Font.Underline = True
    End With
```

Grundlagen

Allgemein

Datum
Zeit

Funktionen

Steuerelemente

Befehle
leisten

Objekte

Diagramme

Ereignisse

User-Formen

Web/Mail

Extern

Gemischtes

Spezial

```

' Hyperlink für die Schaltfläche
ActiveWorkbook.FollowHyperlink _
    Address:=strHLink, _
    NewWindow:=True
Exit Sub
End Sub

```

376 URL in Nebenzelle ausgeben

Die Hyperlinks, die in der Excel-Tabelle zu sehen sind, zeigen nicht immer die zu Grunde liegende Webadresse an. Oftmals wurde für die Anzeige im Tabellenblatt ein anderer Text gewählt, da URLs sehr lang sein können.

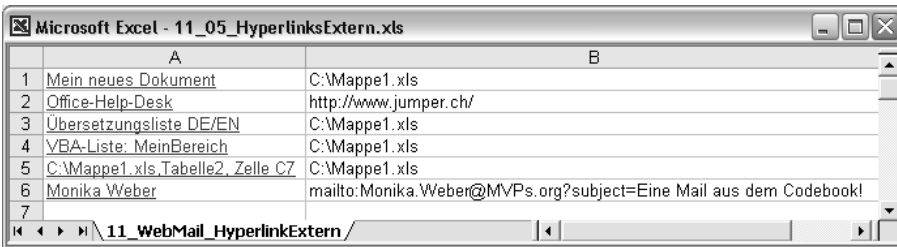


Abbildung 310: URL in rechter Nebenzelle ausgeben

Beim Ausführen der folgenden Prozedur wird die Webadresse `Hyperlinks(1).Address` jeweils in die rechte Nebenzelle des Hyperlinks geschrieben.

```

' =====
' Auf CD      Buchdaten\Beispiele\11_Web_Mail\
' Dateiname   11_05_HyperlinksExtern.xls
' Modul       mdl_07_LocateHypAddress
' =====

Sub LocateHypAddress()
    Dim rngCell As Range

    On Error Resume Next
    For Each rngCell In ActiveSheet.UsedRange
        rngCell.Offset(0, 1).Value = _
            rngCell.Hyperlinks(1).Address
    Next rngCell
End Sub

```

377 Hyperlinks löschen

Um manuell einen Hyperlink zu löschen, klicken Sie mit der rechten Maustaste auf den Hyperlink und wählen aus dem Kontextmenü den Eintrag **HYPERLINK ENTFERNEN**.

Um einen einzelnen Hyperlink per VBA zu löschen, verwenden Sie die Methode `Delete`. Innerhalb der `If`-Entscheidung wird geprüft, ob sich in dem Tabellenblatt ein Hyperlink befindet.

Wenn dies zutrifft, wird der erste Hyperlink gelöscht. Falls kein Hyperlink vorhanden sein sollte, wird eine Meldung auf dem Bildschirm angezeigt.

```
'=====
' Auf CD      Buchdaten\Beispiele\11_Web_Mail\
' Dateiname  11_05_HyperlinksExtern.xls
' Modul      mdl_08_DeleteOneHypLink
'=====

Sub DeleteOneHypLink()
  With ActiveSheet
    If .Hyperlinks.Count > 0 Then
      .Hyperlinks(1).Delete
    Else
      MsgBox "Es wurde kein Hyperlink gefunden."
    End If
  End With
End Sub
```

Wenn Sie manuell sämtliche Hyperlinks eines Tabellenblattes oder einer Mappe löschen möchten, erweist sich das als ziemlich aufwendig, denn Sie müssten jede Zelle, die einen Hyperlink enthält, separat bearbeiten. Das heißt, Sie müssten jede Zelle rechtsklicken und aus dem Kontextmenü den Eintrag **HYPERLINK ENTFERNEN** wählen.

Per VBA können Sie sehr einfach alle Hyperlinks aus einem Tabellenblatt entfernen, indem Sie mit einer **For...Each-Schleife** arbeiten.

```
'=====
' Auf CD      Buchdaten\Beispiele\11_Web_Mail\
' Dateiname  11_05_HyperlinksExtern.xls
' Modul      mdl_09_DeleteHypLinkSheet
'=====

Sub DeleteHypLinkSheet()
  Dim hyp As Hyperlink

  For Each hyp In ActiveSheet.Hyperlinks
    hyp.Delete
  Next hyp
End Sub
```

Um sämtliche Hyperlinks in der gesamten *Mappe* zu entfernen, verwenden Sie zwei verschachtelte **For...Each-Schleifen**:

```
'=====
' Auf CD      Buchdaten\Beispiele\11_Web_Mail\
' Dateiname  11_05_HyperlinksExtern.xls
' Modul      mdl_10_DeleteHypLinkWorkbook
'=====

Sub DeleteHypLinkWorkbook()
```

Grundlagen
Allgemein
Datum und Zeit
Formeln
Diagramme
Objekte
Diagramm
Ereignisse
User-Form
Web/Mail
Extern
Gemischtes
Spezial

```

Dim ws As Worksheet
Dim hyp As Hyperlink

For Each ws In Worksheets
    For Each hyp In ws.Hyperlinks
        hyp.Delete
    Next hyp
Next ws
End Sub

```

378 Nur E-Mail-Links löschen

Wie Sie ja nun wissen, beginnen E-Mail-Links in der Adresse jeweils mit einem `mailto:`. Sie haben somit per VBA die Möglichkeit, nur E-Mail-Links zu löschen, indem Sie das `mailto:` abfragen.

In einer `For...Each`-Schleife werden sämtliche Hyperlinks des aktiven Tabellenblattes angesprochen. Innerhalb der Schleife wird mit einer `If`-Entscheidung geprüft, ob die linken sieben Zeichen dem `mailto:` entsprechen. Wenn dies zutrifft, dann werden die Hyperlinks entfernt.

```

'=====
' Auf CD      Buchdaten\Beispiele\11_Web_Mail\
' Dateiname   11_05_HyperlinksExtern.xls
' Modul       mdl_11_DeleteMailLinks
'=====

Sub DeleteMailLinks()
    Dim hyp As Hyperlink

    For Each hyp In ActiveSheet.Hyperlinks
        If Left(hyp.Address, 7) = "mailto:" Then
            hyp.Delete
        End If
    Next hyp
End Sub

```

379 E-Mail-Links nicht löschen

Im Rezept 378 wurden nur die E-Mail-Links gelöscht. Natürlich können somit auch alle Links *außer* E-Mail-Links gelöscht werden, dazu vertauschen wir lediglich den Operator `=` (Gleich) durch den Operator `<>` (Ungleich).

```

'=====
' Auf CD      Buchdaten\Beispiele\11_Web_Mail\
' Dateiname   11_05_HyperlinksExtern.xls
' Modul       mdl_12_DeleteWebLinks
'=====

Sub DeleteWebLinks()
    Dim hyp As Hyperlink

    For Each hyp In ActiveSheet.Hyperlinks

```

```
If Left(hyp.Address, 7) <> "mailto:" Then
    hyp.Delete
End If
Next hyp
End Sub
```

380 Den HTML-Code einer Excel-Datei ansehen

Wenn Sie Ihre Excel-Dateien auf dem Inter- oder Intranet zur Verfügung stellen möchten, können Sie entweder direkt die *.xls-Datei veröffentlichen oder die Excel-Datei unter dem Format *.htm abspeichern. Um die Datei als *.htm abzuspeichern, wählen Sie den Menüpunkt DATEI | ALS WEBSEITE SPEICHERN. Sie haben hier die Möglichkeit, verschiedene Einstellungen vorzunehmen. Sie können wählen, ob die GESAMTE ARBEITSMAPPE oder nur eine AUSWAHL davon veröffentlicht werden soll.

Das Kontrollkästchen INTERAKTIVITÄT steht zur Verfügung. Wenn Sie dieses aktivieren, haben die Besucher der Webseite die Möglichkeit, eine eingeschränkte Auswahl an Excel-Funktionen interaktiv zu benutzen. Wenn Sie beispielsweise der Besitzer eines Hotels sind und auf Ihrer Webseite die Zimmerpreise veröffentlichen, dann könnten Sie den Besuchern ermöglichen, die Anzahl Übernachtungen selbst einzugeben, so dass der gesamte Aufenthalt der Reservation online und interaktiv berechnet wird.

Über die Schaltfläche TITEL ÄNDERN können Sie der Webseite einen Titel hinzufügen.

Des Weiteren steht eine Schaltfläche VERÖFFENTLICHEN zur Verfügung. Wenn Sie diese anklicken, öffnet sich ein weiteres Dialogfenster. Hier können noch mehr Einstellungen vorgenommen werden. Unter anderem der genaue Bereich, der veröffentlicht werden soll. Bei Interaktivität können Sie zudem wählen, ob die ARBEITSMAPPENFUNKTIONALITÄT oder PIVOTTABLE-FUNKTIONALITÄT zur Verfügung stehen soll. Die zweite Auswahl macht nur Sinn, wenn in Ihrem Tabellenblatt eine oder mehrere Pivot-Tabellen zu Verfügung stehen. Diese können dann im Browser-Fenster interaktiv umgestellt werden. TITEL und DATEINAME können verändert werden.

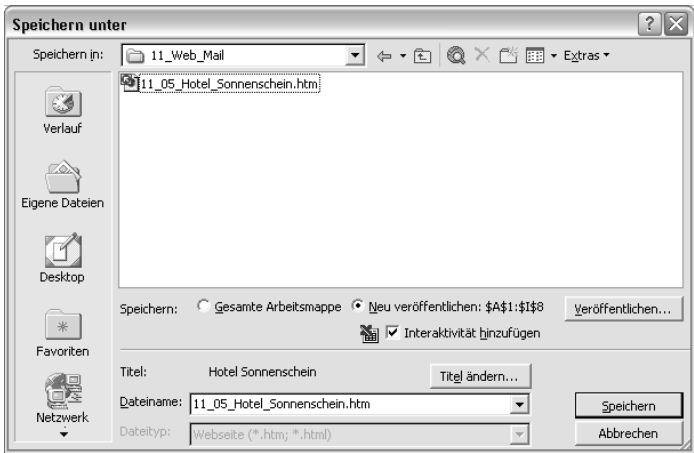


Abbildung 311: Excel-Datei als Webseite (*.htm) speichern

Bei aktiviertem Kontrollkästchen VERÖFFENTLICHTE WEBSEITE IM BROWSER ÖFFNEN können Sie die Schaltfläche VERÖFFENTLICHEN anklicken, um sich das Ergebnis Ihrer Auswahl im Browser-Fenster anzusehen. Voraussetzung dazu ist, dass ein Webbrowser auf Ihrem System installiert ist.

Grundlagen

Allgemein

Datum
Zeit

Formeln

Steuerelemente

Befehle
leisten

Objekte

Diagramme

Ereignisse

User-Form

Web/Mail

Extern

Gemischtes

Spezial

Achtung

Nur der Internet Explorer von Microsoft unterstützt alle diese Einstellungen. Andere Webbrowser bieten teilweise die Möglichkeit an, entsprechende Webkomponenten zu installieren, damit das Dokument betrachtet werden kann, was jedoch leider nicht immer funktioniert. Das bedeutet, dass die Webseite nicht durch alle Besucher genutzt werden kann. Überlegen Sie sich von daher gut, ob Sie dieses Feature wirklich einsetzen möchten.

Der Besucher hat im Internet Explorer die Möglichkeit, die Webdatei nach Excel zu exportieren, indem er das Excel-Symbol anklickt (siehe Abbildung 312).

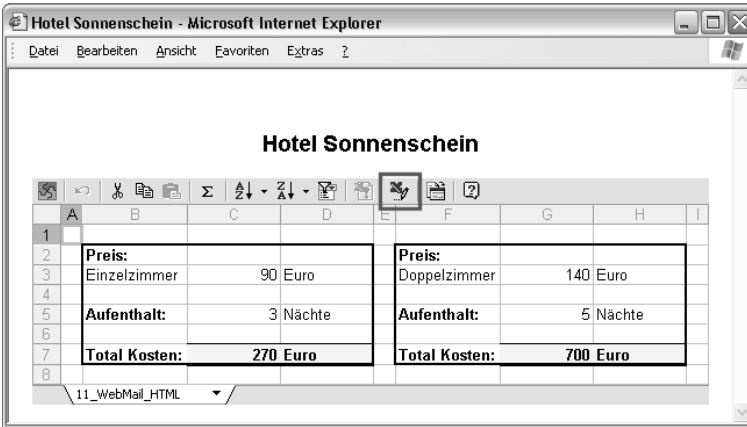


Abbildung 312: Ansicht im Browser-Fenster

Achtung

Wenn die Datei im Format *.htm veröffentlicht wird, sind keine VBA-Prozeduren verfügbar. Die Tabellenfunktionen sind nur eingeschränkt verwendbar.

Egal unter welchem Format die Datei abgespeichert wurde, Sie können per VBA jederzeit den HTML-Code (HyperText Markup Language) ansehen, der hinter der Datei mitgeschrieben wird. Excel hat die Angewohnheit, dabei sehr viel Code zu erzeugen. Wenn eine neue leere Excel-Mappe erzeugt wird, die noch keinen einzigen Eintrag enthält, könnte man meinen, sie ist leer. Wenn Sie jedoch den folgenden Code ausführen, wird deutlich, dass dies keineswegs der Fall ist.

Sie können den Script-Editor auch manuell starten, sofern ein Browser auf Ihrem System installiert ist. Wählen Sie dazu aus Ihrer Excel-Mappe heraus den Menüpunkt DATEI | WEBSEITENVORSCHAU. Ihr Standard-Browser wird gestartet. Wählen Sie im Browser-Fenster den Menüpunkt ANSICHT | QUELLTEXT. Der Texteditor zeigt nun den Code an.

```
'=====
' Auf CD      Buchdaten\Beispiele\11_Web_Mail\
' Dateiname  11_06_HTML.xls
' Modul      mdl_01_LookAtHTMLCode
'=====
```

```
Sub LookAtHTMLCode()
```

```
ActiveWorkbook.HTMLProject.Open
End Sub
```

Eine HTML-Seite kann aus mehreren Projekten, wie z. B. HTML-Code und CSS (Cascading Style Sheets), bestehen. CSS-Dateien sind Format-Vorlagen für Ihre Webseiten. Wenn Sie sich ansehen möchten, welche Projekte in Ihrem HTML-Dokument erzeugt wurden, führen Sie den nachfolgenden Code aus. In einer For-Schleife werden die Projekte abgefragt. In einem Meldungsfenster MsgBox werden die einzelnen Projekte angezeigt.

```
'=====
' Auf CD      Buchdaten\Beispiele\11_Web_Mail\
' Dateiname  11_06_HTML.xls
' Modul      mdl_02_LookAtHTMLProjects
'=====

Sub LookAtHTMLProjects()
    Dim i As Integer

    For i = 1 To ActiveWorkbook.HTMLProject.HTMLProjectItems.Count
        MsgBox ActiveWorkbook.HTMLProject.HTMLProjectItems(i).Name
    Next i
End Sub
```

Tipp

Es würde zu weit führen, hier näher auf HTML einzugehen. Falls Sie sich dafür interessieren, empfehlen wir Ihnen, entsprechende Literatur zu erwerben, wie z.B. »HTML & CSS lernen«, Verlag: Addison-Wesley, ISBN-Nr. 3-8273-2067-4.

381 Eine Webabfrage erstellen

Wenn Sie beispielsweise Börsenkurse abfragen möchten und Ihnen ein Hyperlink zu der entsprechenden Webseite nicht ausreicht, da Sie die Daten direkt in Excel weiterbearbeiten möchten, können Sie in Excel eine Webabfrage erstellen.

Klicken Sie auf den Menüpunkt: DATEN | EXTERNE DATEN IMPORTIEREN | NEUE WEBABFRAGE ERSTELLEN. Geben Sie in dem Fenster, das sich nun öffnet den gewünschten URL ein und klicken Sie auf die Schaltfläche START. Die Webdaten werden aufbereitet und einzelne Zeilen mit Pfeilen gekennzeichnet. Wenn Sie nur einige dieser Zeilen importieren möchten (siehe Abbildung 303), klicken Sie auf den gewünschten Pfeil. Der Pfeil verwandelt sich damit in ein Häkchen. Klicken Sie anschließend auf die Schaltfläche IMPORTIEREN. Ein neues Dialogfenster wird geöffnet. Geben Sie hier den Zielbereich, also die Zelle ein, ab welcher die Daten importiert werden sollen. Excel wird je nach Datenmenge eine Weile arbeiten und dann die gewünschten Informationen im Tabellenblatt ausgeben. Die Symbolleiste EXTERNE DATEN wird eingeblendet. Eine Schaltfläche in der Symbolleiste stellt ein rotes Ausrufezeichen dar. Die Schaltfläche ist dazu da, die Webabfrage zu aktualisieren.

Wenn Sie eine solche Webabfrage erstellen und dabei den Makro-Rekorder die Schritte aufzeichnen lassen, präsentiert sich nach der Aufzeichnung folgender Code im entsprechenden VBA-Modul:

Grundlagen

Allgemein

Datum und Zeit

Funktionen

Steuerelemente

Befehle

Objekte

Diagramme

Ereignisse

User-Formen

Web/Mail

Extern

Gemischtes

Spezial

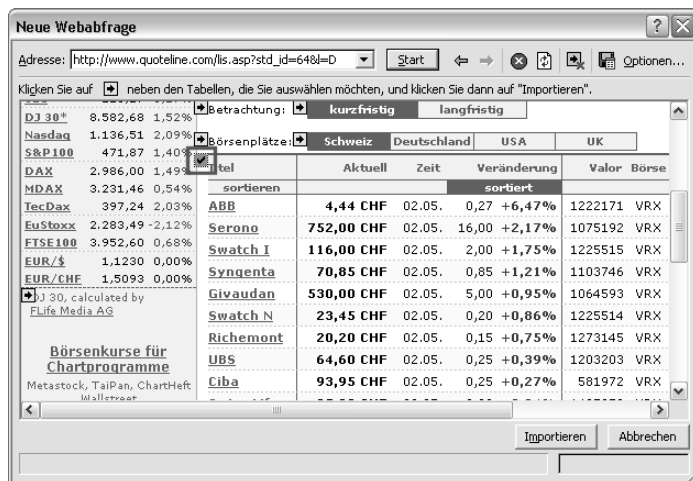


Abbildung 313: Webabfrage auf Schweizer Börsenkurse

```

' =====
' Auf CD      Buchdaten\Beispiele\11_Web_Mail\
' Dateiname   11_07_WebQuery.xls
' Modul       mdl_01_WebQuery
' =====

```

```

Sub WebQuery()
    With ActiveSheet.QueryTables.Add(Connection:= _
        "URL;http://www.quoteline.com/lis.asp?std_id=64&l=D", _
        Destination:=Range("A1"))
        .Name = "MyQueryTable"
        .FieldNames = True
        .RowNumbers = False
        .FillAdjacentFormulas = False
        .PreserveFormatting = True
        .RefreshOnFileOpen = False
        .BackgroundQuery = True
        .RefreshStyle = xlInsertDeleteCells
        .SavePassword = False
        .SaveData = True
        .AdjustColumnWidth = True
        .RefreshPeriod = 0
        .WebSelectionType = xlSpecifiedTables
        .WebFormatting = xlWebFormattingNone
        .WebTables = "30"
        .WebPreFormattedTextToColumns = True
        .WebConsecutiveDelimitersAsOne = True
        .WebSingleBlockTextImport = False
        .WebDisableDateRecognition = False
        .WebDisableRedirections = False
        .Refresh BackgroundQuery:=False
    End With
End Sub

```

Die Bedeutung der einzelnen Codezeilen sowie verwendbare Konstanten können Sie der nachfolgenden Tabelle entnehmen:

Eigenschaft	Konstanten	Beschreibung
FieldNames		Spaltenüberschrift
RowNumbers		Zeilennummerierung
FillAdjacentFormulas		Formeln
PreserveFormatting		Formatierungen (Pivot)
RefreshOnFileOpen		Aktualisierung (Pivot)
BackgroundQuery		Abfrage im Hintergrund ausführen
RefreshStyle	xlInsertDeleteCells	Teile von Zeilen werden hinzugefügt oder gelöscht
	xlOverwriteCells	Keine neuen Zeilen hinzufügen, nur bestehende Zeilen überschreiben
	xlInsertEntireRows	Falls erforderlich, ganze Zeilen hinzufügen
SavePassword		Kennwortinformationen, falls erforderlich
SaveData		Pivot-Informationen speichern
AdjustColumnWidth		Spaltenbreite anpassen
RefreshPeriod		Anzahl autom. Aktualisierungen pro Minute
WebSelectionType	xlEntirePage	Gesamte Seite importieren
	xlAllTables	Alle Tabellen importieren (Standard)
	xlSpecifiedTables	Nur bestimmte Zeilen der Tabelle importieren
WebFormatting	xlWebFormattingAll	Alle Formatieren übernehmen
	xlWebFormattingRTF	Nur RTF-Formatierungen übernehmen
	xlWebFormattingNone	Keine Formatierungen übernehmen (Standard)
WebTables		Durch Komma getrennte Angabe, welche Webseiten importiert werden sollen (Wenn Webselection = xlSpecitiedTables)
WebPreFormatted-TextToColumns		Gibt zurück oder legt fest, ob die Analyse der Daten in HTML <PRE>-Tags der Webseite beim Importieren der Seite in eine Abfragetabelle in Spalten erfolgen soll
WebConsecutive-DelimitersAsOne		True, wenn aufeinander folgende Trennzeichen beim Importieren von Daten aus HTML <PRE>-Tags in einer Webseite in eine Abfragetabelle als einzelnes Zeichen behandelt werden, und die Datenanalyse in Spalten erfolgen soll
WebSingleBlock-TextImport		True, wenn die Daten aus den HTML <PRE>-Tags in der angegebenen Webseite beim Importieren der Seite in eine Abfragetabelle gleichzeitig verarbeitet werden sollen

Tabelle 83: Webabfrage

Grund-lagen

Allge-mein

Datu Zeit

tungen

Steuer elemen

Befehl leisten

Objekt

Dia-gramm

Ereig-nisse

User-Form

Web/Mail

Extern

Ge-misch tes

Specia

Eigenschaft	Konstanten	Beschreibung
WebDisableDate-Recognition		True, wenn Daten, die Datumsangaben darstellen, beim Importieren einer Webseite in eine Abfragetabelle als Text analysiert werden
WebDisable-Redirections		True, falls die Umleitung von Webabfragen für ein QueryTable-Objekt deaktiviert wird. Der Standardwert ist False
Refresh Background-Query		True, wenn Abfragen der PivotTable-Berichte oder der Abfragetabelle asynchron (im Hintergrund) durchgeführt werden

Tabelle 83: Webabfrage (Forts.)

382 Webabfragen aktualisieren

Wenn die Symbolleiste EXTERNE DATEN eingeblendet ist, kann die im Tabellenblatt enthaltene Webabfrage per Klick auf die Schaltfläche DATEN AKTUALISIEREN (rotes Ausrufezeichen) aktualisiert werden. Die Daten werden dabei neu aus dem Inter- oder Intranet übertragen und an die Excel-Tabelle übergeben. Um die Aktualisierung durchführen zu können, muss eine der Zellen innerhalb der bereits bestehenden Daten aktiviert sein.

Die Aktualisierung kann je nach Datenmenge ein Weilchen dauern. In der Statuszeile wird während des Übertragens eine sich drehende Weltkugel angezeigt. Sobald die Weltkugel wieder verschwunden ist, ist die Übertragung beendet.

Sie können die Daten auch per VBA aktualisieren. Verwenden Sie dazu die Eigenschaft `QueryTable` in Kombination mit der Methode `Refresh`. Eine Verbindung zum Internet muss bestehen.

```
'=====
' Auf CD      Buchdaten\Beispiele\11_Web_Mail\
' Dateiname   11_07_WebQuery.xls
' Modul       mdl_02_RefreshWebQuery
'=====
```

```
Sub RefreshWebQuery()
    ' Daten aktualisieren
    ActiveSheet.Range("A1").QueryTable.Refresh
End Sub
```

Falls sich mehrere Webabfragen auf Ihrem Tabellenblatt befinden, können Sie unter Zuhilfenahme einer `For...Each`-Schleife alle gleichzeitig aktualisieren.

```
'=====
' Auf CD      Buchdaten\Beispiele\11_Web_Mail\
' Dateiname   11_07_WebQuery.xls
' Modul       mdl_02_RefreshWebQuery
'=====
```

```
Sub RefreshQueryTableSheet()
    Dim qt As QueryTable
```

```
For Each qt In Worksheets(1).QueryTables
    qt.Refresh
Next
End Sub
```

Wenn sich mehrere Webabfragen in Ihrer Mappe befinden, können Sie alle gleichzeitig aktualisieren, indem Sie die Anweisung `ActiveWorkbook.RefreshAll` verwenden.

```
'=====
' Auf CD      Buchdaten\Beispiele\11_Web_Mail\
' Dateiname  11_07_WebQuery.xls
' Modul      mdl_02_RefreshWebQuery
'=====

Sub RefreshAllQueryTables()
    ActiveWorkbook.RefreshAll
End Sub
```

383 Eine Webabfrage bereinigen

Beim Import von Webseiten kann es vorkommen, dass leere Zeilen oder Spalten vorhanden sind. Diese lassen sich durch einen VBA-Code schnell entfernen.

Achtung

Der folgende Code ist mit Vorsicht zu genießen. Er löscht Zeilen und Spalten aufgrund von Leereinträgen in der *ersten* Zeile und Spalte, auch wenn weitere Zellen dieser Zeilen und Spalten gefüllt sind! Der Code kann somit nicht immer angewendet werden. Erstellen Sie vor dem Ausführen des Codes sicherheitshalber eine Kopie Ihres Tabellenblattes, denn VBA-Prozeduren können Sie nicht »rückgängig« machen, außer Sie würden dazu eigens eine Prozedur erstellen.

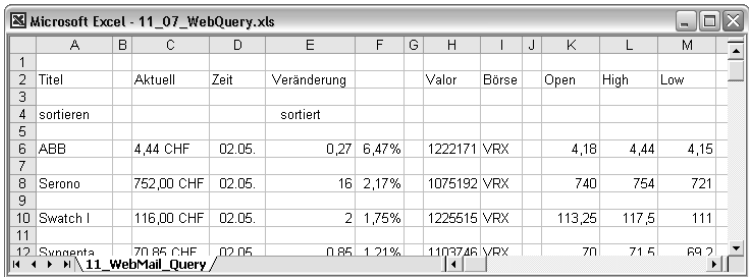


Abbildung 314: Anzeige vor dem Bereinigen der Leerzeilen und -spalten

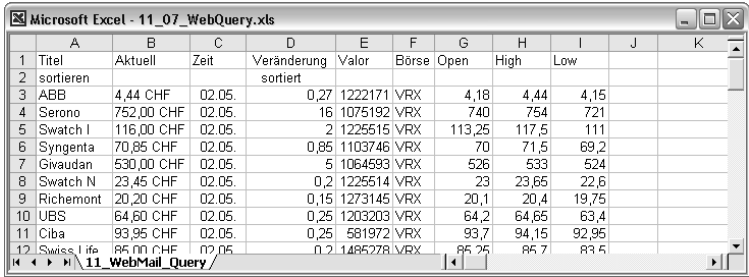


Abbildung 315: Leerzeilen und -spalten bereinigt

Grundlagen

Allgemein

Datum und Zeit

Formeln

Steuerungselemente

Befehlsliste

Objekt

Diagramm

Ereignisse

User-Form

Web/Mail

Extern

Gemischtes

Spezial

Um die Leerzeilen und -spalten zu löschen, bedienen wir uns der Methode `SpecialCells` in Verbindung mit der Konstanten `xlCellTypeBlanks` und der Methode `Delete`.

```
'=====
' Auf CD      Buchdaten\Beispiele\11_Web_Mail\
' Dateiname   11_07_WebQuery.xls
' Modul       mdl_03_CleanWebQuery
'=====

Sub CleanWebQuery()
    ' Leere Zeilen entfernen
    Columns(1).Cells.SpecialCells(xlCellTypeBlanks) _
        .EntireRow.Delete

    ' Leere Spalten entfernen
    Rows(1).Cells.SpecialCells(xlCellTypeBlanks) _
        .EntireColumn.Delete
End Sub
```

384 Webabfrage aktualisieren und bereinigen (kombiniert)

Wenn Sie die beiden vorangegangenen Rezepte miteinander kombinieren möchten, so dass zuerst die Aktualisierung ausgeführt wird und danach die leeren Zeilen und Spalten entfernt werden, werden Sie feststellen, dass dies nicht funktioniert, obwohl die `Refresh`-Anweisung an erster Stelle steht. Der Grund liegt darin, dass die Webabfrage länger dauert als die Bereinigung der Leereinträge. Das bedeutet, dass die Daten im Tabellenblatt noch während der Aktualisierung bereinigt werden. Die Daten der Webabfragen werden erst nach der Bereinigung der Leereinträge neu übertragen.

Wenn Sie dennoch nicht zwei Prozeduren nacheinander manuell aufrufen möchten, können Sie in der ersten Prozedur, welche die Daten aktualisiert, die zweite Prozedur aufrufen. In der ersten Prozedur fügen wir am Ende eine zusätzliche Codezeile ein (`Application.OnTime`). Wir legen zudem mittels `TimeValue` fest, dass die zweite Prozedur erst nach fünf Sekunden gestartet werden soll. Je nach Datenmenge, die übertragen werden soll, reichen die fünf Sekunden nicht aus und der Wert muss erhöht werden.

```
'=====
' Auf CD      Buchdaten\Beispiele\11_Web_Mail\
' Dateiname   11_07_WebQuery.xls
' Modul       mdl_04_RefreshAndClean
'=====

Sub RefreshWebQueryCombi()
    ' Daten aktualisieren
    ActiveSheet.Range("A1").QueryTable.Refresh

    ' Ausführung um 5 Sekunden verzögern
    Application.OnTime Now + TimeValue("00:00:05"), _
        "CleanWebQueryCombi"
End Sub
```

```
'=====
' Auf CD      Buchdaten\Beispiele\11_Web_Mail\
' Dateiname   11_07_WebQuery.xls
' Modul       mdl_04_RefreshAndClean
'=====

Sub CleanWebQueryCombi()
    ' Leere Zeilen entfernen
    Columns(1).Cells.SpecialCells(xlCellTypeBlanks) _
        .EntireRow.Delete

    ' Leere Spalten entfernen
    Rows(1).Cells.SpecialCells(xlCellTypeBlanks) _
        .EntireColumn.Delete
End Sub
```

385 Webabfragen löschen

Wenn wir von »Webabfragen löschen« sprechen, ist damit lediglich die Verbindung zur Datenquelle gemeint. Bei einer Webabfrage also die Verbindung zum Inter- oder Intranet. Beim Löschen einer Webabfrage werden somit nicht die Daten selbst gelöscht, sondern es wird lediglich die Leitung zur Quelle getrennt. Danach kann keine Aktualisierung mehr stattfinden. Die Webabfrage müsste neu erstellt werden.

Um die Verbindung zu einer bestimmten Webabfrage zu trennen, geben Sie eine Zelle in der Datenquelle an und verwenden dann die Eigenschaft `QueryTable` in Kombination mit der Methode `Delete`.

```
'=====
' Auf CD      Buchdaten\Beispiele\11_Web_Mail\
' Dateiname   11_07_WebQuery.xls
' Modul       mdl_05_DeleteQueryConnection
'=====

Sub DeleteOneQueryConnection()
    Range("A1").QueryTable.Delete
End Sub
```

Um alle Webabfragen eines Tabellenblattes zu löschen, setzen wir zusätzlich eine `For...Each`-Schleife ein.

```
'=====
' Auf CD      Buchdaten\Beispiele\11_Web_Mail\
' Dateiname   11_07_WebQuery.xls
' Modul       mdl_05_DeleteQueryConnection
'=====

Sub DeleteAllQueryConnections()
    Dim qt As QueryTable

    For Each qt In ActiveSheet.QueryTables
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

```

    qt.Delete
Next qt
End Sub

```

386 Einen Webbrowser einbinden

Sie können alternativ zu einer Webabfrage oder einem Hyperlink auch ein Browser-Fenster in Ihr Tabellenblatt einbinden. Es handelt sich dabei um ein ActiveX-Steuerelement aus der Symbolleiste STEUERELEMENT-TOOLBOX.

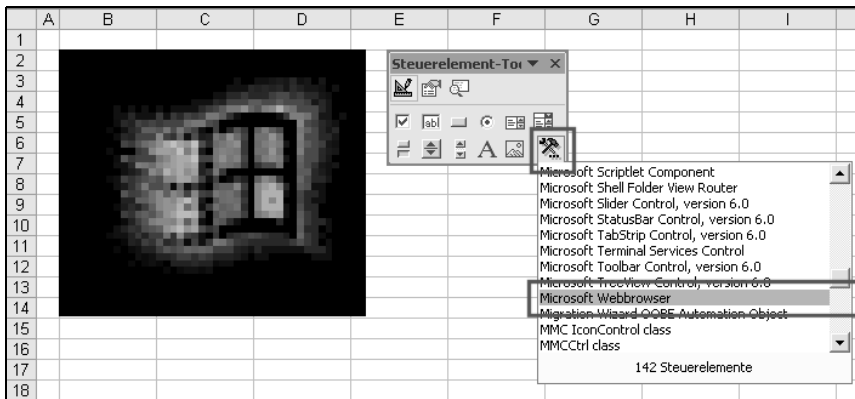


Abbildung 316: Ein Browser-Fenster einbinden

Vorgehen:

1. Um ein Browser-Fenster einzubinden, blenden Sie die Symbolleiste STEUERELEMENT-TOOLBOX ein
2. Klicken Sie auf die letzte darin enthaltene Schaltfläche WEITERE STEUERELEMENTE
3. Ein DropDown-Menü wird angezeigt
4. Scrollen Sie mit der Bildlaufleiste bis zum Eintrag MICROSOFT-WEBBROWSER
5. Klicken Sie die Auswahl an
6. Der Mauszeiger verändert sich in ein Kreuz
7. Ziehen Sie in Ihrem Tabellenblatt einen Rahmen
8. Wechseln Sie in den VBA-Editor
9. Fügen Sie ein Modul ein
10. Geben Sie den folgenden Code ein
11. Hinter der Anweisung `Navigate` geben Sie in Anführungs- und Schlusszeichen die gewünschte Webadresse ein

Falls Sie mehrere Webbrowser in Ihr Tabellenblatt einbinden möchten, achten Sie beim Erstellen der VBA-Prozedur auf die Nummerierung der Browser-Fenster (`WebBrowser1`, `WebBrowser2`, `WebBrowser3` usw.).

Nach dem Ausführen des Codes wird die Webseite im Browser-Fenster angezeigt. Stellen Sie vor dem Ausführen des Codes sicher, dass eine Verbindung zum Internet besteht.

```
'=====
' Auf CD      Buchdaten\Beispiele\11_Web_Mail\
' Dateiname  11_08_WebBrowser.xls
' Modul      mdl_01_WebBrowser
'=====

Sub IncludeWebbrowser()
    ActiveSheet.WebBrowser1.Navigate "http://www.jumper.ch"
End Sub
```

Hinweis

Falls Sie im Nachhinein die Größe des Browser-Fensters verändern möchten, wechseln Sie unter Zuhilfenahme der Symbolleiste **STEUERELEMENT-TOOLBOX** (erste Schaltfläche **ENTWURFSMODUS**) in den Entwurfsmodus. Im ersten Moment sieht es so aus, als wäre der Webbrowser nicht mehr vorhanden, da das Fenster nicht mehr sichtbar ist. Klicken Sie auf die Stelle, wo Sie den Webbrowser eingefügt haben. An den Ecken- und Seitenmarkierungen ist er nun wieder zu erkennen. Ziehen Sie mit gedrückter linker Maustaste eine der Markierungen, bis die gewünschte Größe erreicht ist. Schalten Sie den Entwurfsmodus wieder aus, damit die Webseite wieder angezeigt wird.

387 Eine XML-Datei erzeugen

XML (eXtensible Markup Language) ist seit der Version 2002 (XP) ein Thema in Excel. In dieser Version haben Sie erstmals die Möglichkeit, eine Mappe im Format **.xml* abzuspeichern. Wenn Sie versuchen, eine Excel-Mappe im XML-Format abzuspeichern, werden Sie sehr schnell feststellen, dass nur reine Daten erlaubt sind. XML unterstützt keine Objekte wie Grafiken oder Diagramme. Auch VBA-Code kann nicht in ein XML-Dokument abgespeichert werden. Beim Speichern einer Mappe mit Objekten, die nicht unterstützt werden, erhalten Sie eine entsprechende Meldung. Wenn man sich den Quellcode ansieht, ist ein unglaublich umfangreicher XML-Code hinterlegt, selbst für die kleinste in Excel erstellte Datenbank. Dem Quellcode ist beispielsweise zu entnehmen, wer die Datei erstellt hat, wann sie zuletzt gespeichert wurde und so weiter.

Hinweis

Microsoft stellt auf einer ihrer deutschsprachigen Webseiten eine Übersicht zur Verfügung, der Sie entnehmen können, welche Einschränkungen für XML-Tabellen bestehen:
<http://office.microsoft.com/germany/assistance/2002/articles/xlFeaturesLimitationXml-Spread.aspx>

Wenn Sie mit der Version 2003 von Office arbeiten, werden Sie erkennen, dass der Trend immer mehr Richtung XML geht, denn dort wurden zusätzlich einige entscheidende neue Features eingebaut, wie Sie hier noch kennen lernen werden.

Was ist XML?

Mittels XML haben Sie die Möglichkeit, reine Textdateien in hierarchischer Struktur aufzubauen. Der Aufbau ähnelt HTML (HyperText Markup Language), ist jedoch nicht damit zu verwechseln. HTML ist eine Seitenbeschreibungssprache, die es erlaubt, Dokumente unter Zuhilfenahme von fest vorgeschriebenen und standardisierten Tags zu beschreiben, sprich: zu formatieren. Beide Markup-Sprachen wurden durch das *W3C* (World Wide Web Consortium) standardisiert.

In XML werden zwar auch spitze Klammern verwendet, den darin enthaltenen Text können Sie jedoch selbst bestimmen.

```
<MeinVorname> Monika </MeinVorname>
```

Diese Tags beschreiben die Datenfelder und organisieren in ihrer Struktur den hierarchischen Aufbau der Datenbank. Es würde zu weit führen, Sie an dieser Stelle in das Thema XML einzuführen, zumal Kenntnisse im Einsatz mit Office zwar von Vorteil, jedoch nicht unbedingt erforderlich sind. Falls Sie sich näher mit dem Thema XML befassen möchten, empfehlen wir Ihnen, entsprechende Fachliteratur zu erwerben, oder im Internet danach zu suchen. Literaturvorschläge zu XML finden Sie am Ende des Rezepts 390.

Wo ist denn nun der Sinn und Zweck von XML in Office?

Die Idee von Microsoft ist es, eine Möglichkeit zu schaffen, die Office-Anwendungen so zu optimieren, dass die Plattform XML genutzt werden kann. Es soll möglich sein, Daten, die im XML-Format vorliegen, auf einfache Weise jederzeit auch in Excel, Word, Access usw. verfügbar zu machen. XML ist als Basis dazu wie geschaffen. Durch den Entscheid, XML zu nutzen, haben Sie erstmals die Gelegenheit, ein einheitliches und unabhängiges Datenformat zu verwenden. Sie können sich damit eine Menge Arbeit ersparen, denn bisher konnte der Datenaustausch zwischen den einzelnen Applikationen ein echtes Problem darstellen, da jede Anwendung mit einem eigenen Datenformat arbeitet. Oftmals mussten die Daten ausgeschnitten und in der Zielapplikation wieder eingefügt werden. Damit war es jedoch meistens noch nicht getan, denn in der Regel bestand der große Aufwand in der Nachformatierung der Daten.

Wie wird eine XML-Datei erstellt?

Um eine XML-Datei zu erstellen, können Sie mit einem Texteditor arbeiten. Unter Windows ist es der »Notepad«. Wichtig dabei ist, dass Sie die Datei mit der Endung *.xml abspeichern. Im folgenden Beispiel werden wir eine sehr einfache, kleine Datenbank in XML erstellen.

Damit der Server erkennt, dass es sich um XML-Daten handelt, wird in der ersten Zeile eine entsprechende Deklaration vorgenommen. Danach können die Daten hierarchisch aufgebaut werden:

```
'=====
' Auf CD      Buchdaten\Beispiele\11_Web_Mail\
' Dateiname  11_09_MyXML.xml
'=====

<?xml version="1.0"?>

<Daten>
  <Autoren>
    <Vorname>Monika</Vorname>
    <Nachname>Weber</Nachname>
    <Land>Schweiz</Land>
  </Autoren>

  <Autoren>
    <Vorname>Melanie</Vorname>
    <Nachname>Breden</Nachname>
    <Land>Deutschland</Land>
  </Autoren>
</Daten>
```

Zur Überprüfung der Korrektheit der verwendeten Tags können Sie die Datei mit dem Internet Explorer oder einem anderen Browser öffnen. Falls fehlerhafte Anweisungen vorliegen, wird

Ihnen der Browser die entsprechende Stelle kennzeichnen, wie Sie der Abbildung 317 entnehmen können.

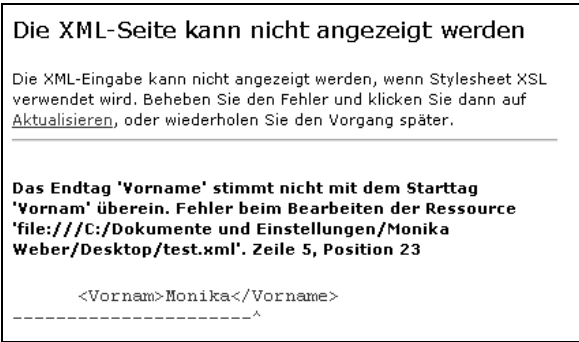


Abbildung 317: Fehler in XML-Codes werden entsprechend gekennzeichnet

Wenn die Daten in korrekter Form vorliegen, präsentiert sich die Anzeige im Browser-Fenster wie folgt:

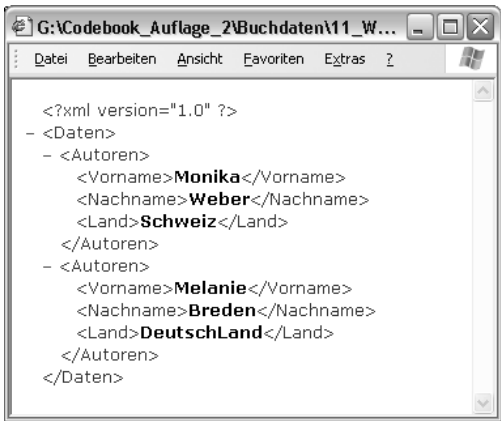


Abbildung 318: Anzeige einer XML-Datei im Internet Explorer

Ab Excel 2002 besteht, wie bereits erwähnt, die Möglichkeit, Daten zu lesen, die im Format *.xml vorliegen. Sie brauchen dazu lediglich die XML-Datei zu öffnen. Geben Sie in Excel im Dialogfenster ÖFFNEN das entsprechende Dateiformat an. Wie die XML-Daten sich in Excel präsentieren, können Sie der Abbildung 319 entnehmen. Das Beispiel zeigt den XML-Code, den wir weiter vorne erstellt haben.

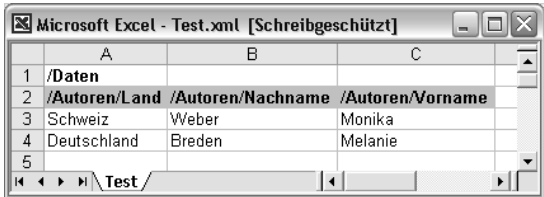


Abbildung 319: XML-Daten in Excel dargestellt

388 Import und Export von XML-Dateien

In der Office-Version 2003 wurde die Nutzung von XML erheblich verbessert. Sie haben nun die Möglichkeit, mit so genannten Schemas zu arbeiten. Bereits beim Öffnen einer XML-Datei wird dies deutlich, denn neu wird ein Dialogfenster angezeigt, das drei Optionsfelder anbietet. Damit kann ausgewählt werden, in welcher Form die Daten importiert werden sollen:

- ▶ Als eine XML-Liste
- ▶ Als eine schreibgeschützte Arbeitsmappe
- ▶ XML-Quell-Aufgabenbereich verwenden

Wenn Sie sich für den ersten Punkt entscheiden, präsentiert sich Ihnen eine fertige wohlgeformte Liste mit bereits vorhandenem Autofilter. Sie sind also bereits mit einem weiteren neuen Feature in Excel 2003 konfrontiert. Zu dieser Liste lässt sich eine eigene Symbolleiste einblenden. Die Symbolleiste nennt sich LISTE UND XML.



Abbildung 320: Excel-Liste aus XML-Datei

Wenn Sie sich beim Import für das zweite Optionsfeld entscheiden, geschieht dasselbe wie in der Excel-Version 2002. Die Datei wird ganz gewöhnlich importiert. Die Mappe ist schreibgeschützt.

Sehr interessant ist der dritte Punkt. Wenn Sie diesen auswählen, wird am rechten Rand des Excel-Fensters die Struktur der XML-Datei angezeigt. Sie können nun selbst, nach Belieben, die Datenfelder zusammensetzen. Das Prinzip erinnert an Pivot-Tabellen, die ebenfalls flexibel nach Wunsch gestaltet und umgestellt werden können. Sie können die Felder aus der XML-Quelle (rechterhand) in das Tabellenblatt ziehen und nach Belieben ordnen. Zuerst wird eine leere Liste angezeigt, die lediglich Überschriften enthält. Ein Klick auf das rote Ausrufezeichen in der Symbolleiste LISTE UND XML importiert und aktualisiert die Daten. Um die Daten, ähnlich wie in Pivot-Tabellen, umzustellen, packen Sie das zu verschiebende Feld an einem der äußeren Rahmen und ziehen es in die gewünschte Zelle.

Den Weg, um Daten eines Schemas zu importieren, kennen Sie nun. Wie verhält es sich jedoch mit dem Export? Wenn die Daten importiert wurden, können Sie diese über die Symbolleiste LISTE UND XML, Schaltfläche EXPORT auch wieder exportieren. Anders verhält es sich bei einer selbst erstellten Liste. Eine Liste zu erstellen ist recht einfach. Erfassen Sie die Daten in Excel wie gewohnt. Klicken Sie dann mit der rechten Maustaste in den Datenbereich und wählen Sie aus dem Kontextmenü den Eintrag LISTE ERSTELLEN. Ein Klick in der Symbolleiste LISTE UND XML auf die Schaltfläche EXPORT reicht aus, um die Daten in ein XML-Format zu exportieren.

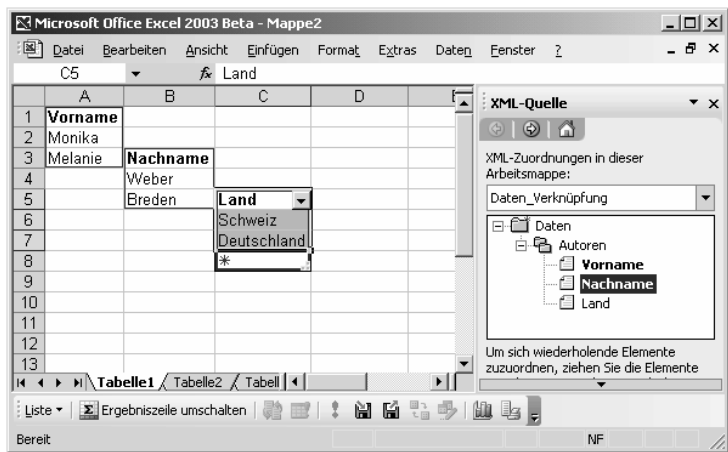


Abbildung 321: XML-Felder nach Belieben platzieren

Um einen Export zu realisieren, können Sie auch einen VBA-Code schreiben. Ob die Daten in Form einer LISTE vorliegen, ist dabei unerheblich. Um per VBA eine XML-Datei zu erstellen, müssen wir mit einer entsprechenden Bibliothek arbeiten, denn bei XML handelt es sich um ein externes Dateiformat (mehr zum Thema »Extern« erfahren Sie in der entsprechenden Kategorie). Sie können zwischen »Early Binding« und »Late Binding« wählen. Bei »Early Binding« legen Sie *zuerst* den Verweis auf die gewünschte Bibliothek fest. In Excel 2002 nennt sich die Bibliothek MICROSOFT XML V3.0. In Excel 2003 lautet der Name der Bibliothek MICROSOFT XML V5.0. Nach dem Aktivieren des Verweises auf die Bibliothek können Sie mit der entsprechenden Deklaration beginnen. Für die Version 2002 lautet der Befehl:

```
Dim xml As New MSXML2.DOMDocument30
```

Damit wird auf das *DOM* (Document Object Model) zugegriffen. In der Version 2003 muss das *DOMDocument30* durch *DOMDocument50* ersetzt werden.

Wenn Sie sich für »Late Binding«, also für das Erzeugen des Objektes während der Laufzeit, entscheiden, lautet die Anweisung:

```
Dim xml As Object  
Set xml = CreateObject("Microsoft.MXMLDOM")
```

In unserem Beispiel verwenden wir »Early Binding« und setzen somit den manuellen Verweis auf die entsprechende Bibliothek. Um Ihnen anhand dieses Beispielen deutlich zu machen, wie ein Export in eine XML-Datei realisiert werden kann, werden wir die entsprechenden Daten direkt in der Prozedur unterbringen. Das Ganze ist so leichter verständlich. Sie haben alternativ natürlich auch die Möglichkeit, die Daten aus dem Tabellenblatt zu beziehen. Wie Sie dies realisieren können, werden wir Ihnen anhand eines zweiten Codes zeigen.

Die Beschreibung der einzelnen Codeabschnitte finden Sie, kursiv geschrieben, direkt in der Prozedur:

Grund-lagen

Allge-mein

Datu Zeit

tungen

Steuer elemen

Befehl leisten

Objekt

Dia-gramm

Ereig-nisse

User-Form

Web/Mail

Extern

Ge-misch tes

Specia

```

' =====
' Auf CD      Buchdaten\Beispiele\11_Web_Mail\
' Dateiname   11_10_CreateXML.xls
' Tabelle     11_WebMail_CreateXML1
' Modul       mdl_01_CreateXML
' =====

' Folgender Verweis muss gesetzt sein:
' Microsoft XML, v3.0
Sub CreateXML()
    ' Mittels DOM das Dokument generieren
    Dim xml As New MSXML2.DOMDocument30
    Dim xmlElmAuthor As MSXML2.IXMLDOMElement
    Dim xmlElmFirstname As MSXML2.IXMLDOMElement
    Dim xmlElmSurname As MSXML2.IXMLDOMElement
    Dim xmlElmCountry As MSXML2.IXMLDOMElement

    ' Versionsnummer und erste Hierarchie einfügen
    xml.loadXML "<?xml version='1.0'?><Daten/>"

    ' XML-Felder erzeugen (createElement)
    Set xmlElmAuthor = xml.createElement("Autoren")
    Set xmlElmFirstname = xml.createElement("Vorname")
    Set xmlElmSurname = xml.createElement("Name")
    Set xmlElmCountry = xml.createElement("Land")

    ' Daten an XML-Felder übergeben (appendChild)
    With xmlElmAuthor
        .appendChild(xmlElmFirstname).Text = "Monika"
        .appendChild(xmlElmSurname).Text = "Weber"
        .appendChild(xmlElmCountry).Text = "Schweiz"
    End With

    ' Daten anhängen
    xml.documentElement.appendChild xmlElmAuthor

    ' Vorgang für zweiten Datensatz wiederholen
    Set xmlElmAuthor = xml.createElement("Autoren")
    Set xmlElmFirstname = xml.createElement("Vorname")
    Set xmlElmSurname = xml.createElement("Name")
    Set xmlElmCountry = xml.createElement("Land")

    With xmlElmAuthor
        .appendChild(xmlElmFirstname).Text = "Melanie"
        .appendChild(xmlElmSurname).Text = "Breden"
        .appendChild(xmlElmCountry).Text = "Deutschland"
    End With

    xml.documentElement.appendChild xmlElmAuthor

    ' Datei speichern
    xml.Save "C:\XMLTest.xml"

```

```
' Erfolgreiches Schreiben bestätigen
MsgBox "Die Datei '" & C:\XMLText.xml & "' wurde erstellt"
End Sub
```

Wir wollen das Ganze nun auf unsere kleine Excel-Datenbank anwenden (siehe Abbildung 322). Dazu verwenden wir eine `For`-Schleife, mittels welcher die Daten an die XML-Felder übergeben werden. Dabei spielt es keine Rolle, wie viele Datensätze in Excel vorhanden sind. Die Größe der Prozedur bleibt immer die gleiche. Zu Beginn der Schleife muss lediglich der Datenbereich festgelegt werden. In unserem Beispiel ist das der Bereich `A2:C3`. Bei der obigen Prozedur musste pro Datensatz ein komplett neuer Codeabschnitt eingefügt werden. Das ist bei dieser Prozedur unnötig.

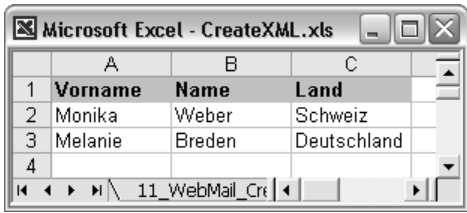


Abbildung 322: Excel-Daten für das XML-File

```
' =====
' Auf CD      Buchdaten\Beispiele\11_Web_Mail\
' Dateiname   11_10_CreateXML.xls
' Tabelle     11_WebMail_CreateXML2
' Modul       mdl_02_CreateXMlExcel
' =====

' Folgender Verweis muss gesetzt sein:
' Microsoft XML, v3.0
Sub CreateXMlExcel()
    ' Mittels DOM das Dokument generieren
    Dim xml As New MSXML2.DOMDocument30
    Dim xmlElmAuthor As MSXML2.IXMLDOMElement
    Dim xmlElmFirstname As MSXML2.IXMLDOMElement
    Dim xmlElmSurname As MSXML2.IXMLDOMElement
    Dim xmlElmCountry As MSXML2.IXMLDOMElement

    Dim i As Integer

    ' Versionsnummer und erste Hierarchie einfügen
    xml.loadXML "<?xml version='1.0'><Daten/>"

    ' Mittels einer Schleife die Zeilen der Excel-Datenbank
    ' durchlaufen
    For i = 1 To Range("A2:C3").Rows.Count
        ' XML-Felder erzeugen (createElement)
        Set xmlElmAuthor = xml.createElement("Autoren")
        Set xmlElmFirstname = xml.createElement("Vorname")
        Set xmlElmSurname = xml.createElement("Name")
        Set xmlElmCountry = xml.createElement("Land")
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

```

' Daten an XML-Felder übergeben (appendChild)
With xmlElmAuthor
    .appendChild(xmlElmFirstname).Text = Cells(i + 1, 1).Value
    .appendChild(xmlElmSurname).Text = Cells(i + 1, 2).Value
    .appendChild(xmlElmCountry).Text = Cells(i + 1, 3).Value
End With
' Daten anhängen
xml.documentElement.appendChild xmlElmAuthor
Next i

' Datei speichern
xml.Save "C:\XMLTest.xml"

' Erfolgreiches Schreiben bestätigen
MsgBox "Die Datei " & "C:\XMLTest.xml" & " wurde erstellt"
End Sub

```

Um den Code bzw. die Datei einzusehen, die erzeugt wurden, sehen Sie sich die Datei *C:\XML-Test.xml* an. Wenn Sie die Datei per Doppelklick öffnen, präsentiert Ihnen Ihr Browser-Fenster die XML-Datei, die per VBA erzeugt wurde. Sie entspricht der Abbildung 318.

389 XML-SS-Schemas

Seit der Excel-Version 2000 ist es möglich, mit XML-Schemas zu arbeiten. Der Oberbegriff dazu lautet XML-SS (eXtensible Markup Language-SpreadSheet). Wenn Sie sich mit XML-Schemas befassen, wird auch langsam klar, weshalb Excel unter anderem so viel XML-Code erzeugt, selbst wenn eine leere Excel-Tabelle im Format **.xml* gespeichert wird.

Was ist ein XML-SS-Schema?

Die XML-Dateien, mit denen wir bisher gearbeitet haben, entbehrten jeglicher Formatierungen. Es wurden keine Farben verwendet, die Spaltenbreite wurde nicht festgelegt, es wurden keine Zahlenformatierungen vorgenommen usw. In einem XML-SS-Schema werden sämtliche Informationen zu einem Tabellenblatt gespeichert, die den Formatierungen entsprechen. XML-SS-Schemas sind somit die Formatvorlagen für Ihre XML-Dateien. Sie sind im Aufbau ähnlich dem CSS (Cascading Style Sheets), welches für HTML-Dateien eingesetzt wird.

Anhand eines kleinen Beispiels wollten wir veranschaulichen, was im Hintergrund von Excel in Bezug auf XML geschieht, wenn wir eine Zelle formatieren. Wir nehmen dazu die Zelle A1. Zuerst werden wir die Zelle ein klein wenig manuell formatieren. Zu viele Formatierungen dürfen wir nicht vornehmen, denn sonst reicht die Kapazität der MsgBox (max. 512 Zeichen) nicht mehr aus, um den gesamten XML-Code anzuzeigen. Der Zelle wird ein roter Hintergrund zugewiesen. Die Schrift wird in gelber Farbe formatiert und in die Zelle ein Name geschrieben.

Um den XML-Code anzeigen zu lassen, verwenden wir den folgenden VBA-Code. Beachten Sie die Konstante `xlRangeValueXMLSpreadsheet`, welche nach `Value` verwendet wird. Sie veranlasst, dass die Zelle A1 in XML-Codierung dargestellt wird:

```

' =====
' Auf CD      Buchdaten\Beispiele\11_Web_Mail\
' Dateiname  11_11_GetXMLCode.xls

```

```
' Modul      mdl_01_GetXMLCode.xls
' =====

Sub GetXMLCode()
    MsgBox ActiveSheet.Range("A1"). _
        Value(xlRangeValueXMLSpreadsheet)
End Sub
```

Die Informationsmenge, die in der Dialogbox ausgegeben wird, ist auf den ersten Blick verblüffend, da wir ja nur eine einzelne Zelle ausgegeben haben:



Abbildung 323: Ausgabe des XML-Codes der Zelle A1

Tipp

Wenn Sie in unserem VBA-Code die Anweisung `MsgBox` durch `Debug.Print` ersetzen, werden die XML-Codezeilen ins Direktfenster geschrieben. Wenn das Direktfenster in Ihrem VBA-Editor nicht angezeigt wird, blenden Sie es ein, indem Sie die Tastenkombination `[Strg]+[G]` drücken. Alternativ können Sie in Ihrem Direktfenster auch direkt die Codezeile eingeben, angeführt von einem Fragezeichen `?Activecell.Value(xlRangeValueXMLSpreadsheet)`. Von da aus können Sie den erzeugten Code bequem kopieren und weiterverwenden.

Die Daten sind hierarchisch strukturiert, wie Sie das schon von Excel und VBA her kennen. Das oberste Element der Hierarchie ist die Mappe `<Workbook>`. Innerhalb des `<Workbook>`-Tags sind Deklarationen enthalten, die immer gleich sind. Der Webserver kann daran erkennen, dass es sich um eine Office-Applikation Excel von Microsoft handelt.

Eine Stufe tiefer in der Hierarchie folgt das Tabellenblatt `<Worksheet>` und schließlich die Tabelle `<Table>`, die aufgebaut werden soll. In der Tabelle enthalten sind Zeilen `<Row>` und Zellen `<Cell>`. Das Tag für die Daten, welche in den Zellen dargestellt werden, lautet `<Data>`. Denken Sie beim Erstellen eines XML-Schemas daran, dass jedes Tag `<...>` mit einem Endtag `</...>` abgeschlossen werden muss. Der Querstrich innerhalb der spitzen Klammern deutet auf das Ende des Tags hin.

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

Noch bevor die Tabelle aufgebaut wird, werden die Stylesheets, also die Formatvorlagen erstellt. Dabei wird das Tag `<Styles> ... </Styles>` verwendet. Innerhalb dieses Tags werden die Informationen für jede einzelne Formatvorlage in einem `<Style>...</Style>`-Tag festgelegt. Jedem Style wird eine eindeutige Identifikationsnummer (ID) zugewiesen. Auf diese ID wird Bezug genommen, wenn die Tabelle aufgebaut wird. Die ID kann beispielsweise an eine Zelle übergeben werden. Die Zelle wird dann mit den entsprechenden Formatierungen angezeigt. Wenn Sie die Abbildung 323 betrachten, werden Sie nun erkennen, dass der XML-Code zwei Formatvorlagen beinhaltet. Eine mit der ID `Default`, welche für die Standardwerte steht, und eine weitere mit der ID `s32`, der andere Formate zugewiesen sind. In der zweiten Formatvorlage sind unsere Formatierungen für Schriftfarbe und Hintergrundfarbe enthalten (`Font` und `Interior`).

Nachdem die Formatvorlagen festgelegt sind, kann die Tabelle `<Table>` im Tabellenblatt `<Worksheet>` aufgebaut werden. Im `<Table>`-Tag selbst, also noch vor dem Ende der spitzen Klammer, ist zu erkennen, wie viele Spalten und Zeilen die Tabelle umfasst. In unserem Beispiel haben wir nur eine Zelle, die Zelle A1, aufgerufen. Daher zweimal eine 1. Die Standardbreite der Spalte wird ebenfalls innerhalb des `<Table>`-Tags festgelegt (60). Die Werte sind immer in Pixel angegeben. Innerhalb des `<Table>`-Tag werden die Zeile `<Row>` und schließlich die Zelle `<Cell>` mit den Daten `<Data>` aufgebaut. Achten Sie beim Tag `<Cell>` darauf, dass hier die ID »s32« des zweiten Style verwendet wird.

Hinweis

Microsoft bietet auf ihren englischen MSDN-Webseiten eine Übersicht zu der Hierarchie und den Formatierungsmöglichkeiten an:

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnexcel2k2/html/odc_xmlss.asp

Da die Liste sehr umfangreich ist, werden wir darauf verzichten, diese hier aufzuführen.

390 XSL(T) Stylesheets für XML

Wie Sie im vorangegangenen Rezept erfahren haben, können Sie innerhalb von Ihren XML-Dateien auch Formatierungen verwenden. Sinnvoller ist es jedoch, die XML-Daten getrennt von den Formatierungen abzulegen. Der Vorteil besteht darin, dass XML-Dateien ohne irgendwelchen Ballaststoff an Formatierungen und somit übersichtlicher aufgebaut werden können. Die Formatierungen werden getrennt in einer eigenen Datei abgelegt. Es sind somit zwei Dateien erforderlich. Speichern Sie beide Dateien im selben Verzeichnis ab. Die Stylesheet-Datei trägt die Endung `*.xsl`. `XSL(T)` steht für eXtensible Stylesheet Language (Transformation). Dieses Format kann sowohl für HTML- als auch Excel-Dateien verwendet werden.

In der XML-Datei sind somit *nur* die Daten und keinerlei Formatierungen enthalten. In der XSL-Datei sind in der Regel keine Daten, sondern nur Informationen zur Formatierung und zur Strukturierung der Datenbank abgelegt.

Um der XML-Datei mitzuteilen, auf welcher Formatvorlage sie basiert, wird in der zweiten Codezeile eine entsprechende Deklaration vorgenommen. In der Deklaration wird der Name der `*.xsl`-Datei angegeben (`href="11_12_codebook.xsl"`). Die XSL-Datei werden wir in einem zweiten Schritt aufbauen.

```
'=====
' Auf CD      Buchdaten\Beispiele\11_Web_Mail\
' Dateiname   11_12_Codebook.xml
'=====
```

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="11_12_codebook.xsl"?>

<Publications>
  <Author>
    <FirstName>Monika</FirstName>
    <LastName>Weber</LastName>
    <ISBN>382731979X</ISBN>
    <Title>Excel Codebook Band 1</Title>
  </Author>

  <Author>
    <FirstName>Monika</FirstName>
    <LastName>Weber</LastName>
    <ISBN>3827321018</ISBN>
    <Title>Excel Codebook Band 2</Title>
  </Author>

  <Author>
    <FirstName>Monika</FirstName>
    <LastName>Weber</LastName>
    <ISBN>3827320674</ISBN>
    <Title>HTML und CSS lernen</Title>
  </Author>

  <Author>
    <FirstName>Melanie</FirstName>
    <LastName>Breden</LastName>
    <ISBN>3827321018</ISBN>
    <Title>Excel Codebook Band 2</Title>
  </Author>
</Publications>
```

Die XSL-Datei kann unter Zuhilfenahme der entsprechenden Tags wie eine HTML-Datei aufgebaut werden. Wichtig sind die Deklarationszeilen zu Beginn der Prozedur. Damit die Transformation stattfinden kann, ist vor allem die zweite Codezeile von Bedeutung. Microsoft bietet seit Internet Explorer Version 5.0 einen so genannten Parser an, der die Transformation von XML zu Microsoft-Programmen ermöglicht. Früher wurde die Codezeile "<http://www.w3.org/TR/WD-xsl>" eingesetzt. Der Standard, der sich hinter dieser Webadresse verbirgt, ist jedoch veraltet, denn er wurde durch das W3C überarbeitet. Leider wurde der neue Standard erst veröffentlicht, nachdem der Microsoft Internet Explorer 5.0 bereits auf dem Markt war. Somit wurde die IE Version 5.0 noch mit dem alten Parser ausgeliefert. Wenn Sie mit der neuen Version arbeiten möchten, verwenden Sie die Codezeile "<http://www.w3.org/1999/XSL/Transform>". Um die Befehle dieser Version nutzen zu können, müssen Sie gegebenenfalls bei Microsoft den neuen Parser »MSXML Parser 3.0« beziehen. Er ist kostenlos verfügbar unter: <http://msdn.microsoft.com/xml>.

Nach der Deklaration beginnt der eigentliche Kern der XSL-Datei. Er wird durch das Tag <HTML> eingeleitet. Die Datei besteht aus zwei Teilen, dem Kopf <HEAD> und dem Körper <BODY>. Im Kopf werden unsere Formatvorlagen <STYLE> festgelegt. Dies sind reine Hintergrundinformationen, sie werden nicht in der Tabelle angezeigt.

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

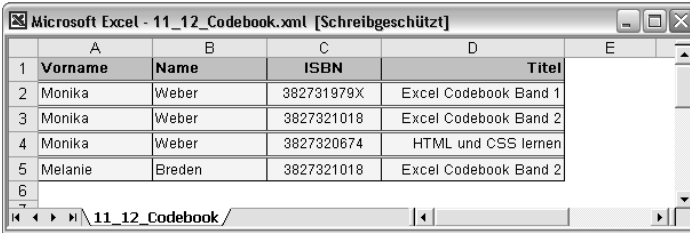
Web/
Mail

Extern

Ge-
misch
tes

Specia

Wir verwenden zwei Formatvorlagen (Stylesheets): `MyHeader` und `MyRow`. Die Vorlage `MyHeader` wird für die Überschrift der Tabelle verwendet: Hellgraue `silver` Hintergrundfarbe und Schriftstil fett `bold`. Die zweite Vorlage `MyRow` wird für die Daten verwendet. Über jeder Zelle soll ein roter Doppelstrich angezeigt werden. Als Hintergrundfarbe haben wir `bisque` gewählt, was in Excel dem Hellgelb entspricht.



	A	B	C	D	E
1	Vorname	Name	ISBN	Titel	
2	Monika	Weber	382731979X	Excel Codebook Band 1	
3	Monika	Weber	3827321018	Excel Codebook Band 2	
4	Monika	Weber	3827320674	HTML und CSS lernen	
5	Melanie	Breden	3827321018	Excel Codebook Band 2	
6					

Abbildung 324: Ausgabe der formatierten XML-Datei in Excel

Nach dem Kopf folgt der Körper `<Body>`. Hier wird die eigentliche Tabelle `<TABLE>` aufgebaut, die in Excel angezeigt wird. Der erste Codeblock innerhalb der Tabelle dient der Formatierung unserer vier Spalten `COLGROUP`. Es wird die Breite `WIDTH` und die Textausrichtung `ALIGN` festgelegt. Wenn Sie die vier Codezeilen weglassen, dann wird der Inhalt der Tabelle standardmäßig linksbündig angezeigt und die Spaltenbreite richtet sich jeweils am längsten Eintrag aus.

Der zweite Codeblock bereitet die Überschrift auf. Hier setzen wir die erste Vorlage ein: `CLASS="MyHeader"`. Das Tag `<TR>` steht für englisch `Table-Row`, zu Deutsch: Zeile. Innerhalb der Zeilen werden die Zellen `<TD>` (`Table-Data`) aufgebaut.

Nachdem die Überschrift aufgebaut ist, können wir die Daten einlesen. An Stelle der eigentlichen Daten, welche ja getrennt in der XML-Datei vorliegen, setzen wir die Feldnamen ein, die wir dort verwendet haben.

Damit die Länge der Datenbank dynamisch ist, verwenden wir eine `for-each`-Schleife. Mittels `select="Publications/Author"` geben wir an, welche Daten in der Schleife ausgelesen werden sollen. `Publication` und `Author` entsprechen den Tags in der XML-Datei. Innerhalb dieser Schleife verwenden wir die zweite Vorlage `CLASS = "MyRow"`. Für jedes Feld, das aus der XML-Datei ausgelesen wird, ist die Anweisung `value-of select="..."` erforderlich. Innerhalb der Anführungs- und Schlusszeichen wird jeweils der gewünschte Feldname geschrieben. Auch hier muss der Feldname immer mit einem Tag in der XML-Datei übereinstimmen.

```
'=====
' Auf CD      Buchdaten\Beispiele\11_Web_Mail\
' Dateiname  11_12_Codebook.xml
'=====
```

```
<?xml version='1.0' encoding="ISO-8859-1"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    version="1.0">
  <xsl:template match="/">
    <HTML>
      <HEAD>
        <STYLE>
          .MyHeader { background-color: silver;
                      font-weight: bold; }
```



```

        .myRow    { background-color: bisque;
                    border-top: thin double red}
    </STYLE>
</HEAD>

<BODY>
  <TABLE border="1">
    <COLGROUP WIDTH="100" ALIGN="LEFT"></COLGROUP>
    <COLGROUP WIDTH="100" ALIGN="LEFT"></COLGROUP>
    <COLGROUP WIDTH="100" ALIGN="CENTER"></COLGROUP>
    <COLGROUP WIDTH="160" ALIGN="RIGHT"></COLGROUP>

    <TR>
      <TD CLASS="MyHeader">Vorname</TD>
      <TD CLASS="MyHeader">Name</TD>
      <TD CLASS="MyHeader">ISBN</TD>
      <TD CLASS="MyHeader">Titel</TD>
    </TR>

    <xsl:for-each select="Publications/Author">
      <TR>
        <TD CLASS = "myRow">
          <xsl:value-of select="FirstName"/>
        </TD>
        <TD CLASS = "myRow">
          <xsl:value-of select="LastName"/>
        </TD>
        <TD CLASS = "myRow">
          <xsl:value-of select="ISBN"/>
        </TD>
        <TD CLASS = "myRow">
          <xsl:value-of select="Title"/>
        </TD>
      </TR>
    </xsl:for-each>
  </TABLE>
</BODY>
</HTML>
</xsl:template>
</xsl:stylesheet>
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

Wenn Sie sowohl die XML-Datei als auch die XSL-Datei aufgebaut haben, dann können Sie die *XML-Datei* in Excel öffnen. Beim Öffnen der XML-Datei können Sie wählen, ob Sie das Stylesheet verwenden möchten. Es wird eine entsprechende Dialogbox angezeigt:

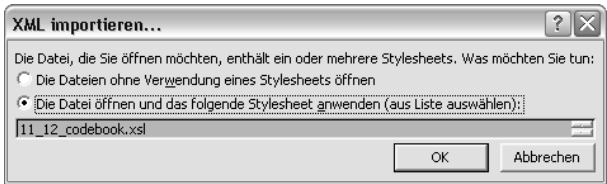


Abbildung 325: XML importieren mit Stylesheet

XML ist eine Sprache für sich, auf die wir hier nicht näher eingehen können.

Literaturvorschläge zum Thema XML:

»Jetzt lerne ich XML«, Verlag: Markt + Technik, ISBN-Nr. 3-8272-5924-X.

»XML und VBA«, Verlag: Addison-Wesley, ISBN-Nr. 3-8273-1952-8.

391 Smart Tags

Smart Tags sind erst seit der Version 2002 (XP) in Excel verfügbar. Was sind Smart Tags? Smart Tags erscheinen z.B., wenn Sie in einer Zelle einen bestimmten Begriff eingeben, der von Excel als Smart Tag erkannt wird. Das Verhalten erinnert an die Autokorrektur in Word. Auch die Funktion Copy&Paste (siehe Abbildung 326) lässt ein Smart Tag erscheinen. Smart Tag ist sozusagen der Oberbegriff für aktionssensitive Menüs.

Zu erkennen sind Smart Tags in Excel in der Regel am violetten Dreieck, das in der rechten unteren Ecke einer Zelle angezeigt wird. Zudem erscheint ein kleines Symbol (Icon). Bei Klick auf das Symbol öffnet sich ein Menü. Je nach Smart Tag werden dort verschiedene Einträge angezeigt. Dies jedoch nur, wenn die Smart-Tags-Optionen aktiviert sind. Aktivieren können Sie diese unter EXTRAS | AUTOKORREKTUR-OPTIONEN, Registerkarte SMARTTAGS.

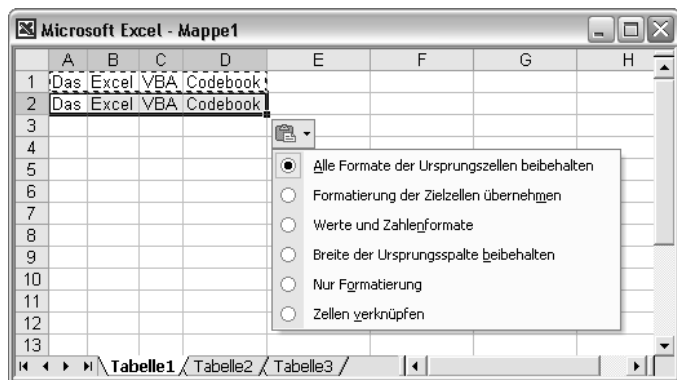


Abbildung 326: Ein Smart Tag: Copy & Paste

Smart Tags sind ActiveX-DLLs, die gewöhnlich in Programmiersprachen wie VB oder C++ programmiert werden. Einfache Smart Tags können jedoch auch per XML selbst definiert werden, wie Sie später noch erfahren werden. Falls Sie zusätzliche vordefinierte Smart Tags in Ihre Liste in der AUTOKORREKTUR hinzufügen möchten, können diese auf einer der Microsoft-Webseiten bezogen werden: <http://www.officesmarttags.com/>. Verschiedene Unternehmen/Anbieter stellen ihre Smart Tags zur Verfügung. Leider haben unsere Tests gezeigt, dass die meisten davon in der deutschsprachigen Office-Version nicht unterstützt werden.

Wenn in einer Zelle ein Smart Tag angezeigt wird, bedeutet das, dass die Mappe in einem XML-kompatiblen Format vorliegt. Ein Smart Tag muss in der Form eines URI (Uniform Resource Identifier) definiert worden sein. Diesem folgt eine Raute (#) sowie ein Name. In unserem Beispiel lautet er StockTicker.

Im ersten Teil des Codes wird sichergestellt, dass die Smart Tags aktiv sind und angezeigt werden können. In die Zelle A1 wird das Schlüsselwort eingetragen, welches das Smart Tag anzeigt. Im letzten Teil der Prozedur wird dem Eintrag in Zelle A1 der URI zugewiesen. Wir werden in unserem nächsten Rezept per XML ein solches URI verwenden. Sie werden dann auch eine nähere Beschreibung dazu erhalten.

```
=====
' Auf CD      Buchdaten\Beispiele\11_Web_Mail\
' Dateiname   11_13_SmartTags.xls
' Modul       mdl_01_CreateSmartTag
=====
```

```
Sub CreateSmartTag()  
    ' Sicherstellen, dass die SmartTags in der Mappe aktiv sind ...  
    ActiveWorkbook.SmartTagOptions.EmbedSmartTags = True  
  
    ' ... und als SmartTags beschriftet werden können  
    Application.SmartTagRecognizers.Recognize = True  
  
    ' Eintrag in Zelle A1  
    Range("A1").Value = "MSFT"  
  
    ' SmartTag hinzufügen und URI angeben  
    Range("A1").SmartTags.Add _  
        ("urn:schemas-microsoft-com:smartrtags#StockTicker")  
End Sub
```

Die folgende Prozedur gibt die Bezeichnung des Smart Tag in Form seiner XML-Deklaration zurück:

```
=====
' Auf CD      Buchdaten\Beispiele\11_Web_Mail\
' Dateiname  11_13_SmartTags.xls
' Modul      mdl_02_ShowXMLCode
=====
```

```
Sub ShowXMLCode()  
    ' SmartTag als XML-Wert ausgeben  
    MsgBox Range("A1").SmartTags.Add _  
        ("urn:schemas-microsoft-com:smctags#StockTickerSymbol").XML  
End Sub
```

392 Ein eigenes Smart-Tag-Menü erstellen (XML)

Wie bereits im vorangegangenen Rezept erwähnt, werden Smart Tags in der Regel in VB oder C++ programmiert. Der einfachste Weg, um eigene Smart Tags zu erstellen, führt jedoch über XML. Sie können in einer XML-Datei Schlüsselwörter erfassen und diese mit Hyperlinks oder lokalen Pfadangaben verknüpfen. Zum Erstellen einer XML-Datei verwenden Sie beispielsweise den Texteditor Notepad von Windows oder beliebig einen anderen Editor.

Grundlagen

Allgemein

Datu
Zeit

tungen

Steuer
elemen

Befehl leisten

Objekt

Diagramm

Ereignisse

User-Form

**Web/
Mail**

Extern

Ge-
misch
tes

Specia

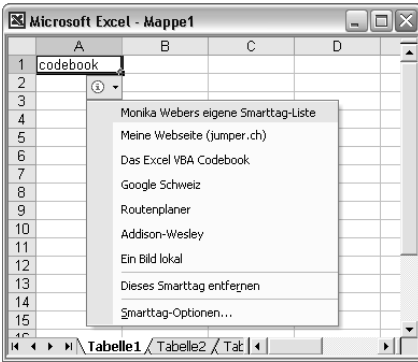


Abbildung 327: Ein selbst erstelltes Smart-Tag-Menü

1. Schließen Sie alle Ihre Office-Applikationen inkl. Internet Explorer
2. Geben Sie in Notepad den folgenden Code ein:

```
' =====
' Auf CD      Buchdaten\Beispiele\11_Web_Mail\
' Dateiname  11_12_MySmartTags.xml
' =====

<FL:smarttaglist xmlns:FL="urn:schemas-microsoft-com:smarttags:list">

  <FL:name>Autor: Monika Weber</FL:name>
  <FL:description>Eine Liste an eigenen Smart Tags</FL:description>
  <FL:moreinfourl>http://www.jumper.ch</FL:moreinfourl>
  <FL:lcid>1031, 1033</FL:lcid>
  <FL:smarttag type="urn:schemas-microsoft-com:smarttags#terms">

    <FL:caption>Monika Webers eigene Smarttag-Liste</FL:caption>

    <FL:terms>
      <FL:termlist>
        monika, weber, excel, vba, help-desk, forum, google,
        routenplaner, codebook, addison-wesley, bild
      </FL:termlist>
    </FL:terms>

    <FL:actions>
      <FL:action id="MyWebsite">
        <FL:caption>Meine Webseite (jumper.ch)</FL:caption>
        <FL:url>http://www.jumper.ch</FL:url>
      </FL:action>

      <FL:action id="MyCodebooksite">
        <FL:caption>Das Excel VBA Codebook</FL:caption>
        <FL:url>http://excel.codebooks.de</FL:url>
      </FL:action>

      <FL:action id="MyGoogle">
```

```
<FL:caption>Google Schweiz</FL:caption>
<FL:url>http://www.google.ch</FL:url>
</FL:action>

<FL:action id="MyRoute">
  <FL:caption>Routenplaner</FL:caption>
  <FL:url>http://www.map24.ch</FL:url>
</FL:action>

<FL:action id="MyAW">
  <FL:caption>Addison-Wesley</FL:caption>
  <FL:url>http://www.addison-wesley.de</FL:url>
</FL:action>

<FL:action id="MyPicture">
  <FL:caption>Ein Bild lokal</FL:caption>
  <FL:url>c:\Magier.jpg</FL:url>
</FL:action>
</FL:actions>

</FL:smarntag>
</FL:smarntaglist>
```

- 3. Speichern Sie die Datei mit der Endung *.xml im folgenden Verzeichnis ab: C:\Programme\Gemeinsame Dateien\Microsoft Shared\Smart Tag\Lists. Der Dateiname spielt dabei keine Rolle. Sie können auch die Datei 11_13_MySmartTags.xml verwenden und im angegebenen Pfad abspeichern. Die Datei befindet sich auf der Buch-CD.
- 4. Starten Sie Excel und geben Sie einen der Suchbegriffe aus der termlist in eine Zelle ein. Zum Beispiel »codebook« oder »routenplaner« (ohne Anführungs- und Schlusszeichen).
- 5. Das violette Dreieck und ein Icon erscheinen.
- 6. Klicken Sie auf das Icon, um das Menü zu öffnen, und anschließend auf einen der Menüpunkte.
- 7. Sie gelangen nun auf die gewünschte Webseite, oder die lokale Datei öffnet sich.

Erläuterungen zum XML-Code:

Es handelt sich bei den Codezeilen um eine spezielle Syntax, die durch Microsoft zur Verfügung gestellt wird. Es ist daher unerlässlich, die Zeile 1 smarntaglist genauso wie in unserem Beispiel zu hinterlegen. Hier finden Sie nun also auch wieder ein URI, wie bereits im vorangegangenen Rezept. FL dient als Alias für die weiteren Tags, um nicht jedes Mal die gesamte lange Codezeile angeben zu müssen. Die Zuweisung erfolgt über: FL="urn:schemas-microsoft-com:smart-tags:list.

Die weiteren fünf Codezeilen sind Deklarationen bzw. interne Informationen, die in der Smart-Tag-Liste nicht angezeigt werden. In der Zeile 2 name können Sie beispielsweise den Autor der XML-Datei erwähnen. In der Zeile 3 description folgt eine Beschreibung zu den Smart Tags. In der Zeile 4 moreinfourl können Sie eine Quelle angeben, wo weitere Informationen zu Smart Tags zu finden sind. In der Zeile 5 lcid können Sprachcodes verwendet werden. Diese dienen der Erkennung der Schlüsselwörter. Der Code 1031 steht für Deutsch, der Code 1033 steht für US-Englisch. Je nach Spracheinstellung findet eine Erkennung statt. Wenn die Suchbegriffe sprachunabhängig sein sollen,

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

dann können Sie an Stelle der Codes eine 0 (Null) oder ein * eingeben, oder die gesamte Codezeile weglassen. In der Zeile 6 `smarttag type` wird die Definition des Smart-Tag-Typs hinterlegt.

Nach der Deklaration folgt der eigentliche Kern des Smart-Tag-Menüs, nämlich die sichtbaren Menüeinträge. Die Zeile `caption` legt die Überschrift des Smart Tags fest.

Im nächsten Codeblock `terms` werden die Suchbegriffe erfasst, die später in die Excel-Zellen eingegeben werden können, um das Smart Tag erscheinen zu lassen.

Im letzten Codeblock `actions` werden die einzelnen Hyperlinks oder lokalen Dateipfade hinterlegt. Sie bilden die eigentlichen anklickbaren Menüpunkte. Natürlich müssen diese Hyperlinks oder Pfadangaben stimmen, ansonsten erfolgt keine Reaktion bei Anklicken des Menüpunktes. Innerhalb von jedem `action`-Block wird eine Überschrift `caption` und der Hyperlink oder die lokale Pfadangabe `url` festgelegt. Mittels `action id="..."` werden die Felder eindeutig identifiziert, indem ihnen ein Name zugewiesen wird. Ansonsten könnte XML die einzelnen Menüpunkte nicht voneinander unterscheiden.

Smart Tags, wie wir sie hier erzeugt haben, sind in allen Office-Applikationen verfügbar, die diese Art von Menüs unterstützen.

Die Variante, die hier beschrieben wurde, reicht zwar aus, um einfache Smart Tags zu programmieren, wenn Sie jedoch aufwendigere Formen erzeugen möchten, dann kommen Sie nicht darum herum, mit einem Entwicklungssystem für COM-Objekte (Component Object Model) zu arbeiten, wie dies bei VB oder C++ der Fall ist. Mit solchen Programmiersprachen können eigene DLLs (Dynamic Link Libraries) erzeugt werden. VBA ist dazu nicht in der Lage.

393 Smart Tags löschen

Wenn Smart Tags in der Autokorrektur aktiviert sind, erscheinen bei jedem übereinstimmenden Suchbegriff die violetten Dreiecke in der linken unteren Ecke der Zelle. Nicht in jedem Dokument ist dies erwünscht. Um ein Smart Tag manuell zu löschen, klicken Sie auf das Smart Tag-Icon und treffen die Auswahl DIESES SMARTTAG ENTFERNEN. Diese Option ist standardmäßig vorhanden und muss nicht programmiert werden. Wenn viele Smart Tags in einem Tabellenblatt enthalten sind, ist dieses Vorgehen ziemlich aufwendig.

Um nur einen Smart Tag zu löschen, können Sie diesen über den Index ansprechen. Für den Fall, dass kein Smart Tag im Tabellenblatt vorhanden ist, nehmen wir eine entsprechende Prüfung vor.

```
' =====
' Auf CD      Buchdaten\Beispiele\11_Web_Mail\
' Dateiname  11_14_SmartTagsDelete.xls
' Modul      mdl_01_DeleteOneSmartTag
' =====
```

```
Sub DeleteOneSmartTag()
    With ActiveSheet
        If .SmartTags.Count > 0 Then
            .SmartTags(1).Delete
        Else
            MsgBox "Es sind keine SmartTags vorhanden."
        End If
    End With
End Sub
```

Um alle Smart Tags des aktiven Tabellenblattes zu löschen, verwenden Sie eine For-Schleife:

```
'=====
' Auf CD      Buchdaten\Beispiele\11_Web_Mail\
' Dateiname   11_14_SmartTagsDelete.xls
' Modul       mdl_02_DeleteSmartTagsSheet
'=====

Sub DeleteSmartTagsSheet()
    Dim st As SmartTag

    For Each st In ActiveSheet.SmartTags
        st.Delete
    Next st
End Sub
```

Um sämtliche Smart Tags der Mappe zu entfernen, verwenden Sie zwei verschachtelte For-Schleifen:

```
'=====
' Auf CD      Buchdaten\Beispiele\11_Web_Mail\
' Dateiname   11_14_SmartTagsDelete.xls
' Modul       mdl_03_DeleteSmartTagsWorkbook
'=====

Sub DeleteSmartTagsWorkbook()
    Dim ws As Worksheet
    Dim st As SmartTag

    For Each ws In Worksheets
        For Each st In ws.SmartTags
            st.Delete
        Next st
    Next ws
End Sub
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

Extern

In der Kategorie »Extern« finden Sie eine umfangreiche Sammlung an Beispielen, wie Excel mit anderen Microsoft-Anwendungen kommunizieren kann. Es geht dabei unter anderem darum, wie Informationen aus anderen Anwendungen importiert werden können oder umgekehrt. Dies ist ein sehr umfangreiches Gebiet. Es ist wohl kaum möglich, auf ein paar Buchseiten sämtliche Bedürfnisse abzudecken. Wir möchten Ihnen jedoch einige Muster aufzeigen und Sie damit auf den richtigen Weg führen. Wie so oft entspringt ja, basierend auf anderen Vorschlägen, eine eigene Idee.

Achtung

Damit Sie sich das Erstellen von Beispielen ersparen können, haben wir im Verzeichnis <CD>:\Buchdaten\Beispiele\12_Extern einen Ordner Namens *Codebook_12_Test* erstellt. Damit Sie die Codes testen können, müssen Sie den gesamten Ordner auf Ihr Laufwerk C:\ kopieren, denn die meisten der Beispiele sprechen direkt dieses Laufwerk, in Verbindung mit dem Ordner *Codebook_12_Test*, an. Alle Codes, welche sich in Textdateien *.txt befinden, können Sie in Excel im VBA-Editor in ein Modul kopieren und dann mit der Taste F5 ausführen.

Early binding und Late binding

Bevor wir uns nun den einzelnen Rezepten zuwenden, möchten wir Ihnen etwas an genereller Theorie in Bezug auf die Zusammenarbeit mit externen Office-Anwendungen vermitteln. Sobald Sie auf eine andere Anwendung zugreifen, müssen Sie im VBA-Editor im Menü EXTRAS | VERWEISE einen Verweis auf die Bibliothek dieser Anwendung setzen. Dies ist erforderlich, da andere Office-Anwendungen zum Teil mit anderen Objekten, Methoden und Eigenschaften arbeiten. Es gibt zwei verschiedene Wege, um die Bibliothek von anderen Anwendungen zu verwenden. Der eine Weg nennt sich »early binding« (frühe Bindung) und der andere »late binding« (späte Bindung). In unseren Rezepten werden wir wahlweise den einen oder anderen Weg wählen. Worin besteht der Unterschied?

In »early binding« wird ein Objekt zu einer anderen Office-Anwendung erzeugt, noch bevor die Verbindung zur Bibliothek dieser Anwendung besteht. In früheren Versionen war es sogar erforderlich, sich dieses Verfahrens zu bedienen, und es wird auch heute noch verwendet, denn es hat gewisse Vorteile gegenüber dem »late binding«. Ein Vorteil besteht darin, dass Code geschrieben werden kann, der die Verfügbarkeit der erforderlichen Bibliothek eigenständig ermittelt, egal auf welcher Version die Prozedur ausgeführt wird. Mittels »late binding« wird die Verbindung zur Bibliothek erst zur Laufzeit hergestellt. Der Nachteil von »late binding« besteht darin, dass zum Zeitpunkt des Erstellens von Code keine Verbindung zur Zielanwendung besteht. Der VBA-Editor bietet somit keine Hilfe an.

In »early binding« wird der Verweis zu einer Bibliothek aktiviert (EXTRAS | VERWEISE). Es besteht somit eine Verbindung zu dieser Bibliothek, noch bevor der Code ausgeführt wird. Seit der Office-Version 97 wird dieses Verfahren durch alle Office-Anwendungen unterstützt. Code, der sich der frühen Bindung bedient, ist schneller. Es ist einfacher, einen Code mit »early binding« zu schreiben, denn bei korrekter Variablen-Deklaration und Referenzierung erscheinen automatisch Vorschläge zu den Objekten, Eigenschaften und Methoden der Zielanwendung (IntelliSense).

Grundlagen
Allgemein
Datum und Zeit
Formeln
Diagramme
Steuerungselemente
Befehle
Objekte
Diagramme
Ereignisse
User-Formen
Web/Mail
Extern
Gemischtes
Spezial

Verwandschaft von VBA zu DOS

Für all jene, die bereits zu DOS-Zeiten aktiv waren und DOS-Anweisungen kennen, werden einige der Befehle, die auf den folgenden Seiten beschrieben sind, wenig vertraut sein, denn sie sind zum Teil den VBA-Anweisungen sehr ähnlich. So wird zum Beispiel der DOS-Befehl `CD`, der zum Wechseln eines Verzeichnisses dient, in VBA als `ChDir` geschrieben. Eine gewisse Verwandschaft scheint somit zu bestehen.

DOS heißt ausgeschrieben *Disk Operating System*. Es handelt sich dabei um das legendäre erste Betriebssystem von Bill Gates und somit von Microsoft. DOS wird immer mehr durch modernere Betriebssysteme wie z.B. Windows verdrängt.

Auf den nächsten Seiten geht es jedoch nicht um DOS, sondern vielmehr um den Windows-Explorer, das Inhaltsverzeichnis Ihrer Festplatte. Es können Laufwerke, Ordner und Dateien angezeigt werden. Natürlich ist das längst nicht alles. Dateien können kopiert, umbenannt, von Ordner zu Ordner verschoben werden und vieles mehr.

Nun, wozu denn den Explorer mit VBA steuern? Versuchen Sie einmal, alle Datei-Namen eines Ordners schön übersichtlich in einer Datei aufzulisten. Markieren im Explorer geht. Auch mehrere Dateien lassen sich gleichzeitig auswählen. Kopieren scheint auch anzusprechen. Wenn Sie nun jedoch versuchen, den Text nach Excel zu kopieren, dann ist Schluss, denn die Kopierfunktion ist nicht dazu gedacht, Dateinamen auszulesen, sondern die Dateien selbst und nicht etwa deren Namen anzusprechen. Nun drängt sich die Frage auf, wie man dennoch die Dateinamen auslesen könnte. Genau das wird eins der Themen sein, die auf den folgenden Seiten beschrieben sind.

Dabei gibt es noch weit mehr Möglichkeiten, als nur die Dateinamen auszulesen. Natürlich ließen sich auch unschöne Dinge wie Makro-Viren programmieren. Aber wir raten Ihnen ganz dringend davon ab, denn auch wenn Sie »nur« Ihrem Arbeitskollegen einen Streich spielen wollen: Es ist rechtswidrig und Sie können damit in Konflikt mit dem Gesetz und der Justiz geraten. Vergessen Sie es also und denken Sie nicht im Entferntesten daran, jemandem per VBA Schaden zuzufügen.

394 Dateien löschen

In unserem ersten Beispiel wird eine Datei aus einem Unterordner gelöscht. Wichtig dabei ist, dass die korrekte Angabe des Pfades und des Dateinamens erfolgt. Sollte eine der Angaben nicht stimmen, wird eine entsprechende Fehlermeldung ausgegeben.

Um eine Datei zu löschen, verwenden Sie die Methode `Kill`. Ob es sich dabei um eine Excel-Datei oder um einen anderen Dateityp handelt, spielt keine Rolle.

Achtung

Es gibt keine Rückfrage, ob Sie die Datei wirklich löschen möchten.

```
'=====
' Auf CD      Buchdaten\Beispiele\12_Extern\
' Dateiname   12_01_Explorer.xls
' Modul       mdl_01_FileDelete
'=====

Sub FileDelete()
    Kill ("C:\Codebook_12_Test\MoreTests\MyFile.txt")
End Sub
```

Wenn Sie aus einem Ordner sämtliche Dateien eines bestimmten Typs, wie z.B. *.txt löschen möchten, können Sie mit Platzhaltern wie * oder ? arbeiten. Der Stern steht dabei stellvertretend für eine restliche Zeichenfolge und das Fragezeichen steht für ein einzelnes Zeichen.

Nachfolgend ein paar Beispiele:

Rechnung*.xls

Es werden alle Dateien gelöscht, welche mit *Rechnung* beginnen und die Endung *xls* tragen. Es könnte sich dabei um eine Vielzahl von Rechnungen handeln, die jeweils am Ende des Dateinamens noch eine Nummerierung haben.

*.xls

Es werden alle Dateien mit der Endung *xls* gelöscht.

Rechnung_?00.xls

Hier werden alle Rechnungen gelöscht, die eine runde Hunderter-Nummer tragen. So werden beispielsweise die Rechnungen 100, 200 und 300 gelöscht, wobei die Rechnungen mit der Nummer 101, 235, 377 usw. erhalten bleiben.

Im nachfolgenden Beispiel werden aus dem angegebenen Unterordner sämtliche Beispiele mit der Endung *.txt gelöscht.

```
=====
' Auf CD      Buchdaten\Beispiele\12_Extern\
' Dateiname   12_01_Explorer.xls
' Modul       mdl_01_FileDelete
=====
```

```
Sub txtFilesDelete()  
    Kill ("C:\Codebook_12_Test\MoreTests\*.txt")  
End Sub
```

Wenn der Inhalt des Ordners komplett gelöscht werden soll, dann lassen Sie die Datei-Endung weg und geben stattdessen noch ein Sternchen (*) ein.

```

=====
' Auf CD      Buchdaten\Beispiele\12_Extern\
' Dateiname  12_01_Explorer.xls
' Modul      mdl_01_FileDelete
=====

```

```
Sub DeleteAllFiles()  
    Kill ("C:\Codebook_12_Test\MoreTests\*.*")  
End Sub
```

395 Dateien umbenennen

Wenn Dateien aus Excel und VBA heraus umbenannt werden sollen, ist das recht einfach. Die Anweisung dazu lautet `Name...As....`. Etwas umständlicher ist die Fehlerbehandlung, die in unserem Beispiel in Form einer `If`-Entscheidung erfolgt. In der äußeren `If`-Entscheidung wird geprüft, ob überhaupt eine Datei mit dem Namen *MyRenameFile.xls* im Verzeichnis *C:\Codebook_12_Test* vorhanden ist. Wenn nicht, wird eine Meldung ausgegeben. In der inneren `If`-Entscheidung wird geprüft, ob bereits eine Datei mit dem Namen *NewRenameFile.xls* im glei-

chen Verzeichnis vorhanden ist. Bei Zutreffen wird eine entsprechende Meldung ausgegeben. Ansonsten wird die Datei *MyRenameFile.xls* umbenannt in *NewRenameFile.xls*.

```
'=====
' Auf CD      Buchdaten\Beispiele\12_Extern\
' Dateiname   12_01_Explorer.xls
' Modul       mdl_02_Rename
'=====

Sub RenameFile()
    Dim strPath As String
    Dim strFileOld As String
    Dim strFileNew As String

    strPath = "C:\Codebook_12_Test\"
    strFileOld = "MyRenameFile.xls"
    strFileNew = "NewRenameFile.xls"

    If Dir(strPath & strFileOld) = "" Then
        MsgBox "Es ist keine Datei mit dem Namen " _
            & strFileOld & " vorhanden."
        Exit Sub
    Else
        If Dir(strPath & strFileNew) <> "" Then
            MsgBox "Eine Datei mit dem Namen " _
                & strFileNew & " ist bereits vorhanden."
        Else
            ' Hier findet die Umbenennung statt
            Name (strPath & strFileOld) As (strPath & strFileNew)
        End If
    End If
End Sub
```

Wenn mehrere Dateien umbenannt werden sollen, so ist das nicht immer ganz einfach, denn dies sollte nach Möglichkeit in einem Durchlauf geschehen. Wenn es sich um Nummerierungen handelt, kann mit einer Schleife gearbeitet werden.

In unserem Beispiel werden Rechnungsnummern von zweistellig auf dreistellig umgestellt, also von *Invoice_01.xls* auf *Invoice_001.xls*.

```
'=====
' Auf CD      Buchdaten\Beispiele\12_Extern\
' Dateiname   12_01_Explorer.xls
' Modul       mdl_02_Rename
'=====

Sub RenameFiles1()
    Dim strPath As String
    Dim strFile As String
    Dim i As Integer

    strPath = "C:\Codebook_12_Test\Invoice1\"
```

```
For i = 1 To 9
    strFile = "Invoice_0" & i & ".xls"
    If Dir(strPath & strFile) <> "" Then
        Name (strPath & strFile) As _
            (strPath & "Invoice_00" & i & ".xls")
    End If
Next i
End Sub
```

Wenn Sie Nummern von zweistellig auf dreistellig umstellen möchten, ist bereits eine zweite Schleife erforderlich. Die Erste für Rechnungen von 01 bis 09, eine Zweite für Rechnungen von 10 bis 99.

```
'=====
' Auf CD      Buchdaten\Beispiele\12_Extern\
' Dateiname   12_01_Explorer.xls
' Modul       mdl_02_Rename
'=====

Sub RenameFiles2()
    Dim strPath As String
    Dim strFile As String
    Dim i As Integer

    strPath = "C:\Codebook_12_Test\Invoice2\"

    For i = 1 To 9
        strFile = "Invoice_0" & i & ".xls"
        If Dir(strPath & strFile) <> "" Then
            Name (strPath & strFile) As _
                (strPath & "Invoice_00" & i & ".xls")
        End If
    Next i

    For i = 10 To 99
        strFile = "Invoice_" & i & ".xls"
        If Dir(strPath & strFile) <> "" Then
            Name (strPath & strFile) As _
                (strPath & "Invoice_0" & i & ".xls")
        End If
    Next i
End Sub
```

Der Abbildung 328, Bild links, können Sie entnehmen, wie die Dateinamen vor der Neuenummerierung ausgesehen haben. Auf dem Bild links können Sie erkennen, dass die Nummern vor dem Ausführen des Codes zweistellig sind. Auf dem Bild links sind die Nummern nach dem Ausführen des Codes dreistellig.

Im Zusammenhang mit Dateien haben Sie nun ein paar VBA-Anweisungen erfahren. Da es noch mehr solcher Befehle gibt, finden Sie an dieser Stelle eine Zusammenfassung. Auf einige der hier aufgeführten Befehle werden wir auf den nächsten Seiten noch zu sprechen kommen.

Grundlagen
Allgemein
Datum und Zeit
Formeln
Diagramme
Objekte
Benutzerdefinierte Funktionen
Web/Mail
Extern
Gemischtes
Spezial

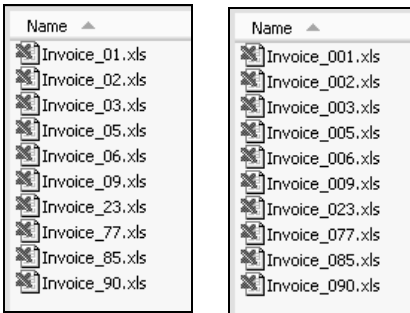


Abbildung 328: Ansicht vor und nach dem Umbenennen der Dateien

VBA	Erläuterung
CurDir (Laufwerk)	Aktueller Pfad
Environ ({envstring number})	Systeminformationen
ChDir (Pfadname)	Verzeichnis wechseln
ChDrive (Laufwerk)	Laufwerk wechseln
MkDir (Pfadname)	Verzeichnis erstellen
RmDir (Pfadname)	Verzeichnis löschen
Name (Datei_x) As (Datei_y)	Datei umbenennen
FileCopy (Quelle), (Ziel)	Datei kopieren
Kill (Pfadname)	Datei löschen
Dir (Pfadname[, Attribute])	Datei oder Verzeichnis auflisten
FileLen (Pfadname)	Größe der Datei in Byte
FileDateTime (Pfadname)	Datum und Zeit der letzten Änderung
GetAttr (Pfadname)	Attribute einer Datei oder eines Verzeichnisses
SetAttr (Pfadname[, Attribute])	Attribute vergeben oder verändern

Tabelle 84: VBA-Befehle für Explorer

396 Mit ActiveX ein Objekt erzeugen

In diesem Rezept wollen wir ein ActiveX-Objekt erzeugen. Sie werden hier keinen kompletten VBA-Code finden, sondern werden erst einmal mit den ActiveX-Komponenten vertraut gemacht. Anstelle vieler VBA-Anweisungen lassen sich ActiveX-Komponenten einsetzen. Die Möglichkeiten sind dabei sehr umfangreich. Mittels ActiveX ist es möglich, andere Programme aus Excel heraus anzusprechen und zu steuern. Das bedeutet, dass unter Verwendung von ActiveX nicht mehr nur die VBA-Befehle aus den Excel-Bibliotheken genutzt werden können, sondern auch die Befehle des Programms, das gesteuert wird. Das kann beispielsweise der Explorer sein, oder aber auch Anwendungen wie Word, Access, PowerPoint usw. Oftmals ist es dabei erforderlich, dass ein Verweis auf die Bibliothek des »Fremdprogrammes« gesetzt wird, damit auch deren Eigenschaften, Methoden und Objekte zur Verfügung stehen. Für die folgenden Beispiele ist dies jedoch noch nicht erforderlich. Sobald ein weiterer Verweis gesetzt werden muss, werden wir Sie darauf hinweisen.

Mittels ActiveX wollen wir ein Objekt erzeugen. Damit Sie sich eine Vorstellung davon machen können, wieso das geschieht, müssen Sie die Syntax kennen. Diese lautet:

```
CreateObject(Klasse,[Servername])
```

Damit die Syntax etwas verständlicher wird, finden Sie nachfolgend zwei Beispiele, die zeigen, wie Anweisungen in der Praxis aussehen können. Das erste Beispiel wird verwendet, um die Verbindung mit der Word-Anwendung aufzunehmen. Das zweite Beispiel werden wir auf den nächsten Seiten immer wieder einsetzen. Es wird verwendet, um mit Windows und dem Explorer zu kommunizieren.

```
CreateObject("Word.Application")  
  
CreateObject("Scripting.FileSystemObject")
```

Natürlich kann auch für Excel selbst ein Objekt erzeugt werden. Zum Beispiel ein Blatt. Dadurch wird deutlich, dass eigentlich auch in Excel selbst jede Bibliothek mittels ActiveX gesteuert wird.

```
CreateObject("Excel.Sheet")
```

Mittels `CreateObject` wird, wie es der Name schon sagt, ein Objekt erzeugt. Dabei muss eine Klasse angegeben werden. Optional kann zusätzlich ein Servername angegeben werden. Diese beiden Angaben erfolgen innerhalb der runden Klammern, zwischen Anführungs- und Schlusszeichen. Falls keine Angabe zum Server erfolgt, wird der lokale Computer verwendet.

Wie eine Anweisung unter Angabe des Servers aussehen könnte, können Sie der nachfolgenden Codezeile entnehmen. Sie können damit ein Objekt auf einem anderen PC im Netzwerk erstellen. Als Name wird ein Teil des Namens verwendet, der auch bei Freigaben eingesetzt wird (`\\MyServer\Public`), wobei in diesem Falle `MyServer` der Teil ist, der in den Code integriert wird. Es wird dabei also zusätzlich zur Klasse der Name des Servers eingegeben. Wenn ein Servername verwendet wird, der nicht existiert, wird eine Fehlermeldung ausgegeben.

```
CreateObject("Excel.Application", "MyServer")
```

Es gibt noch mehr Anwendungsgebiete für ActiveX. Diese werden jeweils an der entsprechenden Stelle auf den folgenden Buchseiten beschrieben.

397 Dateien verschieben

Die Anweisung, um Dateien zu verschieben, lautet: `MoveFile`. Im nachfolgenden Beispiel wird die Datei *MyFile2.xls* in einen Unterordner verschoben. Dies geschieht innerhalb einer Codezeile (welche hier über drei Buchzeilen umbrochen werden muss). In der `If`-Entscheidung wird geprüft, ob im Quellordner eine Datei mit dem Namen *MyFile2.txt* vorhanden ist. Es wird zudem geprüft, ob im Zielordner noch keine Datei mit diesem Namen vorhanden ist. Wenn beide Prüfungen positiv ausfallen, wird die Prozedur fortgesetzt. Eine entsprechende Meldung `MsgBox` wird auf dem Bildschirm angezeigt.

Hinweis

Falls auf Ihrem System ein Virens Scanner installiert ist, kann durch die Methode `MoveFile` eine Warnung ausgelöst werden.

Grundlagen

Allgemein

Datum
Zeit

tungen

Steuer
elemente

Befehl
leisten

Objekt

Diagramm

Ereignisse

User-
Form

Web/
Mail

Extern

Gemischtes

Spezial

```
'=====
' Auf CD      Buchdaten\Beispiele\12_Extern\
' Dateiname  12_01_Explorer.xls
' Modul      mdl_03_Move
'=====

Sub MoveFile()
    Dim strPathOld As String
    Dim strPathNew As String

    strPathOld = "C:\Codebook_12_Test\MyFile2.txt"
    strPathNew = "C:\Codebook_12_Test\MoreTests\"

    If Dir(strPathOld) <> "" And _
        Dir(strPathNew & "MyFile2.txt") = "" Then
        CreateObject("Scripting.FileSystemObject").MoveFile _
            strPathOld, strPathNew
        MsgBox "Die Datei wurde verschoben."
    Else
        MsgBox "Die Datei konnte nicht verschoben werden."
    End If
End Sub
```

Um einen gesamten Ordner zu verschieben, verwenden wir die Methode `MoveFolder`.

Um den gesamten Inhalt eines Ordners in einen anderen Ordner zu verschieben, werden wir mit Platzhaltern arbeiten (*.*), die es ermöglichen, alle Dateien in dem Ordner anzusprechen. Zu Beginn der Prozedur wird dimensioniert und referenziert.

Zuerst wird die Variable `objFSO` deklariert. Danach folgt die Referenzierung. Unter Zuhilfenahme des Befehles `Set` wird der Variablen `objFSO` die ActiveX-Komponente zugewiesen. Danach werden sämtliche Dateien vom Ordner `C:\Codebook_12_Test\MoveTestOld` in den Ordner `C:\Codebook_12_Test\MoveTestNew` verschoben.

```
'=====
' Auf CD      Buchdaten\Beispiele\12_Extern\
' Dateiname  12_01_Explorer.xls
' Modul      mdl_03_Move
'=====

Sub MoveFiles()
    Dim strPathOld As String
    Dim strPathNew As String
    Dim objFSO As Object

    strPathOld = "C:\Codebook_12_Test\MoveTestOld\*.*"
    strPathNew = "C:\Codebook_12_Test\MoveTestNew\"

    Set objFSO = CreateObject("Scripting.FileSystemObject")

    If Dir(strPathOld) <> "" And _
        Dir(strPathNew & "MyFile2.txt") = "" Then
```

```
objFS0.MoveFile strPathOld, strPathNew
MsgBox "Die Dateien wurden verschoben."
Else
MsgBox "Die Dateien konnten nicht verschoben werden."
End If

Set objFS0 = Nothing
End Sub
```

Grund-
lagen

Allge-
mein

Datu
Zeit

398 Dateien kopieren

Das Kopieren der Daten verhält sich ähnlich wie das Verschieben. Sie können wahlweise eine oder mehrere Dateien kopieren. Auch hier können Sie mit Platzhaltern wie * oder ? arbeiten. Die Anweisung für das Kopieren von Dateien lautet `CopyFile`.

Während Sie eine Datei kopieren, haben Sie zugleich die Möglichkeit, dieser einen neuen Namen zuzuweisen. Dies ist vor allem dann ein Muss, wenn die Datei im selben Ordner kopiert werden soll. Zwei Dateien mit demselben Namen und im selben Ordner sind nicht möglich. Ein anderer Name für die Kopie ist somit erforderlich. Wenn VBA auch sonst immer einen Fehler ausgibt, wenn etwas nicht funktioniert: In diesem Falle nicht.

In unserem nächsten Beispiel wird innerhalb ein und desselben Ordners eine Kopie der Datei *MyCopyFile.txt* erzeugt. Der Name der Kopie soll *MyNewCopyFile.txt* lauten.

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

```
'=====
' Auf CD      Buchdaten\Beispiele\12_Extern\
' Dateiname   12_01_Explorer.xls
' Modul       mdl_04_Copy
'=====

Sub CopyFile()
Dim objFS0 As Object
Set objFS0 = CreateObject("Scripting.FileSystemObject")

objFS0.CopyFile "C:\Codebook_12_Test\MyCopyFile.txt", _
               "C:\Codebook_12_Test\MyNewCopyFile.txt"

Set objFS0 = Nothing
End Sub
```

Im nächsten Beispiel sollen sämtliche Dateien des Ordners *Codebook_12_Test* in den Unterordner *MoreTests* kopiert werden. Da alle Dateien kopiert werden müssen, können Sie zweimal das Sternchen (*) als Platzhalter verwenden. Zum einen für alle Dateinamen und zum anderen für alle Dateierweiterungen: *.*. Beim Ziel reicht es, den Pfad anzugeben, die Platzhalter können Sie wahlweise weglassen.

```
'=====
' Auf CD      Buchdaten\Beispiele\12_Extern\
' Dateiname   12_01_Explorer.xls
' Modul       mdl_04_Copy
'=====
```

```

Sub CopyFileDir()
    Dim objFSO As Object
    Set objFSO = CreateObject("Scripting.FileSystemObject")

    objFSO.CopyFile "C:\Codebook_12_Test\*.*", _
        "C:\Codebook_12_Test\MoreTests\"

    Set objFSO = Nothing
End Sub

```

399 Datei-Eigenschaften auslesen

Datei-Eigenschaften sind jene Informationen, die Sie anzeigen können, wenn Sie im Windows-Explorer mit der rechten Maustaste auf eine Datei klicken und dann den Kontext-Menüpunkt EIGENSCHAFTEN auswählen. Dem Dialogfenster, das daraufhin geöffnet wird, sind alle Eigenschaften der Datei zu entnehmen. Unser Ziel ist es, dieses Fenster per VBA mehr oder weniger nachzubilden und die Informationen in einem Meldungsfenster `MsgBox` auszugeben. Wir werden uns bei dieser Gelegenheit weitere VBA-Eigenschaften ansehen.



Abbildung 329: Zugriffsinformationen, im Bild gekennzeichnet

Auch hier arbeiten wir mit einer ActiveX-Komponente. Wir erzeugen in einem ersten `Set` ein Objekt, mit dem wir auf den Explorer zugreifen, um die gewünschten Informationen zu erhalten. Ein zweites `Set` verwenden wir, um zusätzlich zum erstellten Objekt auf den Pfad und Dateinamen zu referenzieren. Dazu verwenden wir die Anweisung `GetFile`, gefolgt von der Angabe des Pfades und der Datei. Diese Angaben werden der Variablen `objFile` übergeben.

Nach und nach werden wir der Variablen `strMsg` die gewünschten Informationen übergeben, um sie am Ende der Prozedur vollständig als Meldungsfenster auf dem Bildschirm anzuzeigen.

Zuerst übergeben wir der Variablen `strMsg` den Dateinamen `objFile.Name` und das Laufwerk `objFile.Drive`. Diese beiden Angaben sollen in Großbuchstaben ausgegeben werden (`UCase`), damit sie sich deutlich vom restlichen Text abheben. Danach wird ein Zeilenumbruch `vbCrLf` sowie eine gestrichelte Linie eingefügt.

Als Nächstes wird der Dateityp `objFile.Type` übergeben. In unserem Beispiel ist es ein Textdokument. Es handelt sich dabei um eine Datei mit der Endung `*.txt`.

In den nächsten Codezeilen wird zuerst nur der Pfad ausgegeben `objFile.ParentFolder` und in einer zweiten Zeile der vollständige Pfad mit dem Dateinamen `objFile.Path`.

Dateiname und Pfad können auch in der 8.3-Konvention ausgegeben werden. Das bedeutet: nach alter DOS-Art, wo die maximale Größe eines Pfad- oder Dateinamens noch auf acht Zeichen für den Dateinamen und drei Zeichen für die Dateierweiterung beschränkt waren. Sie können wahlweise nur den Dateinamen `objFile.ShortName` oder den Dateinamen plus Pfad `objFile.ShortPath` ausgeben. Da diese Angaben standardmäßig in Großbuchstaben angezeigt werden, haben wir sie hier im Code in Kleinbuchstaben umgewandelt (`LCase`).

In einer weiteren Zeile in der `MsgBox` soll die Größe der Datei in Byte ausgegeben werden (`objFile.Size`).

Hier wird zuerst das Datum angezeigt, welches darüber informiert, wann die Datei erstellt worden ist (`objFile.CreateDate`). Danach wird angegeben, wann der letzte Zugriff erfolgte, also wann die Datei das letzte Mal geöffnet wurde (`objFile.DateLastAccessed`). Schließlich wird noch das Datum ermittelt, wann die Datei das letzte Mal bearbeitet wurde (`objFile.DateLastModified`).

Den Dateien können verschiedene Attribute `Attributes` mitgeliefert werden, z.B. dass die Datei mit einem Schreibschutz belegt werden soll, oder dass die Datei versteckt werden soll. Der Tabelle 85 können Sie die verschiedenen Werte und deren Beschreibung entnehmen. Die Werte lassen sich zusammenzählen. Wenn nun eine Datei als schreibgeschützt (1) und Archiv (32) hinterlegt wurde, dann werden diese Werte addiert als 33 ausgegeben.

Konstante	Wert	Beschreibung
Normal	0	Normale Datei. Ohne Attribute.
ReadOnly	1	Schreibgeschützt. Attribut mit Lese- und Schreibzugriff.
Hidden	2	Versteckte Datei. Attribut mit Lese- und Schreibzugriff.
System	4	Systemdatei. Attribut mit Lese- und Schreibzugriff.
Volume	8	Datenträgerbezeichnung des Laufwerks. Attribut mit Schreibschutz.
Directory	16	Ordner oder Verzeichnis. Das Attribut ist schreibgeschützt.
Archive	32	Die Datei hat sich seit der letzten Sicherung geändert. Attribut mit Lese-/Schreibzugriff.
Alias	64	Verknüpfung. Attribut mit Schreibschutz.
Compressed	128	Komprimierte Datei. Attribut mit Schreibschutz.

Tabelle 85: Dateiattribute

Wie der Inhalt des Nachrichtenfensters `MsgBox` aussieht, das nach dem Ausführen des Codes auf dem Bildschirm erscheint, können Sie der Abbildung 330 entnehmen.



Abbildung 330: Dateiinformationen

Der komplette Code sieht wie folgt aus:

```
' =====
' Auf CD      Buchdaten\Beispiele\12_Extern\
' Dateiname   12_01_Explorer.xls
' Modul       mdl_05_FileInfo
' =====
```

Sub FileInfo()

```
Dim objFSO As Object
Dim objFile As Object
Dim strMsg As String
```

```
Set objFSO = CreateObject("Scripting.FileSystemObject")
Set objFile = _
    objFSO.GetFile("C:\Codebook_12_Test\TextFile.txt")
```

```
' Dateiname und Laufwerk
strMsg = "Datei: " & _
    UCase(objFile.Name) & _
    " - Laufwerk: " & _
    UCase(objFile.Drive) & vbCrLf
strMsg = strMsg & "-----" & vbCrLf
```

```
' Dateityp
strMsg = strMsg & _
    "Dateityp: " & vbCrLf & _
    objFile.Type & vbCrLf
strMsg = strMsg & "-----" & vbCrLf
```

```
' Pfad, Pfad und Dateiname
strMsg = strMsg & _
    "Pfad: " & vbCrLf & vbCrLf & _
    objFile.Parentfolder & vbCrLf
strMsg = strMsg & _
    "Pfad und Datei: " & vbCrLf & _
```

```
objFile.Path & vbCrLf & vbCrLf

' Dateiname, Dateiname und Pfad in 8.3-Konvention
strMsg = strMsg & _
    "Dateiname 8.3: " & vbCrLf & _
    LCase(objFile.ShortName) & vbCrLf
strMsg = strMsg & _
    "Pfad/Datei 8.3: " & vbCrLf & _
    LCase(objFile.ShortPath) & vbCrLf & vbCrLf

' Dateigröße in Byte
strMsg = strMsg & _
    "Größe: " & vbCrLf & vbCrLf & _
    objFile.Size & _
    " Byte" & vbCrLf
strMsg = strMsg & "-----" & vbCrLf

' Zugriffsinformationen
strMsg = strMsg & _
    "Erstellt am: " & vbCrLf & _
    objFile.DateCreated & vbCrLf
strMsg = strMsg & _
    "Geändert am: " & vbCrLf & _
    objFile.DateLastModified & vbCrLf
strMsg = strMsg & _
    "Letzter Zugriff: " & vbCrLf & _
    objFile.DateLastAccessed & vbCrLf
strMsg = strMsg & "-----" & vbCrLf

' Datei-Attribute
strMsg = strMsg & _
    "Attribute: " & vbCrLf & _
    objFile.Attributes & vbCrLf
strMsg = strMsg & "-----" & vbCrLf

MsgBox strMsg

Set objFSO = Nothing
Set objFile = Nothing
End Sub
```

Grundlagen
Allgemein
Datum und Zeit
Diagramme
Steuerungselemente
Befehlslisten
Objekte
Diagramme
Ereignisse
User-Formen
Web/Mail
Extern
Gemischtes
Spezial

400 Dateiattribute setzen

Im vorangegangenen Beispiel wurden die Dateiattribute ausgelesen. Sie haben zudem die Möglichkeit, die Attribute für eine Datei oder einen Ordner per VBA zu setzen. Verwenden Sie dazu die Anweisung `SetAttr`, gefolgt vom Pfad sowie der gewünschten Konstanten. Sowohl die Konstanten als auch die Werte, die Ihnen zur Verfügung stehen, können Sie der Tabelle 86 entnehmen. Die Konstanten können kombiniert werden. Die Werte lassen sich addieren. Wenn Sie eine Datei sowohl als schreibgeschützt als auch als versteckt definieren möchten, können Sie die Werte zusammenzählen. Der Wert wäre demnach eine 3.

Konstante	Wert	Bedeutung
vbNormal	0	Normal (Voreinstellung für Dir und SetAttr)
vbReadOnly	1	Schreibgeschützt
vbHidden	2	Versteckt
vbSystem	4	Systemdatei
vbVolume	8	Datenträgerbezeichnung
vbDirectory	16	Verzeichnis oder Ordner
vbArchive	32	Datei seit letzter Sicherung verändert
vbAlias	64	Auf einem Macintosh ist der Bezeichner ein Alias

Tabelle 86: Dateiattribute zum Setzen

Im folgenden Code wird einer Datei ein Schreibschutz zugewiesen und sie wird zudem versteckt. Wir verwenden dazu den Wert 3.

```
'=====
' Auf CD      Buchdaten\Beispiele\12_Extern\
' Dateiname   12_01_Explorer.xls
' Modul       mdl_06_Attributes
'=====

Sub Attributes1()
    SetAttr "C:\Codebook_12_Test\MyFile3.txt", 3
End Sub
```

Alternativ können Sie auch mit den Konstanten arbeiten. Der Code würde dann wie folgt aussehen:

```
'=====
' Auf CD      Buchdaten\Beispiele\12_Extern\
' Dateiname   12_01_Explorer.xls
' Modul       mdl_06_Attributes
'=====

Sub Attributes2()
    SetAttr "C:\Codebook_12_Test\MyFile3.txt", vbReadOnly + vbHidden
End Sub
```

Um sämtliche Attribute zu entfernen, können Sie die Datei zurücksetzen. Verwenden Sie dazu wahlweise die Konstante vbNormal oder den Wert 0.

```
'=====
' Auf CD      Buchdaten\Beispiele\12_Extern\
' Dateiname   12_01_Explorer.xls
' Modul       mdl_06_Attributes
'=====

Sub Attributes3()
    SetAttr "C:\Codebook_12_Test\MyFile3.txt", 0
End Sub
```

Ob Sie nun mit Werten oder mit Konstanten arbeiten, ist Ihnen überlassen. Einige bevorzugen Konstanten, da diese aussagekräftiger sind, und andere wiederum arbeiten lieber mit Werten.

401 Datei- und Verzeichnisnamen auslesen

Wie bereits zu Beginn dieser Kategorie angesprochen, lassen sich sowohl Dateinamen als auch Namen von Verzeichnissen auslesen. In einem ersten Meldungsfenster werden die Dateinamen und in einem zweiten die Namen der Verzeichnisse angezeigt. In einer `MsgBox` lassen sich maximal 512 Zeichen ausgeben. Wenn diese Anzahl an Zeichen erreicht ist, wird *keine* Fehlermeldung angezeigt. Die `MsgBox` zeigt einfach nur 512 Zeichen an, der Rest bleibt Ihnen vorenthalten. Wenn Ihre Liste sehr umfangreich ist, geben Sie die Daten am besten in einem leeren Tabellenblatt aus. So können Sie sicherstellen, dass Sie auch wirklich alle Informationen erhalten.

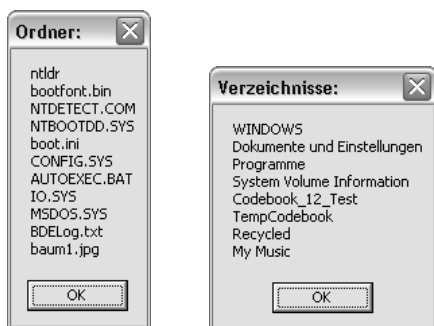


Abbildung 331: Ausgelesene Datei- und Verzeichnisnamen

Unser Beispiel ist so aufgebaut, dass zu Beginn der Prozedur ein Eingabefenster `InputBox` erscheint. Hier kann der gewünschte Pfad eingegeben werden. Der Pfad wird an die Variable `strInput` übergeben. In einer `If`-Entscheidung wird geprüft, ob ein gültiger Pfad eingegeben wurde. Danach werden der Variablen `strMsg` die gewünschten Informationen zugewiesen. Es werden dazu zwei Schleifen verwendet. In der ersten Schleife werden die Dateinamen `Files`, die sich in dem angegebenen Pfad befinden, ausgegeben. In der zweiten Schleife werden die Namen der Verzeichnisse `SubFolders` aufgelistet.

```
'=====
' Auf CD      Buchdaten\Beispiele\12_Extern\
' Dateiname   12_01_Explorer.xls
' Modul       mdl_07_FileDirList
'=====
```

```
Sub FileDirList()
    Dim objFSO As Object
    Dim strInput As String
    Dim strMsg As String
    Dim c As Object

    Set objFSO = CreateObject("Scripting.FileSystemObject")

    strInput = InputBox("Bitte den Pfad eingeben", Default:="C:\")
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

```

If Dir(strInput) = "" Then
    MsgBox "Der Pfad " & strInput & " existiert nicht."
Exit Sub
End If

For Each c In objFSO.GetFolder(strInput).Files
    strMsg = strMsg & c.Name & vbCrLf
Next
MsgBox strMsg, , "Ordner:"

strMsg = ""
For Each c In objFSO.GetFolder(strInput).SubFolders
    strMsg = strMsg & c.Name & vbCrLf
Next
MsgBox strMsg, , "Verzeichnisse:"

Set objFSO = Nothing
End Sub

```

402 Dateien suchen

Vielleicht fragen Sie sich nun, ob es eine Möglichkeit gibt, nur bestimmte Dateien auszugeben. Beispielsweise nur jene, welche mit einem »b« beginnen. Oder nur solche, die die Dateiendung *.xls aufweisen. Wir verwenden dazu die Eigenschaft `FileSearch`. Zudem benötigen wir noch zwei weitere Eigenschaften, nämlich `LookIn`, um den Pfad anzugeben, und `FileName`, um den Dateinamen, oder einen Teil davon, zu hinterlegen. In der `If`-Entscheidung wird mittels der Methode `Execute` geprüft, ob und wie viele Dateien gefunden wurden. Wenn die erste Entscheidung zutrifft, also wenn Dateien gefunden wurden, wird in einer `MsgBox` die Anzahl ausgegeben. Innerhalb dieser Entscheidung befindet sich eine Schleife, die die gefundenen Dateien `FoundFiles` an die Variable `strMsg` übergibt. Am Ende der Prozedur werden diese Informationen in einer weiteren `MsgBox` angezeigt. Wenn keine Dateien gefunden wurden, wird eine `MsgBox` angezeigt, die darauf hinweist, dass keine Dateien mit den entsprechenden Suchkriterien gefunden wurden. Es erfolgt dabei keine Unterscheidung zwischen Groß- und Kleinschreibung.

```

'=====
' Auf CD      Buchdaten\Beispiele\12_Extern\
' Dateiname   12_01_Explorer.xls
' Modul       mdl_08_FileListSearch
'=====

Sub FileListSearchedByName()
    Dim i As Integer
    Dim strMsg As String

    With Application.FileSearch
        .LookIn = "C:\"
        .Filename = "b*"
        If .Execute > 0 Then
            MsgBox "Es wurde(n) " & .FoundFiles.Count & _
                " Datei(en) gefunden"
            For i = 1 To .FoundFiles.Count

```

```
        strMsg = strMsg & .FoundFiles(i) & vbCrLf
    Next i
Else
    MsgBox "Es wurden keine Dateien gefunden"
End If
MsgBox strMsg
End With
End Sub
```

Wenn Sie den Code ausführen, wird das Nachrichtenfenster mit den gewünschten Angaben angezeigt. Je nach Ordner und Suchbegriff, den Sie eingegeben haben, wird die Anzeige unterschiedlich ausfallen.

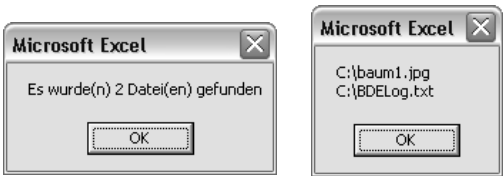


Abbildung 332: Es werden nur Dateien beginnend mit »b*.*« angezeigt

403 Dateien alphabetisch sortiert ausgeben

Sie können Dateien sortiert nach Dateinamen oder Dateityp ausgeben. Es stehen Ihnen fünf Werte zur Verfügung:

Konstante (SortBy:=...)	Erläuterung
msoSortByFileName (Standardwert)	Sortieren nach Dateinamen
msoSortByFileType	Sortieren nach Dateityp
msoSortByLastModified	Sortiert nach dem letzten Änderungsdatum
msoSortByNone	Keine Sortierung
msoSortBySize	Sortieren nach Dateigröße

Tabelle 87: Werte zum Sortieren

Wahlweise können die Daten aufsteigend oder absteigend sortiert (SortOrder:=...) im Nachrichtenfenster ausgegeben werden. Wenn Sie sich für eine aufsteigende Sortierung entscheiden, verwenden Sie den Wert msoSortOrderAscending, für absteigend msoSortOrderDescending.

Im nachfolgenden Code werden die Dateien aufsteigend nach Namen sortiert ausgegeben. Wir verwenden die Methode Execute, die Sie bereits aus dem vorangegangenen Beispiel kennen, gefolgt von den gewünschten Werten.

```
'=====
' Auf CD      Buchdaten\Beispiele\12_Extern\
' Dateiname   12_01_Explorer.xls
' Modul       mdl_09_FileListSorted
'=====
```

Grundlagen

Allgemein

Datum und Zeit

Formeln

Steuerungselemente

Befehlszeile

Objekte

Diagramme

Ereignisse

User-Formen

Web/Mail

Extern

Gemischtes

Spezial

```

Sub FileListSortByName()
    Dim i As Integer
    Dim strMsg As String

    With Application.FileSearch
        .LookIn = "C:\Codebook_12_Test"
        .Filename = "*.xls"
        If .Execute(SortBy:=msoSortByFileName, _
            SortOrder:=msoSortOrderAscending) > 0 Then
            MsgBox "Es wurde(n) " & .FoundFiles.Count & _
                " Datei(en) gefunden"
            For i = 1 To .FoundFiles.Count
                strMsg = strMsg & .FoundFiles(i) & vbCrLf
            Next i
        Else
            MsgBox "Es wurden keine Dateien gefunden"
        End If
    End With
    MsgBox strMsg
End Sub

```

404 Dateien aus Unterverzeichnissen auslesen

Wir erweitern das vorangegangene Rezept, indem wir zudem Unterverzeichnisse mit auslesen. Die Eigenschaft dazu lautet: `SearchSubFolders = True`. Dieser Eigenschaft muss noch der boolesche Wert `True` (Wahr) hinzugefügt werden.



Abbildung 333: Dateien inklusive Inhalt von Unterverzeichnissen auslesen

Es kann vorkommen, dass alte Suchergebnisse im Speicher hängen bleiben. Um diese zu entfernen, fügen wir innerhalb der `With`-Anweisung die Methode `NewSearch` ein.

Der komplette Code sieht nun wie folgt aus:

```
'=====
' Auf CD      Buchdaten\Beispiele\12_Extern\
' Dateiname   12_01_Explorer.xls
' Modul       mdl_10_FileListSubFolder
'=====

Sub FileListInclSubFolder()
    Dim i As Integer
    Dim strMsg As String

    With Application.FileSearch
        .NewSearch
        .LookIn = "C:\Codebook_12_Test"
        .Filename = "*.xls"
        .SearchSubFolders = True
        If .Execute(SortBy:=msoSortByFileName, _
            SortOrder:=msoSortOrderAscending) > 0 Then
            MsgBox "Es wurde(n) " & .FoundFiles.Count & _
                " Datei(en) gefunden"
            For i = 1 To .FoundFiles.Count
                strMsg = strMsg & .FoundFiles(i) & vbCrLf
            Next i
        Else
            MsgBox "Es wurden keine Dateien gefunden"
        End If
    End With
    MsgBox strMsg
End Sub
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

405 Verzeichnis erstellen

In diesem Rezept werden wir per VBA ein Verzeichnis erstellen. Die VBA-Anweisung lautet `MkDir`, gefolgt vom Laufwerk und dem gewünschten Verzeichnisnamen. `MkDir` ist die englische Abkürzung für »Make Directory«. Übersetzt heißt dies »Verzeichnis erstellen«.

```
'=====
' Auf CD      Buchdaten\Beispiele\12_Extern\
' Dateiname   12_01_Explorer.xls
' Modul       mdl_11_Directory
'=====

Sub CreateDirectory()
    MkDir ("C:\Monikas_Folder")
End Sub
```

Wenn Sie den Code ausführen und anschließend den Explorer starten, werden Sie feststellen, dass der Ordner auf dem Laufwerk `C:\` erstellt worden ist. Sollten Sie den Explorer bereits geöffnet haben, müssen Sie zuerst aktualisieren, bevor die Änderung sichtbar wird. Verwenden Sie die Taste F5.

406 Verzeichnis löschen

Die Anweisung, um einen Ordner zu löschen, lautet `RmDir`. Auch dies ist eine englische Abkürzung. Sie lautet ausgeschrieben »Remove Directory«, zu Deutsch »Verzeichnis entfernen«.

```
'=====
' Auf CD      Buchdaten\Beispiele\12_Extern\
' Dateiname   12_01_Explorer.xls
' Modul       mdl_11_Directory
'=====

Sub DeleteDirectory()
    RmDir ("C:\Monikas_Folder")
End Sub
```

407 Verzeichnis wechseln

Der Befehl, um in ein Verzeichnis zu wechseln, lautet `ChDir`. Diese Abkürzung steht für »Change Directory«, die Übersetzung dazu lautet »Verzeichnis wechseln«. Verwenden Sie diesen Befehl und geben Sie in den runden Klammern und eingeschlossen durch Anführungs- und Schlusszeichen den Pfad ein:

```
'=====
' Auf CD      Buchdaten\Beispiele\12_Extern\
' Dateiname   12_01_Explorer.xls
' Modul       mdl_11_Directory
'=====

Sub ChangeDirectory()
    ChDir ("C:\Codebook_12_Test")
End Sub
```

408 Größe eines Verzeichnisses ermitteln

Speicherplatz ist heutzutage wohl meist nicht mehr ein entscheidendes Problem, denn die Festplatten-Kapazität wird immer größer und in der Regel bieten sie mehr Platz an, als man jemals ausschöpfen kann. So bewegen sich die Größen ja längst im GigaByte-Bereich, was nicht zuletzt den Programmen und Anwendungen zuzuschreiben ist, die dies erfordern. Dennoch kann es interessant sein, zu erfahren, wie viel Speicher ein Ordner in Anspruch nimmt. Um dies zu ermitteln, verwenden wir die ActiveX-Komponente `FileSystemObject`. Das Schlüsselwort, um die Größe herauszufinden, lautet `Size`. Die Ausgabe erfolgt in Bytes.

```
'=====
' Auf CD      Buchdaten\Beispiele\12_Extern\
' Dateiname   12_01_Explorer.xls
' Modul       mdl_12_FolderSize
'=====

Sub FolderSize()
    Dim strPath As String
    Dim objFileS As Object
```

```
Dim objFolder As Object
Dim strMsg As String

strPath = "C:\Codebook_12_Test\"
If Dir(strPath) = "" Then
    MsgBox "Der Pfad " & strPath & " existiert nicht."
    Exit Sub
End If

Set objFileS = CreateObject("Scripting.FileSystemObject")
Set objFolder = objFileS.GetFolder(strPath)

MsgBox UCase(objFolder.Name) & " belegt " & _
        objFolder.Size & " Bytes."

Set objFileS = Nothing
Set objFolder = Nothing
End Sub
```

In einem Nachrichtenfenster wird in Bytes angezeigt, wie groß unser Testordner ist:

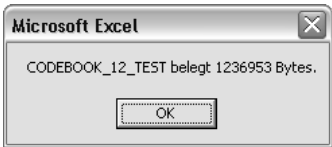


Abbildung 334: Größe eines Verzeichnisses ermitteln

Sie können die Angabe der Größe eines Ordners auch in Kilo- oder MegaByte angeben. Ein Kilo-Byte umfasst 1024 Bytes. Ein MegaByte wiederum hat 1024 KiloBytes. Im nachfolgenden Code werden entsprechende Berechnungen vorgenommen. Damit nur zwei Kommastellen angezeigt werden, verwenden wir die Funktion `Round`, die auf zwei Kommastellen auf- oder abrundet.

```
'=====
' Auf CD      Buchdaten\Beispiele\12_Extern\
' Dateiname  12_01_Explorer.xls
' Modul      mdl_12_FolderSize
'=====

Sub FolderSizeKMBM()
    Dim strPath As String
    Dim objFileS As Object
    Dim objFolder As Object
    Dim strMsg As String

    strPath = "C:\Codebook_12_Test\"
    If Dir(strPath) = "" Then
        MsgBox "Der Pfad " & strPath & " existiert nicht."
        Exit Sub
    End If
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

```

Set objFileS = CreateObject("Scripting.FileSystemObject")
Set objFolder = objFileS.GetFolder(strPath)

strMsg = "Ordnergröße: " & objFolder.Size & _
        " in Byte" & vbCrLf
strMsg = strMsg & _
        "Ordnergröße: " & Round(objFolder.Size / 1024, 2) & _
        " in KiloByte" & vbCrLf
strMsg = strMsg & _
        "Ordnergröße: " & Round(objFolder.Size / 1048576, 2) & _
        " in MegaByte"

MsgBox strMsg, , "Ordner: " & UCase(objFolder.Name)

Set objFileS = Nothing
Set objFolder = Nothing
End Sub

```

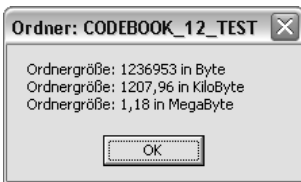


Abbildung 335: Verschiedene Größenangaben

409 Laufwerk oder Ordner?

Sie können bei Bedarf per VBA ermitteln, ob ein Laufwerk oder Dateiordner angesprochen wird. Die Eigenschaft dazu lautet `Type`. In unserem Beispiel wird nach der Fehlerbehandlung referenziert. Dies geschieht zuerst auf das Objekt. Danach auf ein Laufwerk und schließlich auf ein Verzeichnis. Alle Daten, die aus dem System ausgelesen werden, sollen in Großschrift `UCase` angezeigt werden. In der `MsgBox` wird angezeigt, um welchen Typ es sich bei dem referenzierten Objekt handelt.

```

' =====
' Auf CD      Buchdaten\Beispiele\12_Extern\
' Dateiname   12_01_Explorer.xls
' Modul       mdl_13_FileType
' =====

Sub FileType()
    Dim objFSO As Object
    Dim objFolder As Object
    Dim objFile As Object

    On Error GoTo ErrorHandler

    Set objFSO = CreateObject("Scripting.FileSystemObject")
    Set objFolder = objFSO.GetFolder("F:")
    Set objFile = objFSO.GetFolder("C:\Codebook_12_Test")

```

```
MsgBox UCase(objFSO.GetDriveName(objFolder)) & _  
    " ... ist ein ... " & _  
    UCase(objFolder.Type) & _  
    vbCrLf & _  
    UCase(objFile.Name) & _  
    " ... ist ein ... " & _  
    UCase(objFile.Type)  
  
Set objFSO = Nothing  
Set objFolder = Nothing  
Set objFile = Nothing  
  
Exit Sub  
ErrorHandler:  
    MsgBox "Es wurde kein gültiges Laufwerk oder kein gültiger " & _  
        "Dateiname eingegeben, " & vbCrLf & _  
        "oder es muss eine Diskette/CD eingelegt werden."  
End Sub
```

Die Anzeige im Meldungsfenster sieht wie folgt aus:

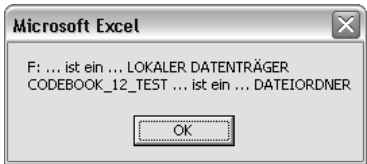


Abbildung 336: Typ ermitteln

410 Laufwerksbuchstaben ermitteln

Sie können per VBA ermitteln, welche Laufwerksbuchstaben auf Ihrem System vorhanden sind und welche Namen diesen zugewiesen wurden. Das Objekt dazu lautet `Drives`. Innerhalb von `Drives` wird der Typ `DriveType` angesprochen. Dabei muss ein Wert übergeben werden. Es stehen uns insgesamt fünf Werte zur Verfügung:

Wert	Bedeutung
0	Unbekannter Typ
1	Auswechselbar (z.B. Diskette oder CD)
2	Fest (z.B. Festplatte)
3	Netzwerk
4	CD-ROM
5	RAM-Disk

Tabelle 88: `DriveTyp`-Eigenschaften

In einer Schleife werden sämtliche Laufwerke ermittelt. Danach wird in einer `If...Then...Else`-Entscheidung geprüft, ob es sich um ein Netzwerklaufwerk `ShareName` oder ein lokales Laufwerk `VolumeName` handelt. Diese Werte werden der Variablen `strName` übergeben, die wir später verwen-

Grund-lagen

Allge-mein

Datu-Zeit

tungen

Steuer-elemen

Befehl-leisten

Objekt

Dia-gramm

Ereig-nisse

User-Form

Web/Mail

Extern

Ge-mischtes

Specia

den, um eine Anzeige im Meldungsfenster zu erhalten. Nach der If-Entscheidung wird die MsgBox für die Ausgabe vorbereitet. Es werden der Variablen `strMsg` der Laufwerksbuchstabe `DriveLetter` sowie die in der Entscheidung ermittelte Variable `strName` übergeben. Als Anzeige erhalten Sie dann eine vollständige MsgBox mit den gewünschten Angaben:



Abbildung 337: Laufwerksbuchstaben und Text ermitteln

```
'=====
' Auf CD      Buchdaten\Beispiele\12_Extern\
' Dateiname  12_01_Explorer.xls
' Modul      mdl_14_DriveList
'=====
```

```
Sub DriveList()
    Dim objFSO As Object
    Dim c As Object
    Dim strMsg As String
    Dim strName As String

    On Error GoTo ErrorHandler
    Set objFSO = CreateObject("Scripting.FileSystemObject")

    For Each c In objFSO.Drives
        If c.DriveType = 3 Then
            strName = c.ShareName    ' Netzwerklaufwerk
        Else
            strName = c.VolumeName  ' Lokales Laufwerk
        End If
        strMsg = strMsg & c.DriveLetter & " - " & strName & vbCrLf
    Next

    MsgBox strMsg
    Set objFSO = Nothing
    Exit Sub

ErrorHandler:
    MsgBox "Das Laufwerk " & c & " ist nicht bereit." & _
        vbCrLf & "Legen Sie eine Diskette/CD ein."
End Sub
```

411 Laufwerk-Eigenschaften

Zum Thema Laufwerke lassen sich noch weitere Eigenschaften ermitteln. Der nachfolgende Code ist so aufgebaut, dass beim Ausführen des Codes ein neues Tabellenblatt erstellt wird. In dieses Tabellenblatt werden die gewünschten Informationen geschrieben. Die Beschreibung der verschiedenen Eigenschaften können Sie direkt dem Code entnehmen:

```
'=====
' Auf CD      Buchdaten\Beispiele\12_Extern\
' Dateiname  12_01_Explorer.xls
' Modul      mdl_15_DriveListComplete
'=====

Sub DriveListComplete()
    Dim objFSO As Object
    Dim objDrive As Object

    Set objFSO = CreateObject("Scripting.FileSystemObject")
    Set objDrive = objFSO.GetDrive("F:")

    ActiveWorkbook.Worksheets.Add Before:=Worksheets(1)
    Range("A1") = objDrive           ' Laufwerksbuchstabe
    Range("A2") = objDrive.AvailableSpace ' Verfügbarer Speicherplatz
    Range("A3") = objDrive.DriveType   ' Laufwerktyp
    Range("A4") = objDrive.FileSystem ' Dateisystem
    Range("A5") = objDrive.FreeSpace   ' Freier Speicherplatz
    Range("A6") = objDrive.IsReady     ' Bereit (Wahr/Falsch)
    Range("A7") = objDrive.Path        ' Pfad
    Range("A8") = objDrive.RootFolder  ' Stammordner e. Laufwerks
    Range("A9") = objDrive.ShareName    ' Netzwerk-Laufwerk
    Range("A10") = objDrive.TotalSize  ' Gesamtkap. des Laufwerks
    Range("A11") = objDrive.VolumeName ' Laufwerksname

    Set objFSO = Nothing
    Set objDrive = Nothing
End Sub
```

412 System-Informationen auslesen (Environ)

Bevor wir uns dem nächsten Thema zuwenden, werden wir Ihnen zeigen, dass auch System-Informationen per VBA ausgelesen werden können. Die Funktion dazu lautet Environ. Auch hier wird beim Ausführen des Codes ein neues Tabellenblatt angelegt. Den Zellen in Spalte A ist der Index zu entnehmen und den Zellen in Spalte B die zugehörige System-Information.

```
'=====
' Auf CD      Buchdaten\Beispiele\12_Extern\
' Dateiname  12_01_Explorer.xls
' Modul      mdl_16_ReadSysteminformations
'=====

Sub ReadSysteminformations()
    Dim i As Integer
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

```
Dim strMsg As String
```

```
ActiveWorkbook.Worksheets.Add Before:=Worksheets(1)
```

```
i = 1
```

```
Do While Environ(i) <> ""
```

```
Cells(i, 1) = i & ".)"
```

```
Cells(i, 2) = Environ(i)
```

```
i = i + 1
```

```
Loop
```

```
End Sub
```

413 Das File System Object (FSO)

Das *File System Object*-Objekt, kurz FSO, wurde bereits in einigen Rezepten dieser Kategorie verwendet. Um es nicht an Erläuterungen mangeln zu lassen, möchten wir Ihnen nun einige Erläuterungen dazu abgeben.

Es stehen drei unterschiedliche Bibliotheken zur Verfügung, mit denen Sie externe Dateien ansprechen können. Von daher gibt es mehrere Möglichkeiten, einen Code zu programmieren. Seit der Version 2000 steht auch in Excel die *Bibliothek Microsoft Scripting Library* zur Verfügung. Diese ermöglicht es, sich über das FSO Zugriff auf andere Dateien zu verschaffen, in diese Dateien Daten zu schreiben oder Daten auszulesen.

Um eine gewünschte Bibliothek in Excel verwenden zu können, muss sie aktiviert sein. Erst dann stehen Ihnen die darin enthaltenen Eigenschaften, Methoden und Objekte zur Verfügung.

Um eine Bibliothek zu aktivieren, klicken Sie in der VBA-Umgebung von Excel auf den Menüpunkt EXTRAS | VERWEISE. In dem Dialogfenster, das sich daraufhin öffnet, sind alle aktivierbaren Bibliotheken ersichtlich. Sie können eine oder mehrere solcher Bibliotheken aktivieren, indem Sie in das Kontrollkästchen vor dem Namen klicken. Ein Häkchen lässt erkennen, dass die Bibliothek aktiv ist. Aktivieren Sie nur diejenigen, die Sie wirklich benötigen, denn je mehr Bibliotheken aktiv sind, desto langsamer werden Ihre Prozeduren. Die Bibliothek, die wir benötigen, nennt sich *Microsoft-Scripting-Runtime*. Dem Pfad im unteren Teil des Dialogfensters (siehe Abbildung 338) können Sie entnehmen, um welche *.dll (Dynamic Link Library) es sich dabei handelt.

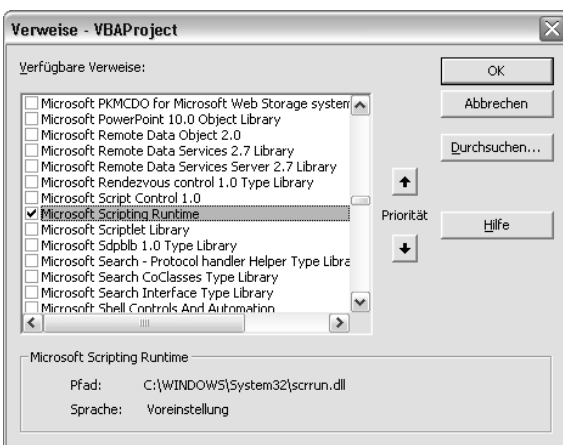


Abbildung 338: Erforderliche Bibliothek für die FSO-Befehle

Methoden	Beispiel	Beschreibung
BuildPath	<code>MsgBox objFSO.BuildPath _ ("C:\Codebook_12_Test\", _ "xxx.txt")</code>	Bildet einen Pfad und Dateinamen
CopyFile	<code>objFSO.CopyFile _ "C:\Codebook_12_Test\MyFile.txt", _ "C:\Codebook_12_Test\MyFile2.txt"</code>	Kopiert eine Datei
CopyFolder	<code>objFSO.CopyFolder _ "C:\Codebook_12_Test", _ "C:\Codebook_12_Test_New"</code>	Kopiert ein Verzeichnis
DeleteFile	<code>objFSO.DeleteFile _ "C:\Codebook_12_Test\MyFile.txt"</code>	Löscht eine Datei
DeleteFolder	<code>objFSO.DeleteFolder _ "C:\Codebook_12_Test", True</code>	Löscht eine Datei auch, wenn sie schreibgeschützt ist (True)
DriveExists	<code>MsgBox _ objFSO.DriveExists("C:\")</code>	Prüft, ob ein Laufwerk vorhanden ist
FileExists	<code>MsgBox _ objFSO.DriveExists _ ("C:\Codebook_12_Test\MyFile.txt")</code>	Prüft, ob eine Datei vorhanden ist
FolderExists	<code>MsgBox _ objFSO.FolderExists _ ("C:\Codebook_12_Test")</code>	Prüft, ob ein Verzeichnis vorhanden ist
GetAbsolutePathName	<code>MsgBox _ objFSO.GetAbsolutePathName("C:")</code>	Gibt vollständigen Pfad zurück
GetBaseName	<code>MsgBox _ objFSO.GetBaseName _ ("C:\Codebook_12_Test\MoreTests")</code>	Zeigt nur den letzten Verzeichnisnamen an
GetDrive	<code>MsgBox _ objFSO.GetDrive("C")</code>	Zeigt das Laufwerk an
GetDriveName	<code>MsgBox _ objFSO.GetDriveName _ ("C:\Codebook_12_Test\MoreTests")</code>	Zeigt nur den Laufwerk-Buchstaben an
GetFile	<code>MsgBox _ objFSO.GetFile _ ("C:\Codebook_12_Test\MyFile.txt")</code>	Zeigt Datei und Pfad an
GetFileName	<code>MsgBox _ objFSO.GetFileName _ ("C:\Codebook_12_Test\MyFile.txt")</code>	Zeigt nur den Dateinamen an
GetFolder	<code>MsgBox _ objFSO.GetFolder _ ("C:\Codebook_12_Test\MoreTests")</code>	Zeigt den Pfad an

Tabelle 89: Eigenschaften und Methoden zum FileSystemObject

Methoden	Beispiel	Beschreibung
GetParentFolder-Name	MsgBox _ objFSO.GetParentFolderName _ ("C:\Codebook_12_Test\MoreTests")	Zeigt nur das Hauptverzeichnis plus Laufwerk an
GetSpecialFolder	MsgBox _ objFSO.GetSpecialFolder(2)	Liefert Windows-System-Informationen 0 = Windows-Ordner 1 = System-Ordner 2= Temp-Ordner
GetTempName	MsgBox _ objFSO.GetTempName	Gibt willkürlich eine temporäre Datei aus
MoveFile	objFSO.MoveFile _ "C:\Codebook_12_Test\MyFile.txt", _ "C:\Codebook_12_Test\MoreTests\"	Verschiebt eine Datei
MoveFolder	objFSO.MoveFolder _ "C:\Codebook_12_Test*", _ "C:\Test_New\"	Verzeichnisse oder Dateien verschieben (Quelle / Ziel)

Tabelle 89: Eigenschaften und Methoden zum FileSystemObject (Forts.)

414 Eine Textdatei erzeugen

Wenden wir uns dem Umgang mit Textdateien zu. Mit dem *FilySystemObject*-Objekt lassen sich nicht nur Daten aus dem Explorer auslesen, sondern Sie haben auch die Möglichkeit, Textdateien zu erzeugen und Daten aus Excel-Tabellenblättern in Textdateien zu schreiben.

In unserem ersten Beispiel werden wir eine neue Textdatei auf dem Laufwerk C:\ im Ordner *Test* erzeugen. In diese Datei sollen zwei Textzeilen geschrieben werden (siehe Abbildung 340).

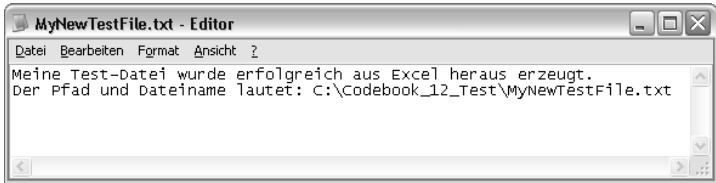


Abbildung 340: Eine Textdatei aus Excel heraus erzeugen

Oberhalb der Prozedur wird die öffentliche Variable `objFSO` deklariert. Damit der Code lauffähig ist, muss der Verweis auf die Bibliothek *Microsoft-Scripting-Runtime* gesetzt sein.

Sowohl den Pfad als auch den Dateinamen werden wir als Variable ansprechen (`objPath` und `objFileName`). Wir deklarieren die Variablen entsprechend zu Beginn der Prozedur. Bei umfangreichen Prozeduren hat dies den Vorteil, dass innerhalb des Codes nicht jedes Mal, wo es erforderlich ist, der ganze Pfad und Dateiname eingetippt werden muss.

Nach der Deklaration weisen wir den Variablen die gewünschten Werte zu. Direkt im Anschluss wird das Objekt zum Erzeugen einer neuen Textdatei referenziert (`Set`). Die Methode dazu lautet `CreateTextFile`, gefolgt vom Pfad- und Dateinamen. Sie können optional den Wert `Overwrite`,

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

gefolgt vom gewünschten booleschen Wert einsetzen. Wenn der Wert auf `True`, also Wahr eingestellt wird, wird die Datei kommentarlos überschrieben, sofern bereits eine Datei mit diesem Namen bestehen sollte.

Die Syntax zum Erzeugen einer Textdatei lautet:

```
Objekt.CreateTextFile(Dateiname[, überschreiben[, Unicode]])
```

Nun können wir damit beginnen, die Datei mit Text zu füllen. Wir verwenden eine `With`-Anweisung (`With objNewFile`). Innerhalb dieser Anweisung werden zwei Textzeilen geschrieben. Für jede Zeile, die geschrieben werden soll, benötigen wir eine `WriteLine`-Methode, gefolgt vom gewünschten Text. Am Ende der `With`-Anweisung wird sichergestellt, dass die Textdatei geschlossen wird (`Close`).

Da eine Textdatei kommentarlos erzeugt oder überschrieben wird, kann es sinnvoll sein, nach Ablauf der Prozedur eine eigene Nachricht `MsgBox` anzuzeigen.

```
'=====
' Auf CD      Buchdaten\Beispiele\12_Extern\
' Dateiname   12_02_TextFile.xls
' Modul       mdl_01_CreateNewTextFile
'=====
```

```
Public objFSO As New FileSystemObject

' Verweis auf Microsoft Scripting Runtime muss gesetzt sein
Sub CreateNewTextFile()
    Dim strPath As String
    Dim strFileName As String
    Dim objNewFile As Object

    strPath = "C:\Codebook_12_Test\"
    strFileName = "MyNewTestFile.txt"

    Set objNewFile = objFSO.CreateTextFile _
        (strPath & strFileName, _
        Overwrite:=True)

    With objNewFile
        .WriteLine ("Meine Test-Datei wurde erfolgreich " & _
            "aus Excel heraus erzeugt.")
        .WriteLine ("Der Pfad und Dateiname lautet: " & _
            strPath & strFileName)
        .Close
    End With

    MsgBox "Die Datei wurde erfolgreich erzeugt", _
        Title:=strPath & strFileName

    Set objNewFile = Nothing
End Sub
```

415 Eine Textdatei ergänzen

Im letzten Rezept haben wir eine komplett neue Textdatei erzeugt. Sie können eine bestehende Textdatei jedoch auch um weitere Zeilen ergänzen. Dazu verwenden wir die Methode `OpenTextFile`, die eine Textdatei, zum Auslesen oder Schreiben von Daten, öffnet.

Die Syntax zum Öffnen einer Datei lautet:

```
Objekt.OpenTextFile(Dateiname[, E/A-Modus[, erstellen[, Format]]])
```

Der Methode `OpenTextFile` können mehrere Parameter übergeben werden. Zuerst muss der Pfad und `Dateiname` festgelegt werden. Danach folgen die Angaben, ob die Datei nur zum Lesen von Daten geöffnet werden soll, oder auch zum Schreiben (`E/A-Modus`). Schließlich kann noch festgelegt werden, in welchem Format die Datei geöffnet werden soll. Sie können dabei wahlweise die Konstante oder den entsprechenden Wert verwenden.

Konstante E/A-Modus	Wert	Beschreibung
<code>ForReading</code>	1	Nur lesen
<code>ForWriting</code>	2	Eine bestehende Datei wird überschrieben
<code>ForAppending</code>	8	Es können Zeilen am Ende der Datei ergänzt werden

Tabelle 90: Ein- oder Ausgabemodus

Konstante Format	Wert	Beschreibung
<code>TristateUseDefault</code>	-2	Datei mit Systemvorgaben öffnen
<code>TristateTrue</code>	-1	Als Unicode-Datei öffnen
<code>TristateFalse</code>	0	Als ASCII-Datei öffnen

Tabelle 91: Format

In unserem Beispiel wird der Variablen `objOpenFile` der Pfad und `Dateiname` übergeben. Die Datei soll um weitere Zeilen ergänzt werden. Wir verwenden somit den Wert `ForAppending`. Die Datei soll im ASCII-Format geöffnet werden, der Wert dafür lautet `TristateFalse`.

Nachdem die Variable die gewünschten Werte enthält, kann sie in der `With`-Anweisung verwendet werden, um weitere Zeilen in die Datei zu schreiben. Innerhalb der Datei sollen zuerst zwei Leerzeilen eingefügt werden, wir verwenden dazu die Methode `WriteBlankLines`, gefolgt von der Anzahl der zu schreibenden Leerzeilen. Danach wird ein Text ohne Zeilenumbruch eingefügt. Dazu verwenden wir die Methode `Write`. Danach werden zwei Zeilen mit Zeilenumbruch eingefügt. Die Methode dazu lautet `WriteLine`.

Methode	Beschreibung
<code>Write</code>	Text einfügen. Am Ende der Zeile erfolgt kein Zeilenumbruch.
<code>WriteLine</code>	Text einfügen. Am Ende der Zeile erfolgt ein Zeilenumbruch.
<code>WriteBlankLines</code>	Leerzeilen einfügen.

Tabelle 92: Methoden zum Schreiben

Grund-lagen
Allge-mein
Datu Zeit
tungen
Steuer elemen
Befehl leisten
Objekt
Dia-gramm
Ereig-nisse
User-Form
Web/Mail
Extern
Ge-misch tes
Specia

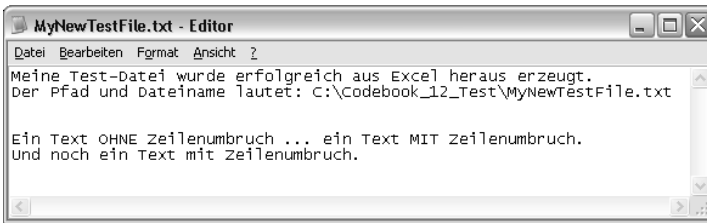


Abbildung 341: Erweiterte Textdatei

Um das Beispiel abzurunden, wird auch hier am Ende der Prozedur eine Meldung ausgegeben, die mitteilt, dass die Zeilen hinzugefügt wurden.

```
'=====
' Auf CD      Buchdaten\Beispiele\12_Extern\
' Dateiname  12_02_TextFile.xls
' Modul      mdl_02_WriteTextFile
'=====
```

```
Public objFSO As New FileSystemObject
```

```
' Verweis auf Microsoft Scripting Runtime muss gesetzt sein
```

```
Sub WriteTextFile()
```

```
    Dim strPath As String
```

```
    Dim strFileName As String
```

```
    Dim objOpenFile As Object
```

```
    strPath = "C:\Codebook_12_Test\"
```

```
    strFileName = "MyNewTestFile.txt"
```

```
    Set objOpenFile = objFSO.OpenTextFile _
        (strPath & strFileName, _
        ForAppending, TristateFalse)
```

```
    With objOpenFile
```

```
        .WriteBlankLines 2
```

```
        .Write "Ein Text OHNE Zeilenumbruch ... "
```

```
        .WriteLine "ein Text MIT Zeilenumbruch."
```

```
        .WriteLine "Und noch ein Text mit Zeilenumbruch."
```

```
    .Close
```

```
    End With
```

```
    MsgBox "Die Datei wurde erfolgreich ergänzt", _
        Title:=strPath & strFileName
```

```
    Set objOpenFile = Nothing
```

```
End Sub
```

416 Eine Textdatei auslesen

Das Verfahren, um Dateien auszulesen, ist dem vorangegangenen Rezept ähnlich. Es wird ebenfalls mit der `OpenTextFile`-Methode gearbeitet. Um nur Daten zu lesen, geben wir diesmal den

Wert `ForReading` an. Es stehen verschiedene Methoden und Eigenschaften zur Verfügung, die für das Auslesen benutzt werden können. Eine Zusammenfassung können Sie den Tabellen 93 und 94 entnehmen.

Methode	Beschreibung
<code>Read</code>	Zeichen lesen
<code>ReadLine</code>	Zeilen lesen
<code>ReadAll</code>	Gesamten Text lesen
<code>Skip</code>	Zeichen überspringen
<code>SkipLine</code>	Zeilen überspringen

Tabelle 93: Methoden zum Auslesen von Textdateien

Eigenschaft	Beschreibung
<code>Line</code>	Zeilennummer zurückgeben
<code>Column</code>	Spaltennummer zurückgeben
<code>AtEndOfLine</code>	Zeilenende
<code>AtEndOfStream</code>	Dateiende

Tabelle 94: Eigenschaften zum Auslesen von Textdateien

In unserer nachfolgenden Prozedur werden insgesamt vier Meldungsfenster in Folge auf dem Bildschirm ausgegeben. Den Inhalt der Meldungsfenster können Sie der Beschreibung des Codes entnehmen:

```
'=====
' Auf CD      Buchdaten\Beispiele\12_Extern\
' Dateiname  12_02_TextFile.xls
' Modul      mdl_03_ReadTextFile
'=====

Public objFSO As New FileSystemObject

' Verweis auf Microsoft Scripting Runtime muss gesetzt sein
Sub ReadTextFile1()
    Dim strPath As String
    Dim strFileName As String
    Dim objReadFile As Object

    strPath = "C:\Codebook_12_Test\"
    strFileName = "MyNewTestFile.txt"

    Set objReadFile = objFSO.OpenTextFile _
        (strPath & strFileName, _
        ForReading, TristateFalse)

    With objReadFile
        MsgBox .Read(5)      ' Die ersten fünf Zeichen auslesen
```

Grundlagen
Allgemein
Datum und Zeit
Arithmetische Funktionen
Steuerungselemente
Befehlsstrukturen
Objektmodelle
Diagramme
Ereignisse
User-Formen
Web/Mail
Extern
Gemischtes
Spezial

```

MsgBox .ReadLine      ' Den Rest der Zeile auslesen
.SkipLine             ' Eine Linie wird übersprungen
MsgBox .ReadAll       ' Den Rest der Datei auslesen
.Close               ' Datei schließen
End With

Set objReadFile = Nothing
End Sub

```

Unser zweites Beispiel zeigt auf, wie eine Textdatei Zeile für Zeile bis zum Ende der Datei `AtEndOfStream` ausgelesen werden kann. Wir verwenden eine `Do...Loop`-Schleife, worin die Zeilen nach und nach der Variablen `strText` übergeben werden. Zu Beginn jeder Zeile soll die Zeilennummer ausgelesen werden (`Line`).



Abbildung 342: Textdatei Zeile für Zeile auslesen

```

' =====
' Auf CD      Buchdaten\Beispiele\12_Extern\
' Dateiname   12_02_TextFile.xls
' Modul       mdl_03_ReadTextFile
' =====

```

```

Public objFSO As New FileSystemObject

' Verweis auf Microsoft Scripting Runtime muss gesetzt sein
Sub ReadTextFile2()
    Dim strPath As String
    Dim strFileName As String
    Dim objReadFile As Object
    Dim strText As String

    strPath = "C:\Codebook_12_Test\"
    strFileName = "MyNewTestFile.txt"

    Set objReadFile = objFSO.OpenTextFile _
        (strPath & strFileName, _
        ForReading, TristateFalse)

    ' Die Zeilennummer angeben (.Line)
    ' Bis zum Ende der Datei Zeile für Zeile auslesen
    Do While objReadFile.AtEndOfStream <> True
        strText = strText & objReadFile.Line & "). " & _
            objReadFile.ReadLine & vbCrLf
    Loop

```

```
Loop
objReadFile.Close

MsgBox strText

Set objReadFile = Nothing
End Sub
```

417 Import aus Textdatei

Nachdem wir uns mit dem Schreiben und Lesen von Textdateien befasst haben, wollen wir uns ansehen, wie die Daten einer Textdatei nach Excel übertragen werden können. Meist werden Daten, die in Textform vorliegen, aus einem Großsystem bzw. aus einer Datenbank ausgelesen. Sie sind durch ein Zeichen voneinander getrennt. Häufig handelt es sich dabei um ein Semikolon. Auch andere Zeichen sind denkbar. Um solche Daten zu importieren, ist nicht immer ein VBA erforderlich. Excel selbst bietet eine Möglichkeit zum Textimport an.

Der einfachste Weg führt über das Öffnen der Datei unter Angabe des Dateityps (Menüpunkt DATEI | ÖFFNEN). Für unseren Textimport wählen wir eine Datei, in der die Daten durch einen Bindestrich (-) voneinander getrennt sind. Wir verwenden zudem ein amerikanisches Datumsformat.

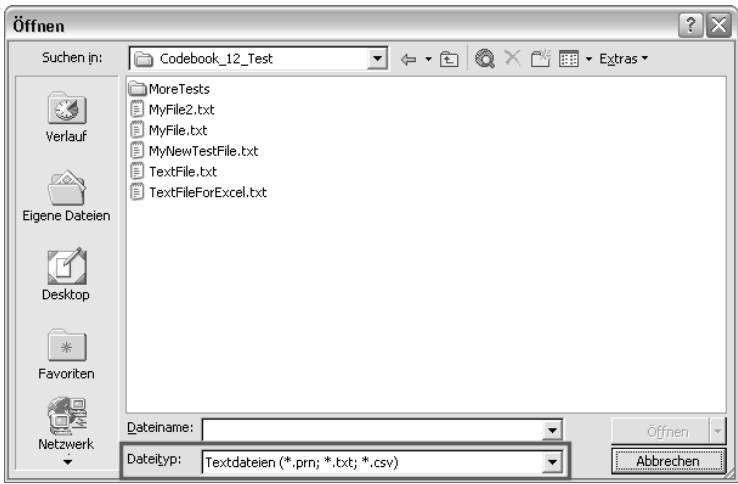


Abbildung 343: Datei mit dem Typ *.txt öffnen

Excel erkennt automatisch, dass es sich um eine Datei handelt, die konvertiert werden muss, und öffnet den entsprechenden Assistenten. Hier haben Sie nun die Möglichkeit, zwischen zwei Optionsfeldern zu wählen: GETRENNT oder FESTE BREITE. Wenn Sie GETRENNT wählen, können Sie im zweiten Schritt das Trennzeichen der Quelldatei angeben, z.B. ein Semikolon. Wenn Sie FESTE BREITE wählen, können Sie im zweiten Schritt selbst bestimmen, wo die Daten getrennt werden sollen. Je nachdem, wie die Quelldatei aufgebaut ist, fällt die Wahl also auf die eine oder andere Option. In unserem Fall, wo die Daten in der Textdatei durch einen Bindestrich getrennt sind, wählen wir GETRENNT.

Grundlagen

Allgemein

Datum und Zeit

Formeln

Steuerelemente

Befehlszeile

Objekt

Diagramm

Ereignisse

User-Form

Web/Mail

Extern

Gemischtes

Spezial

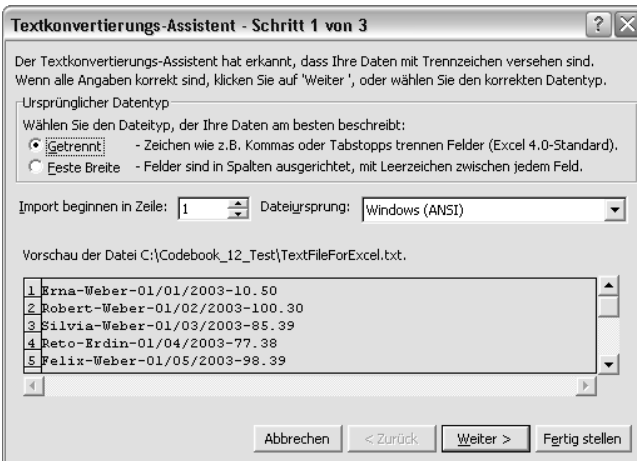


Abbildung 344: Textkonvertierung in fester Breite

Klicken Sie nach der Auswahl auf die Schaltfläche WEITER, um zum zweiten Schritt des Assistenten zu gelangen. Hier haben Sie die Möglichkeit, ein oder mehrere Trennzeichen auszuwählen. Sie können bei Bedarf auch ein eigenes Trennzeichen eingeben, so wie das in unserem Beispiel der Fall ist.

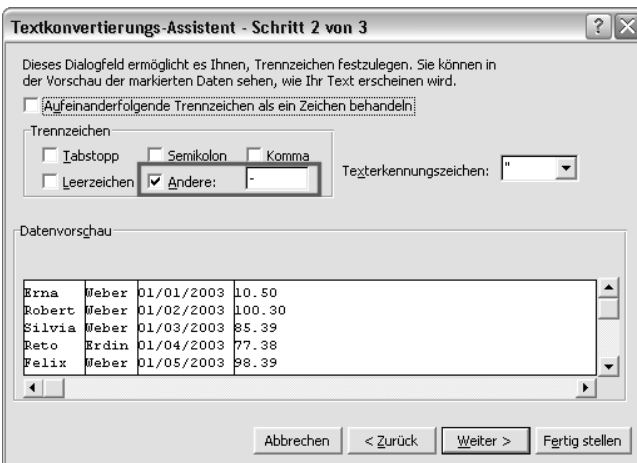


Abbildung 345: Listentrennzeichen auswählen

Nach der Auswahl des Trennzeichens klicken Sie erneut auf die Schaltfläche WEITER, um zum dritten und letzten Schritt des Assistenten zu gelangen. Falls die zu importierende Datei Datumswerte enthält, können Sie hier das Format der Quelldatei auswählen. Dies ist dann erforderlich, wenn die Datei beispielsweise aus einem Land stammt, wo das Datum in anderer Reihenfolge geschrieben wird. In den USA z.B. wird der 15. Januar 2003 als 01/15/2003 geschrieben, bei uns hingegen schreibt man 15.01.2003. Von daher ist es sehr wichtig, dass eine korrekte Umwandlung stattfindet. Auch Dezimal und 1000er-Trennzeichen können sich unterscheiden. So wird beispielsweise in der Schweiz als Dezimalzeichen ein Punkt (.) verwendet, in Deutschland hingegen ein Komma (,). Um dies umzustellen, klicken Sie im dritten Schritt des Assistenten auf die Schalt-

fläche WEITERE. Wenn keine spezielle Einstellung gewählt wird, wird mit den Systemeinstellungen von Windows gearbeitet.

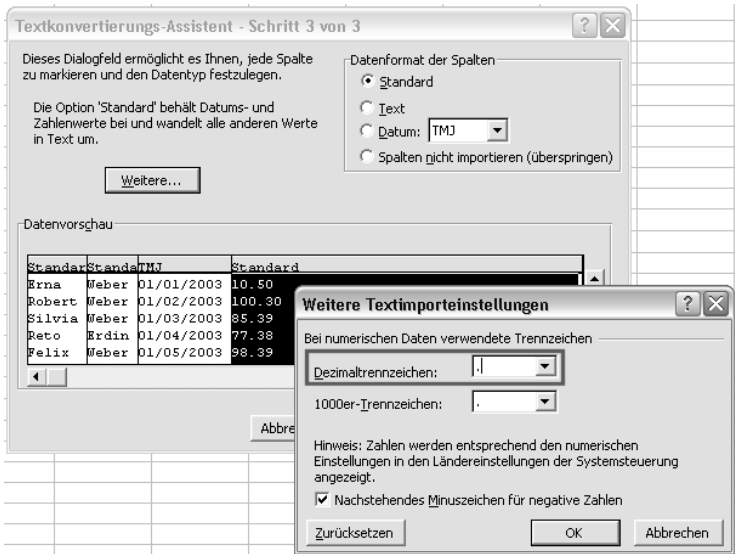


Abbildung 346: Import-Formate auswählen

Nachdem Sie diese Einstellungen vorgenommen haben, sind alle notwendigen Schritte für einen korrekten Import getan. Klicken Sie nun auf die Schaltfläche FERTIG STELLEN, um die Daten zu importieren. Wenn alle Angaben korrekt erfolgt sind, werden die Felder des Excel-Tabellenblattes wunschgemäß ausgefüllt sein.

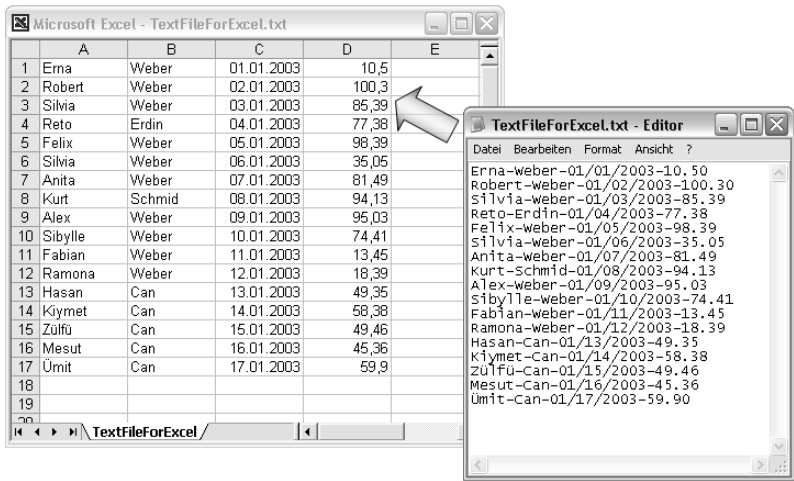


Abbildung 347: Excel- und Textdatei im Vergleich

Sie können Daten zwar mit dem FSO nach Excel importieren, aber Excel bietet auch eigene VBA-Befehle an. Sehen wir uns zuerst den Import über das FSO an. Die nachfolgende Prozedur verwendet eine Do...Loop-Schleife. Die Schleife wird so lange ausgeführt, bis das Ende der Datei

Grundlagen

Allgemein

Datum und Zeit

Funktionen

Steuerelemente

Befehle

Objekte

Diagramme

Ereignisse

UserForm

Web/Mail

Extern

Gemischtes

Spezial

erreicht ist (AtEndOfStream). Innerhalb dieser Schleife geben wir an, wohin die Daten in Excel geschrieben werden sollen. In unserem Beispiel ins erste Tabellenblatt, in die erste Spalte. Die Zeilen ermitteln wir mit `.Line`. Auf diese Weise entsprechen sie den Zeilennummern der Textdatei. Die Daten werden nun Zeile für Zeile in das Excel-Tabellenblatt geschrieben. Der Nachteil dabei ist, dass keine Trennung der Daten erfolgt. In Excel werden die Daten in eine Spalte geschrieben, ohne Berücksichtigung des Bindestrichs.

```
' =====
' Auf CD      Buchdaten\Beispiele\12_Extern\
' Dateiname   12_02_TextFile.xls
' Modul       mdl_04_ImportTextFile
' =====
```

```
Public objFSO As New FileSystemObject

' Verweis auf Microsoft Scripting Runtime muss gesetzt sein
Sub ImportTextFile1()
    Dim strPath As String
    Dim strFileName As String
    Dim objReadFile As Object

    strPath = "C:\Codebook_12_Test\"
    strFileName = "TextFile.txt"

    Set objReadFile = objFSO.OpenTextFile _
        (strPath & strFileName, _
        ForReading, TristateFalse)

    ActiveWorkbook.Worksheets.Add Before:=Worksheets(1)
    Do While objReadFile.AtEndOfStream <> True
        Worksheets(1).Cells(objReadFile.Line, 1) = _
            objReadFile.ReadLine()
    Loop

    Set objReadFile = Nothing
End Sub
```

Der einfachste Weg, um die Excel-eigenen Befehle zu erfahren, ist, einen solchen Import mit dem Makro-Rekorder aufzuzeichnen und anschließend, falls erforderlich, anzupassen. Genau dies haben wir hier getan. Natürlich können Sie dabei die Werte, die nicht benötigt werden, entfernen. Dem aufgezeichneten Code können Sie entnehmen, dass zu Beginn das Verzeichnis gewechselt wurde (ChDir). Danach wurde die Textdatei unter Angabe des Pfades und des Dateinamens mit der Methode `OpenText` geöffnet.

Die Prozedur sieht, übersichtlich strukturiert, wie folgt aus:

```
' =====
' Auf CD      Buchdaten\Beispiele\12_Extern\
' Dateiname   12_02_TextFile.xls
' Modul       mdl_04_ImportTextFile
' =====
```

```
Sub ImportTextFile2()  
    ChDir "C:\Codebook_12_Test"  
  
    Workbooks.OpenText _  
        Filename:="C:\Codebook_12_Test\TextFileForExcel.txt", _  
        Origin:=xlWindows, _  
        StartRow:=1, _  
        DataType:=xlDelimited, _  
        TextQualifier:=xlDoubleQuote, _  
        ConsecutiveDelimiter:=False, _  
        Tab:=False, _  
        Semicolon:=False, _  
        Comma:=False, _  
        Space:=False, _  
        Other:=True, _  
        OtherChar:="-", _  
        FieldInfo:=Array(Array(1, 1), Array(2, 1), _  
                          Array(3, 3), Array(4, 1)), _  
        DecimalSeparator=".", _  
        TrailingMinusNumbers:=True  
End Sub
```

Die Syntax dazu lautet:

Ausdruck.OpenText(Filename, Origin, StartRow, DataType, TextQualifier, ConsecutiveDelimiter, Tab, Semicolon, Comma, Space, Other, OtherChar, FieldInfo, TextVisualLayout, DecimalSeparator, ThousandsSeparator, TrailingMinusNumbers, Local)

Der Tabelle 95 können Sie die Erläuterung der einzelnen VBA-Schlüsselwörter entnehmen.

VBA-Schlüsselwort	Beschreibung
Filename	Pfad und Dateiname.
Origin	Herkunft. Mögliche Werte: xlMacintosh, xlWindows und xlMSDOS.
StartRow	Die Zeilennummer, ab welcher importiert werden soll.
DataType	Gibt an, wie die Daten getrennt werden sollen. xlDelimited bedeutet, dass die Daten in der Quelle durch ein bestimmtes Zeichen getrennt sind. Das Trennzeichen wird später noch bestimmt. xlFixedWidth bedeutet, dass eine bestimmte Spaltenbreite beim Import angegeben wird. Wenn diese Auswahl getroffen wird, dann verändert sich der Array bei FieldInfo (siehe weiter unten), denn dann muss angegeben werden, ab welchem Zeichen die Daten getrennt werden sollen. Bei dieser Einstellung kann auf die Angabe von Trennzeichen verzichtet werden.
TextQualifier	Texterkennungszeichen. Mögliche Werte: Doppeltes Anführungszeichen xlTextQualifierDoubleQuote, Einfaches Anführungszeichen xlTextQualifierNone und kein Anführungszeichen xlTextQualifierSingleQuote.

Tabelle 95: Einzelne Komponenten zum Textimport

Grundlagen

Allgemein

Datum und Zeit

Formeln

Steuerungselemente

Befehlsliste

Objekte

Diagramme

Ereignisse

User-Formen

Web/Mail

Extern

Gemischtes

Spezial

VBA-Schlüsselwort	Beschreibung
ConsecutiveDelimiter	Aufeinander folgende Trennzeichen sollen als ein Zeichen interpretiert werden.
Tab	Der Tabulator ist das Trennzeichen in der Quelldatei.
Semicolon	Das Semikolon ist das Trennzeichen in der Quelldatei.
Comma	Das Komma ist das Trennzeichen in der Quelldatei.
Space	Das Leerzeichen ist das Trennzeichen in der Quelldatei.
Other	Ein anderes Zeichen dient als Trennzeichen. Das Zeichen wird direkt in der nächsten Codezeile OthersChar hinterlegt.
OthersChar	Hier wird das selbst definierte Trennzeichen hinterlegt.
FieldInfo	Hier werden in einem Array die Arrays (Spalten) zusammengefasst. Wenn die Tabelle vier Spalten enthält, dann werden vier Arrays aufgezeichnet. Innerhalb der Klammer hinter dem Array steht an erster Stelle die Spaltenzahl. Die zweite Zahl steht für das Datenformat der Spalten: 1 = Standard 2 = Text 3 = Datum 4 = Spalten nicht importieren (überspringen)
DecimalSeparator	Dezimal Trennzeichen.
TousandsSeparator	Tausender Trennzeichen.
TrailingMinusNumbers	Nachstehendes Minuszeichen für negative Zahlen.

Tabelle 95: Einzelne Komponenten zum Textimport (Forts.)

418 Export nach Textdatei

Im letzten Rezept haben Sie erfahren, wie die Daten einer Textdatei nach Excel geschrieben werden können. Das Ganze funktioniert auch umgekehrt.

Die Excel-Tabelle, die in unserem Beispiel exportiert werden soll, umfasst vier Spalten. Diese vier Spalten sollen in eine Textdatei geschrieben werden. Die einzelnen Zellen sollen in der Textdatei durch ein Semikolon (;) getrennt werden.

In unserer Prozedur wird nach der Variablendeklaration ein Objekt erzeugt, das auf das Dateisystem zugreift: `CreateObject("Scripting.FileSystemObject")`. Danach wird der Pfad hinterlegt.

Die `Do...While`-Schleife wird so lange ausgeführt, bis in der Spalte A ein leerer Eintrag gefunden wird. Wir gehen hier davon aus, dass dann das Tabellenende erreicht wurde. Innerhalb der Schleife wird Zeile für Zeile in die Textdatei geschrieben: `WriteLine`. Das geschieht, indem jeweils die vier Zellen pro Zeile miteinander verbunden werden. Zwischen jede der Zellen wird ein Semikolon eingefügt.

Damit zu erkennen ist, dass der Export beendet ist, wird am Ende der Prozedur eine entsprechende Meldung ausgegeben.

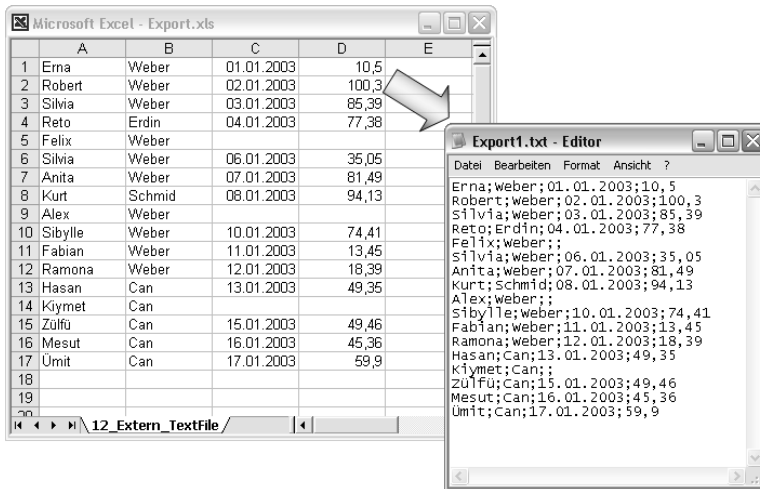


Abbildung 348: Daten aus einer Excel-Tabelle in eine Textdatei exportieren

```
'=====
' Auf CD      Buchdaten\Beispiele\12_Extern\
' Dateiname  12_02_TextFile.xls
' Modul      mdl_05_ExportTextFile
'=====
```

```
Sub ExportTextFile1()
    Dim objFSO As Object
    Dim objText As Object
    Dim i As Integer

    Set objFSO = CreateObject("Scripting.FileSystemObject")
    Set objText = objFSO.CreateTextFile _
        ("C:\Codebook_12_Test\Export1.txt")

    i = 1
    Do While Cells(i, 1) <> ""
        objText.WriteLine _
            Cells(i, 1) & ";" & Cells(i, 2) & ";" & _
            Cells(i, 3) & ";" & Cells(i, 4)
        i = i + 1
    Loop

    MsgBox "Die Datei wurde erfolgreich erzeugt"

    Set objFSO = Nothing
    Set objText = Nothing
End Sub
```

Wir werden nun die vorangegangene Prozedur erweitern. Es soll die Möglichkeit bestehen, dass unmittelbar nach dem Aufrufen der Prozedur in ein Eingabefeld das Listentrennzeichen eingegeben werden kann. Das Ganze wird damit etwas flexibler. Die Prozedur wird entsprechend um eine `InputBox` ergänzt.

Grund-
lagenAllge-
meinDatu
Zeit

tungen

Steuer
elemenBefehl
leisten

Objekt

Dia-
grammEreig-
nisseUser-
FormWeb/
Mail

Extern

Ge-
misch
tes

Specia

Da die Anzahl der Spalten in der Excel-Tabelle variieren kann, erweitern wir die Prozedur um eine `For...Next`-Schleife, die die vorhandenen Spalten zählt (`Columns.Count`) und nacheinander arbeitet.

Es sollen zudem in der Textdatei keine aufeinander folgenden Listentrennzeichen auftauchen. Um dies zu unterdrücken, erweitern wir die `For...Next`-Schleife um eine `If`-Entscheidung, die prüft, ob die Zelle leer ist. Falls die Zelle nicht leer ist `<> ""`, wird deren Inhalt an die Variable `strValue` übergeben. Dieser Variablen wird danach das Listentrennzeichen angefügt. Die Variable wird so lange erweitert, bis alle Spalten einer Zeile abgearbeitet sind. Die Ausgangslage und das Ergebnis können Sie der Abbildung 349 entnehmen.

Damit in der Textdatei am Ende der Zeile kein Listentrennzeichen angezeigt wird, wird der Inhalt der Variablen um das letzte Zeichen gekürzt (`Left(strValue, Len(strValue) - 1)`). Diese Befehlszeile schreibt die gesamte Länge `Len` der Variablen, minus ein Zeichen `Len(strValue) - 1`. Der linke restliche Teil der Zeichenkette bleibt übrig. Der Inhalt der Variablen kann nun in die Textdatei geschrieben werden (`WriteLine`).

Danach beginnt das ganze Spiel von vorne. Es wird die nächste Zeile aufbereitet. Am Ende der Prozedur wird eine Meldung auf dem Bildschirm angezeigt, die darauf hinweist, dass die Prozedur erfolgreich beendet wurde.

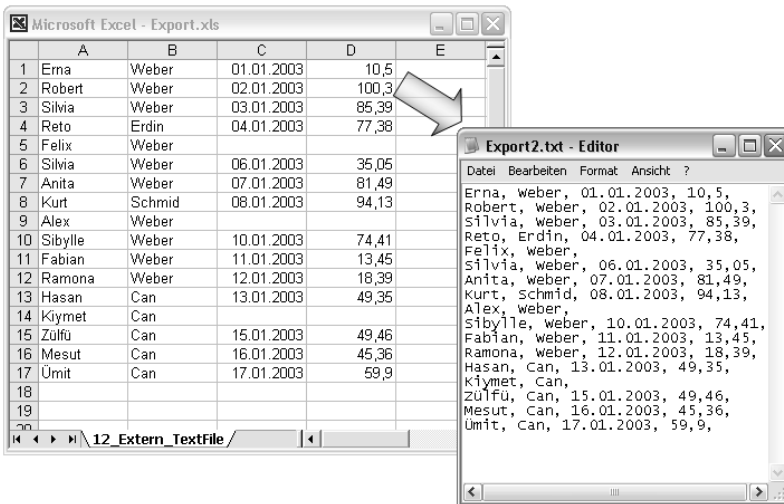


Abbildung 349: Ohne doppelte Listentrennzeichen in Textdatei schreiben

```

' =====
' Auf CD      Buchdaten\Beispiele\12_Extern\
' Dateiname  12_02_TextFile.xls
' Modul      mdl_05_ExportTextFile
' =====

```

```

Sub ExportTextFile2()
    Dim objFSO As Object
    Dim objText As Object
    Dim lngRow As Long
    Dim intCol As Integer

```

```
Dim strInput As String
Dim varValue As Variant

Set objFSO = CreateObject("Scripting.FileSystemObject")
Set objText = objFSO.CreateTextFile _
    ("C:\Codebook_12_Test\Export2.txt")

strInput = Application.InputBox _
    ("Bitte ein Trennzeichen eingeben", _
    "Listentrennzeichen", ";")

lngRow = 1
Do While Cells(lngRow, 1) <> ""
    varValue = ""

    For intCol = 1 To ActiveSheet.UsedRange.Columns.Count
        If Cells(lngRow, intCol) <> "" Then
            varValue = varValue & Cells(lngRow, intCol) & _
                strInput
        End If
    Next intCol

    objText.WriteLine Left(varValue, Len(varValue) - 1)
    lngRow = lngRow + 1
Loop

MsgBox "Die Datei wurde erfolgreich erzeugt"

Set objFSO = Nothing
Set objText = Nothing
End Sub
```

Grundlagen

Allgemein

Datum
Zeit

tungen

Steuerelemente

Befehle
leisten

Objekt

Diagramm

Ereignisse

User-Form

Web/
Mail

Extern

Gemischtes

Spezial

419 Import aus Word

Die nächsten Rezepte widmen sich der Kommunikation zwischen Word und Excel. Es können sowohl Daten aus Word ausgelesen und nach Excel geschrieben werden, als auch umgekehrt.

Die Ausgangslage unseres ersten Beispiels: Es besteht ein Word-Dokument mit dem Namen *MyWordDoc1.doc*. Das Dokument befindet sich im Pfad *C:\Codebook_12_Test*, sofern Sie den entsprechenden Ordner, wie zu Beginn der Kategorie beschrieben, auf Ihr Laufwerk *C:* kopiert haben. Die darin enthaltenen Daten befinden sich in einer Tabelle, was den Import vereinfacht. Wir werden uns direkt im Anschluss ein Beispiel ansehen, wo die Daten im Word-Dokument durch ein Semikolon getrennt sind.

Zu Beginn der Prozedur, direkt nach der Variablendeklaration, wird ein Word-Objekt erzeugt (`CreateObject("Word.Application")`). Danach kann mit den Befehlen von Word gearbeitet werden. Innerhalb einer `With`-Anweisung wird das Word-Objekt verwendet. Die darin enthaltenen Befehle stammen aus Word.

Tipp

Wenn Sie sich mit den VBA-Befehlen von Word nicht auskennen, nehmen Sie am besten den Makro-Rekorder von Word zu Hilfe. Zeichnen Sie innerhalb von Word die gewünschten Schritte auf und bauen Sie diese Befehle in Excel in Ihre Prozedur ein.

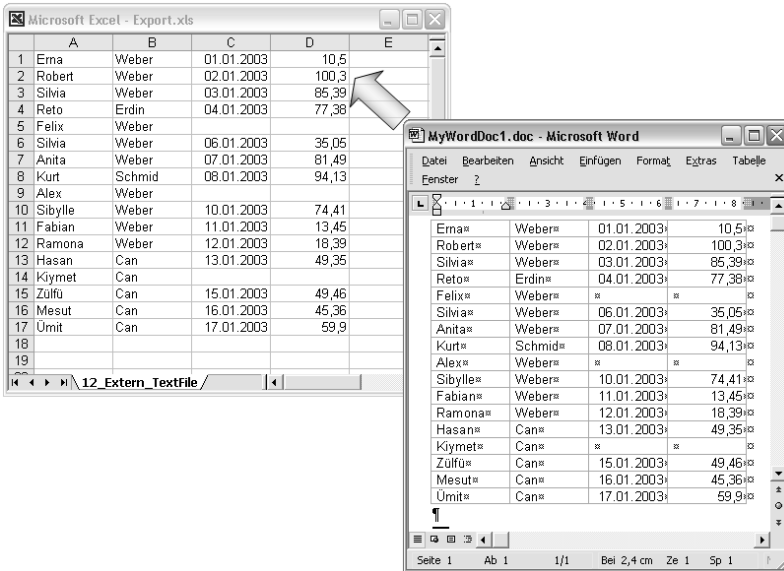


Abbildung 350: Datenimport aus einer Word-Tabelle

In unserem Beispiel wird zuerst der Pfad gewechselt (`ChangeFileOpenDirectory`). Danach wird das Word-Dokument geöffnet. Sie können die Codezeile, die zum Wechseln des Pfades dient, auch weglassen und direkt in der zweiten Zeile den gesamten Pfad inklusive Dateiname eingeben.

Das Word-Dokument wird sichtbar gemacht `Visible = True` und aktiviert `Activate`. Danach wird der gesamte Inhalt der Word-Datei kopiert `Selection.WholeStory`. Es wird der markierte Bereich kopiert `Selection.Copy`.

Um die Daten in eine Excel-Mappe zu übertragen, wird in unserer Datei zuerst eine neue Mappe angelegt `Workbooks.Add`. Dann werden die Daten, die sich noch in der Zwischenablage befinden, eingefügt `ActiveSheet.Paste`.

Es ist sehr wichtig, dass Word nach Beendigung der Prozedur wieder geschlossen wird `objWord.Application.Quit`. Ansonsten besteht die Möglichkeit, dass das Dokument im Hintergrund geöffnet bleibt. Bei erneutem Ausführen des Codes würde das bedeuten, dass das Dokument nur noch mit Schreibschutz geöffnet werden kann.

Am Ende der Prozedur wird der Verweis auf das Word-Objekt wieder gelöscht.

```
'=====
' Auf CD      Buchdaten\Beispiele\12_Extern\
' Dateiname  12_03_Word.xls
' Modul      mdl_01_ImportFromWord1
'=====
```

```
Sub ImportFromWord1()
    Dim objWord As Object

    Set objWord = CreateObject("Word.Application")

    With objWord
```

```
.ChangeFileOpenDirectory "C:\Codebook_12_Test\"
.Documents.Open Filename:="MyWordDoc1.doc"
.Visible = True
.Activate
.Selection.WholeStory
.Selection.Copy

End With

Workbooks.Add
ActiveSheet.Paste

objWord.Application.Quit

Set objWord = Nothing
End Sub
```

Unser zweites Beispiel unterscheidet sich vom ersten insofern, als dass die Daten der Word-Datei sich nicht in einer Word-Tabelle befinden. Die Daten liegen getrennt durch ein Semikolon vor.

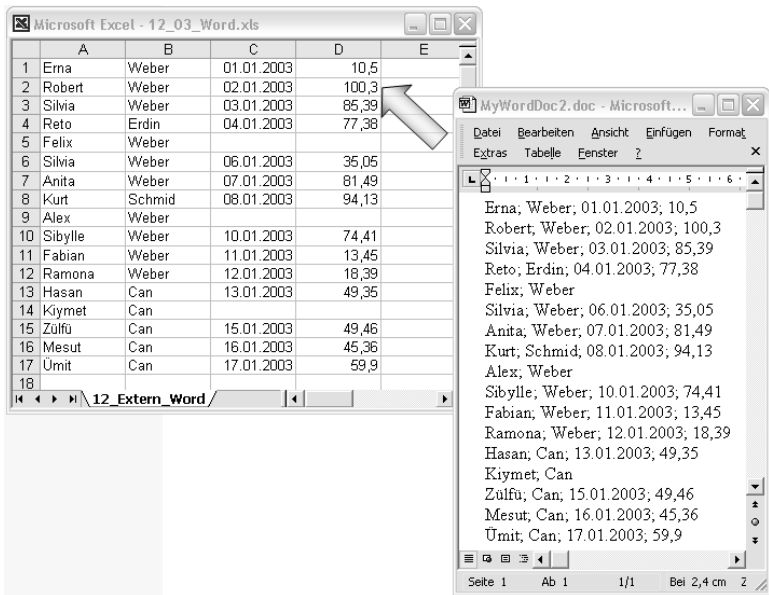


Abbildung 351: Durch Semikolon getrennte Daten aus Word importieren

Die Vorgehensweise im nachfolgenden Code ist bis auf die Anweisung Paste dieselbe wie im letzten Codebeispiel. Damit die Daten jedoch auf vier Spalten verteilt werden, müssen sie getrennt werden. Da die Daten nach dem Import markiert sind, können wir uns direkt auf die Markierung beziehen. Der Text wird nun getrennt. Da die Daten in der Ursprungsform durch ein Semikolon getrennt sind, muss in der Prozedur entsprechend darauf hingewiesen werden (Semicolon:=True). Die Daten für die vier Spalten werden mittels Array verteilt. In Excel können Daten auch manuell getrennt werden. Markieren Sie zuerst die Spalte und klicken Sie dann auf den Menüpunkt DATEN | TEXT IN SPALTEN. Ein Assistent führt Sie durch die einzelnen Schritte. Mehr Informationen zum Datentrennen per VBA erfahren Sie in dieser Kategorie im Rezept 418.

- Grundlagen
- Allgemein
- Datum und Zeit
- Formeln
- Diagramme
- Ereignisse
- User-Formen
- Web/Mail
- Extern
- Gemischtes
- Spezial

```
'=====
' Auf CD      Buchdaten\Beispiele\12_Extern\
' Dateiname   12_03_Word.xls
' Modul       mdl_02_ImportFromWord2
'=====

Sub ImportFromWord2()
    Dim objWord As Object

    Set objWord = CreateObject("Word.Application")

    With objWord
        .ChangeFileOpenDirectory "C:\Codebook_12_Test\"
        .Documents.Open Filename:="MyWordDoc2.doc"
        .Visible = True
        .Activate
        .Selection.WholeStory
        .Selection.Copy
    End With

    Workbooks.Add
    ActiveSheet.Paste
    Selection.TextToColumns _
        Semicolon:=True, _
        FieldInfo:=Array(Array(1, 1), _
            Array(2, 1), _
            Array(3, 1), _
            Array(4, 1))

    objWord.Application.Quit

    Set objWord = Nothing
End Sub
```

420 Export von Excel nach Word

In den vorangegangenen beiden Beispielen wurde der Text aus Word-Dokumenten in eine Excel-Tabelle geschrieben. Das Ganze funktioniert auch umgekehrt.

Die Ausgangslage für unser nächstes Beispiel ist eine Excel-Tabelle mit vier Spalten. Der Inhalt dieser Zellen soll in ein Word-Dokument geschrieben werden. Das Listentrennzeichen ist ein Bindestrich. Vor und nach dem Bindestrich befindet sich ein Leerzeichen. Die Daten sollen in Word in eine Zeile geschrieben werden, so dass ein einziger Absatz entsteht.

Direkt nach der Variablen-Deklaration wird ein Word-Objekt erzeugt. Dieses wird der Variablen `objWord` zugewiesen. In einer `For...Next`-Schleife wird der Bereich A1:D17 abgearbeitet. In der `If...Then...Else`-Entscheidung wird geprüft, ob eine Zelle leer ist. Wenn nicht, wird deren Inhalt der Variablen `strValue` hinzugefügt, gefolgt vom Listentrennzeichen (-). Die Variable wird so nach und nach erweitert.

Wenn die Variable `strValue` komplett mit den gewünschten Werten gefüllt ist, wird ein Word-Dokument erzeugt: `Documents.Add`. Damit das Dokument sichtbar (`Visible=True`) und aktiviert (`Activate`) ist, werden zuvor die entsprechenden Befehle eingegeben. Nachdem das Dokument erzeugt wurde, kann der Inhalt der Variablen an Word übergeben werden: `Selection.Text = ...`

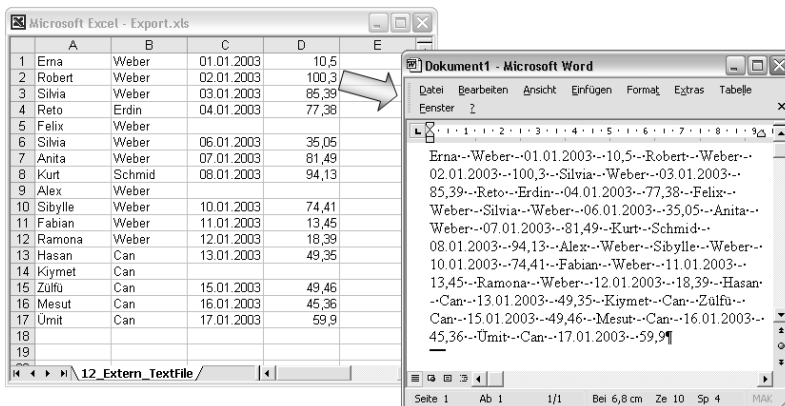


Abbildung 352: Text in einem Absatz nach Word übertragen

Damit vor dem gesamten Textblock kein Listentrennzeichen angezeigt wird, wird dieses `Right(strValue, Len(strValue) - 3)`. Dabei wird die Länge der Variablen ermittelt. Diese wird um drei Stellen gekürzt (Leerzeichen, Bindestrich, Leerzeichen). Der übrige rechte Teil wird, ohne die drei Stellen des Listentrennzeichens, ins Word-Dokument geschrieben.

```
' =====
' Auf CD      Buchdaten\Beispiele\12_Extern\
' Dateiname   12_03_Word.xls
' Modul       mdl_03_ExportToWord1
' =====
```

```
Sub ExportToWord1()
    Dim objWord As Object
    Dim c As Range
    Dim strValue As String

    Set objWord = CreateObject("Word.Application")

    For Each c In Range("A1:D17")
        If c <> "" Then
            strValue = strValue & " - " & c
        End If
    Next c

    With objWord
        .Visible = True
        .Activate
        .Documents.Add
        .Selection.Text = Right(strValue, Len(strValue) - 3)
    End With

    Set objWord = Nothing
End Sub
```

Grund-
lagenAllge-
meinDatu
Zeit

tungen

Steuer
elemenBefehl
leisten

Objekt

Dia-
grammEreig-
nisseUser-
FormWeb/
Mail

Extern

Ge-
misch
tes

Specia

In unserem nächsten Code ist die Ausgangslage dieselbe Excel-Tabelle. Diesmal soll jedoch nicht ein fester Bereich übertragen werden, sondern der benutzte Bereich. Dabei arbeiten wir mit zwei For...Next-Schleifen, die ineinander verschachtelt sind. In der ersten Schleife werden alle Zeilen Rows gezählt und abgearbeitet. In dieser Schleife drin werden alle Spalten Columns gezählt und abgearbeitet. Das bedeutet, dass Zelle für Zelle von links nach rechts gelesen wird. Innerhalb der inneren Schleife wird die Variable strValue gefüllt sowie ein Listentrennzeichen, ein Semikolon und ein Leerzeichen nach jeder Zeichenkette angehängt.

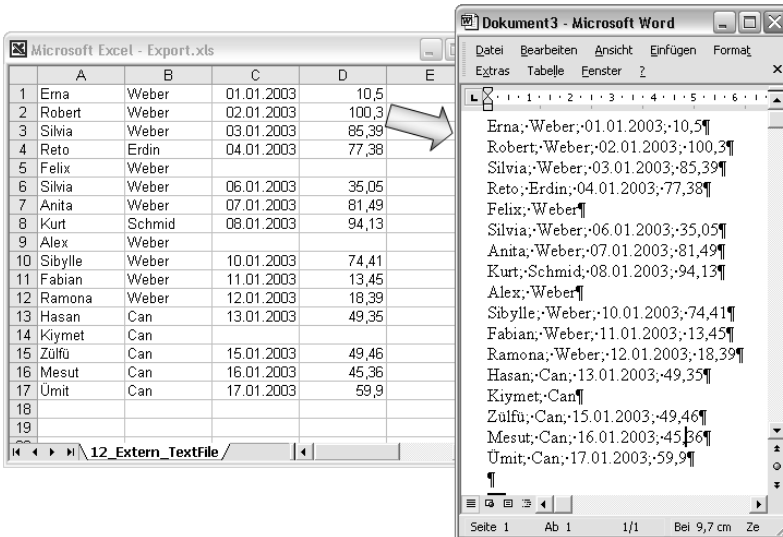


Abbildung 353: Zeilen in einzelnen Absätzen nach Word übertragen

Damit ein Zeilenumbruch am Ende jeder Zeile entsteht, wird jeweils nach der inneren Schleife die Variable neu aufbereitet. Dabei wird die Variable um die letzten zwei Stellen gekürzt, damit am Ende der Zeile kein Listentrennzeichen erscheint. Dann wird der Zeilenumbruch vbCrLf hinzugefügt.

Der Rest der Prozedur, also das Schreiben nach Word, ist gleich wie im vorangegangenen Beispiel, nur dass hier die Variable nicht mehr angepasst werden muss. Dies geschah ja bereits Zeile für Zeile jeweils am Ende der ersten Schleife.

```
' =====
' Auf CD      Buchdaten\Beispiele\12_Extern\
' Dateiname  12_03_Word.xls
' Modul      mdl_04_ExportToWord2
' =====
```

```
Sub ExportToWord2()
    Dim objWord As Object
    Dim dblRow As Double
    Dim intCol As Integer
    Dim strValue As String

    Set objWord = CreateObject("Word.Application")
```



```
For dblRow = 1 To ActiveSheet.UsedRange.Rows.Count
    For intCol = 1 To ActiveSheet.UsedRange.Columns.Count
        If Cells(dblRow, intCol) <> "" Then
            strValue = strValue & Cells(dblRow, intCol) & "; "
        End If
    Next intCol
    strValue = Left(strValue, Len(strValue) - 2) & vbCrLf
Next dblRow

With objWord
    .Visible = True
    .Activate
    .Documents.Add
    .Selection.Text = strValue
End With

Set objWord = Nothing
End Sub
```

421 Export von Zwischenablage nach Word

Bis jetzt haben Sie zwei Rezepte zum Datenübertragen von Excel nach Word erfahren. Dabei wurden jeweils die einzelnen Zellen ausgelesen, aufbereitet und dann nach Word geschrieben. Sie können die Daten jedoch auch so nach Word übertragen, dass dabei die Formatierungen von Excel erhalten bleiben. Damit das deutlich wird, färben wir die einzelnen Spalten ein und vergeben einen Rahmen.

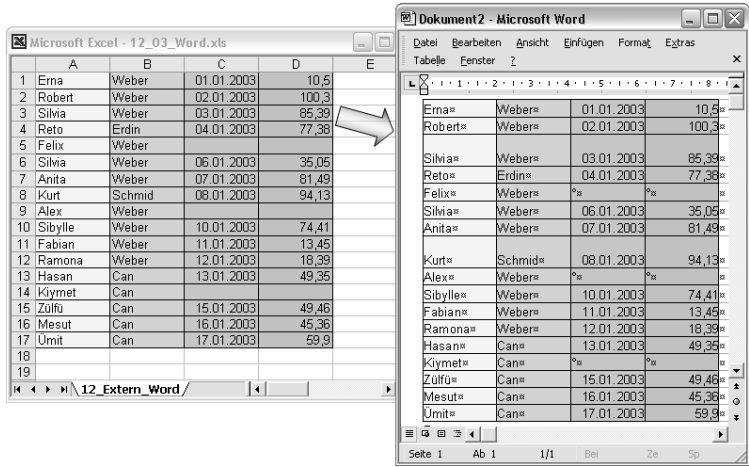


Abbildung 354: Übertrag nach Word via Zwischenablage

Wir bedienen uns dabei der Zwischenablage von Office. In unserem Beispiel wird im aktiven Tabellenblatt der benutzte Bereich kopiert: `ActiveSheet.UsedRange.Copy`. Die Daten gelangen damit in die Zwischenablage. Danach wird ein Word-Dokument erzeugt. Darin wird der Inhalt der Zwischenablage eingefügt (`Selection.Paste`). Mittels `Selection.MoveRight` kann eine bestehende Markierung aufgehoben werden. Dies wird vor allem bei Shapes, also Objekten wie zum Beispiel Grafiken, angewendet. Zu guter Letzt wird der Kopiermodus verlassen: `CutCopyMode = False`.

Grundlagen

Allgemein

Datum
Zeit

ungen

Steuer
elemente

Befehl
leisten

Objekt

Diagramm

Ereignisse

User-
Form

Web/
Mail

Extern

Gemischtes

Spezial

```
'=====
' Auf CD      Buchdaten\Beispiele\12_Extern\
' Dateiname  12_03_Word.xls
' Modul      mdl_05_ExportToWordClipboard
'=====

Sub ExportToWordClipboard()
    Dim objWord As Object

    Set objWord = CreateObject("Word.Application")

    ActiveSheet.UsedRange.Copy

    With objWord
        .Visible = True
        .Activate
        .Documents.Add
        .Selection.Paste
        .Selection.MoveRight
    End With

    Application.CutCopyMode = False

    Set objWord = Nothing
End Sub
```

422 Import aus PowerPoint

Sie können sowohl Daten von PowerPoint nach Excel übertragen als auch umgekehrt. Wenden wir uns zuerst dem Import von Daten aus PowerPoint nach Excel zu.

Unsere Ausgangslage ist eine bestehende PowerPoint-Präsentation, die zwei Folien umfasst. Die Datei dazu lautet *Orchideen.ppt*. Die Präsentation befindet sich im Pfad *C:\Codebook_12_Test*, sofern Sie den entsprechenden Ordner, wie zu Beginn der Kategorie beschrieben, auf Ihr Laufwerk *C:* kopiert haben.

Voraussetzung beim Auslesen von Elementen aus PowerPoint ist, dass Sie mit dem Umgang dieser Anwendung ein wenig vertraut sind. Unsere Präsentation ist sehr einfach aufgebaut. Auf jeder der beiden Folien befinden sich ein Text sowie ein Bild. In der Regel befinden sich solche Elemente je in einem Shape, also in einer Form. Jede dieser Formen kann per VBA über das Objekt *Shape* angesprochen werden. Auf jeder unserer Folien sind somit zwei *Shapes* vorhanden. Da das Bild jeweils an zweiter Stelle auf der Folie steht, wird es über den Index 2 angesprochen: *Shapes(2)*.

Der Befehl, um die Folie selbst anzusprechen, lautet *Slide*. In der Hierarchie von PowerPoint entspricht das *Slide* einem Tabellenblatt in Excel *Worksheet*. Wenn Sie mit Excel-VBA vertraut sind, werden Sie noch mehr Verwandtschaften zwischen dem VBA von PowerPoint und Excel erkennen. Die Hierarchie sowie auch die Objekte sind ähnlich. In Excel zum Beispiel lautet das Objekt für die aktive Mappe *ActiveWorkbook*, in PowerPoint nennt sich die aktive Präsentation *ActivePresentation*. Es würde zu weit führen, hier im Detail auf die Objekte, Eigenschaften und Methoden von PowerPoint einzugehen. Denken Sie daran, dass auch in PowerPoint ein Makro-Rekorder zur Verfügung steht, mit Hilfe dessen Sie Objekte, Methoden und Eigenschaften automatisch ermitteln können.

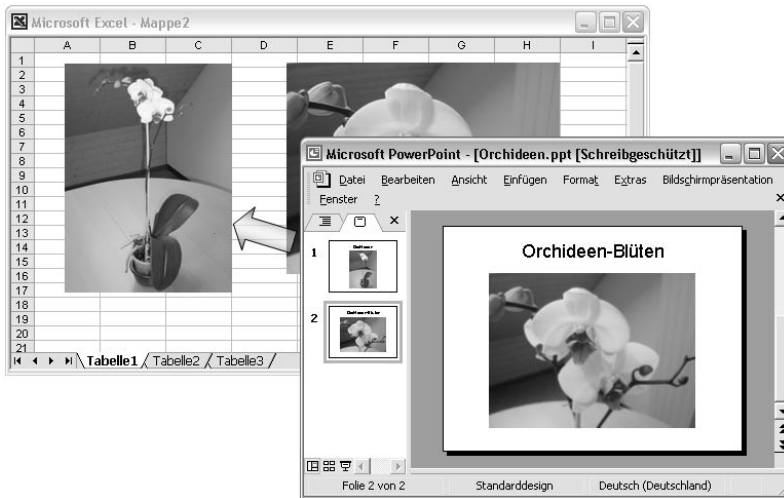


Abbildung 355: Bilder von PowerPoint nach Excel kopieren

In unserem Beispielcode wird zuerst ein PowerPoint-Objekt erzeugt: `CreateObject("Powerpoint.Application")`. Das Objekt wird für die weitere Verwendung der Variablen `objPPT` zugewiesen. Direkt nach der Variablendeklaration und der Referenzierung wird die bestehende Präsentation geöffnet: `Presentations.Open`. Danach wird in Excel ein neues Tabellenblatt eingefügt (`Workbooks.Add`). In dieses Tabellenblatt, welches ja nun das Aktive ist, sollen die Bilder eingefügt werden. Jedes der Bilder wird somit kopiert (`Copy`) und dann in Excel eingefügt (`Paste`).

Sie können innerhalb der Prozedur zudem veranlassen, dass die Größe der einzelnen Bilder reduziert wird. Geben Sie dabei den gewünschten Faktor ein. Verwenden Sie dazu die Methoden `ScaleWidth` für die Breite und `ScaleHeight` für die Höhe. Der Faktor bezieht sich dabei relativ auf die aktuelle Größe des Bildes.

```
'=====
' Auf CD      Buchdaten\Beispiele\12_Extern\
' Dateiname  12_04_PowerPoint.xls
' Modul      mdl_01_ImportFromPowerPoint
'=====
```

```
Sub ImportFromPowerPoint()
    Dim objPPT As Object

    Set objPPT = CreateObject("Powerpoint.Application")

    With objPPT
        .Activate
        .Presentations.Open "C:\Codebook_12_Test\Orchideen.ppt"
    End With

    ' Neue Excel-Mappe anlegen
    Workbooks.Add

    ' Bilder aus Folie 1 und 2 kopieren und in Excel einfügen
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

```
objPPT.ActivePresentation.Slides(1).Shapes(2).Copy
ActiveSheet.Paste Destination:=ActiveSheet.[A1]
objPPT.ActivePresentation.Slides(2).Shapes(2).Copy
ActiveSheet.Paste Destination:=ActiveSheet.[D1]
```

```
' Größe der beiden Bilder reduzieren
With ActiveSheet.Shapes(1)
    .ScaleWidth 0.5, msoFalse, msoScaleFromTopLeft
    .ScaleHeight 0.5, msoFalse, msoScaleFromTopLeft
End With
```

```
With ActiveSheet.Shapes(2)
    .ScaleWidth 0.5, msoFalse, msoScaleFromTopLeft
    .ScaleHeight 0.5, msoFalse, msoScaleFromTopLeft
End With
```

```
Set objPPT = Nothing
End Sub
```

423 Export nach PowerPoint

Wir benötigen hierzu den Verweis auf die Bibliothek von PowerPoint. Wechseln Sie in den VBA-Editor von Excel, wenn Sie sich nicht bereits darin befinden, und klicken Sie auf den Menüpunkt EXTRAS | VERWEISE. Aktivieren Sie das Kontrollkästchen vor MICROSOFT POWERPOINT X.X OBJECT LIBRARY.

Die Ausgangslage für unser Beispiel ist eine Excel-Datei mit einem Diagramm. Eine PowerPoint-Präsentation mit dem Namen *MyPres.ppt* besteht bereits. Diese enthält lediglich eine einzelne leere Folie. Die Präsentation befindet sich im Pfad *C:\Codebook_12_Test*, sofern Sie den entsprechenden Ordner, wie zu Beginn der Kategorie beschrieben, auf Ihr Laufwerk C:\ kopiert haben.

Auf die bestehende leere Folie soll der Tabellenbereich A1:C3 übertragen werden. Dabei werden wir nicht die Daten als solche übertragen, sondern der Bereich wird als Grafik in die Präsentation übertragen. Die zweite Folie muss erst erzeugt werden. Darauf wird das Diagramm platziert.

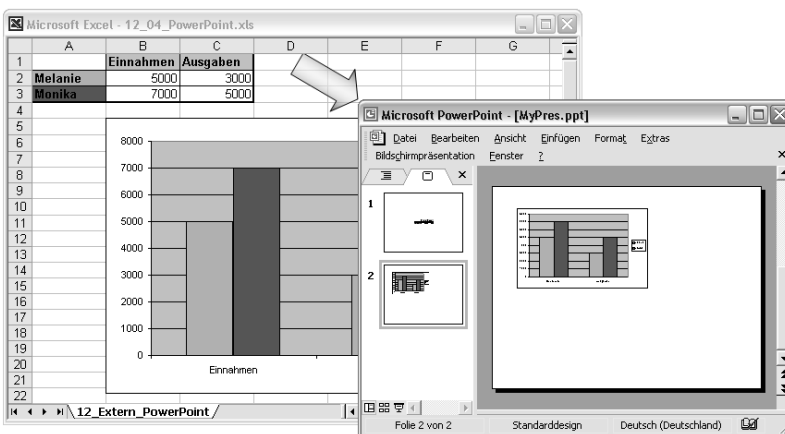


Abbildung 356: Übertrag von Excel nach PowerPoint

Sie können auch ohne VBA einen Bereich in Excel »fotografieren« und das Bild an einer anderen Stelle wieder einfügen. Markieren Sie dazu den Bereich, aus welchem Sie ein Bild erzeugen möchten. Kopieren Sie diesen Bereich. Selektieren Sie den Zielbereich. Klicken Sie mit gedrückter -Taste auf den Menüpunkt EINFÜGEN | BILD EINFÜGEN. Wie Sie sehen können, hat sich der Menüpunkt EINFÜGEN durch das Drücken der -Taste in BILD EINFÜGEN umgewandelt. Der markierte Bereich ist nun als Grafik verfügbar und kann beliebig verschoben, vergrößert und verkleinert werden.

Um per VBA aus einem Bereich ein Bild zu erzeugen, verwenden Sie den Befehl `CopyPicture`. Die Syntax zu diesem Befehl lautet:

`Ausdruck.CopyPicture(Appearance, Format)`

Sie können dem Befehl `CopyPicture` zwei Werte mitliefern. Jedem der Werte können zudem zwei Konstanten übergeben werden. Sie können diese der Tabelle 96 entnehmen.

Wert	Konstante	Beschreibung
Appearance	xlPrinter	Das Bild wird so erzeugt, wie es schlussendlich auf dem Drucker ausgegeben wird.
	xlScreen (Standard)	Das Bild wird so erzeugt, dass es gemäß der Bildschirm- ausgabe angezeigt wird.
Format	xlBitmap	Ausgabe erfolgt als Bitmap.
	xlPicture (Standard)	Ausgabe erfolgt als Bild.

Tabelle 96: Werte und Konstanten zu `CopyPicture`

In unserem Beispielcode wird zuerst mittels `CopyPicture` ein Bild aus dem Bereich A1:C3 erzeugt. Danach wird die bestehende PowerPoint-Präsentation geöffnet: `Presentations.Open`. In die erste Folie `Slide(1)` wird der Inhalt, also das Bild, das sich in der Zwischenablage befindet, eingefügt (`Paste`). Direkt danach wird das Bild positioniert. Mittels `IncrementLeft 300` wird das Bild um 300 Punkte vom linken Bildrand her nach rechts verschoben. Mittels `IncrementTop 200` wird das Bild um 200 Punkte vom oberen Rand her nach unten verschoben. Mehr zum Thema Bilder erfahren Sie in der Kategorie »Objekte«.

In einem zweiten Schritt wird das Excel-Diagramm kopiert: `Shapes(1).Copy`. Da sich in unserer PowerPoint-Präsentation nur eine Folie befindet, das Diagramm jedoch auch einer anderen eingefügt werden soll, muss zuerst an zweiter Stelle eine neue, leere Folie eingefügt werden. Dazu verwenden wir den Befehl `Slides.Add 2, pptLayoutBlank`. Das Diagramm kann nun auf dieser zweiten Folie eingefügt werden (`Slides(2).Shapes.Paste`).

Die Präsentation ist somit vollständig und kann gestartet werden: `SlideShowSettings.Run`. Am Ende der Prozedur wird der Verweis auf das Objekt wieder freigegeben.

```
' =====
' Auf CD      Buchdaten\Beispiele\12_Extern\
' Dateiname  12_04_PowerPoint.xls
' Modul      mdl_02_ExportToPowerPoint
' =====

' Verweis auf Microsoft PowerPoint x.x Object Library
' muss gesetzt sein
```

Grundlagen
Allgemein
Datum Zeit
tungen
Steuer elemente
Befehl leisten
Objekt
Diagramm
Ereignisse
User- Form
Web/ Mail
Extern
Ge- misch- tes
Specia

```

Sub ExportToPowerPoint()
    Dim objPPT As Object
    Set objPPT = CreateObject("Powerpoint.Application")

    ' Excel-Bereich fotografieren
    Sheets(1).Range("A1:C3").CopyPicture _
        Appearance:=xlScreen, _
        Format:=xlPicture

    With objPPT
        .Activate
        .Presentations.Open "C:\Codebook_12_Test\MyPres.ppt"

        ' Fotografierten Bereich in bestehende Folie einfügen
        .ActivePresentation.Slides(1).Shapes.Paste

        ' Bild positionieren
        With .ActivePresentation.Slides(1).Shapes(1)
            .IncrementLeft 300
            .IncrementTop 200
        End With

        ' Diagramm aus Excel heraus kopieren
        ActiveSheet.Shapes(1).Copy

        ' Neue Folie anlegen, Diagramm einfügen
        With .ActivePresentation
            .Slides.Add 2, ppLayoutBlank
            .Slides(2).Shapes.Paste
        End With

        ' PowerPoint-Präsentation starten
        .ActivePresentation.SlideShowSettings.Run
    End With

    Set objPPT = Nothing
End Sub

```

424 Outlook-Kontakte auslesen

Unglücklicherweise stellt Outlook immer noch keinen Makro-Rekorder zur Verfügung. Man muss sich mit der Online-Hilfe oder mit dem Internet behelfen. Eine gute Webseite zu diesem Thema ist unter folgendem URL zu finden: <http://www.slipstick.com/dev/vb.htm>. Leider ist die Seite in Englisch gehalten. Auf dieser Seite sind VBA-Prozeduren und VB-Scripts zu Outlook zu finden.

Um Daten aus Outlook auslesen zu können, muss der Verweis auf die Bibliothek von Outlook gesetzt sein. Klicken Sie in der VBA-Umgebung auf den Menüpunkt EXTRAS | VERWEISE. Aktivieren Sie das Kontrollkästchen MICROSOFT OUTLOOK X.X OBJECT LIBRARY.

In unserem nachfolgenden Beispiel werden wir Daten aus den Kontakten auslesen. Es stehen Ihnen in Outlook XP insgesamt 85 Felder zu den Kontakten zur Verfügung. Das Beispiel zeigt Ihnen die Möglichkeit auf, einige davon auszulesen.

MAPI (Messeging Application Programming Interface) ist das Herz der Nachrichten-Programme von Microsoft. Wir werden es hier verwenden, um den Kontakt-Ordner auszulesen. Es wird ebenfalls benötigt, um Daten aus den anderen Outlook-Ordnern auszulesen.

Zuerst wird ein Outlook-Objekt erzeugt. Dieses wird der Variablen `objOLApp` zugewiesen. Um auf einen der Standardordner von Outlook zugreifen zu können, verwenden wir das Objekt `GetNameSpace("MAPI")`. Dieses weisen wir der Variablen `objNameSpace` zu. Wie Sie dem Code entnehmen können, wurde noch eine dritte Outlook-Variable deklariert, und zwar die Variable `objContItem`. Diese müssen wir als `Object` deklarieren und nicht als `Outlook.ContactItem`. Der Grund liegt darin, dass in einem Kontakt-Ordner nicht nur Kontakte, sondern auch Verteilerlisten vorhanden sein können. Nur wenn die Variable `objContItem` als `Object` deklariert wird, können sowohl Kontakte *als auch* Verteilerlisten ausgelesen werden. Die Variable `i` dient als Zähler. Sie wird vor dem Ende der `For`-Schleife jeweils um den Wert 1 erhöht. Der Zähler wird dazu verwendet, die ausgelesenen Outlook-Kontakte Zeile für Zeile untereinander in die Excel-Tabelle zu schreiben.

Um keine Daten in einer bestehenden Tabelle zu überschreiben, legen wir ein neues Tabellenblatt an: `Worksheets.Add`. Danach werden innerhalb einer `For`-Schleife einzelne Punkte des Kontaktordners angesprochen. Der Standardordner für die Kontakte lautet `olFolderContacts`. Um einen Standardordner von Outlook anzusprechen, verwenden wir die Methode `GetDefaultFolder`.

Es stehen Ihnen insgesamt elf Standardordner zur Verfügung. Welche das sind, können Sie der Tabelle 97 entnehmen:

Konstanten der Methode: <code>GetDefaultFolder</code>	Beschreibung
<code>olFolderCalendar</code>	Kalender
<code>olFolderContacts</code>	Kontakte
<code>olFolderDeletedItems</code>	Gelöschte Objekte
<code>olFolderDrafts</code>	Entwürfe
<code>olFolderInbox</code>	Posteingang
<code>olFolderJournal</code>	Journal
<code>olFolderNotes</code>	Notizen
<code>olFolderOutbox</code>	Postausgang
<code>olFolderSentMail</code>	Gesendete Objekte
<code>olFolderTasks</code>	Aufgaben
<code>olPublicAllPublicFolders</code>	Alle öffentlichen Ordner

Tabelle 97: Outlook-Standardordner

In der `If`-Entscheidung wird geprüft, ob es sich bei dem ausgelesenen Datensatz um einen Kontakt oder eine Verteilerliste handelt. Die ausgelesenen Daten werden in die neu erstellte Excel-Tabelle geschrieben. Es entstehen dabei insgesamt sieben Spalten, denn sieben verschiedene Kontakt-Felder werden ausgelesen. Die Anzahl der Zeilen, die dabei entstehen, hängt von der Anzahl Ihrer Kontakte und Verteilerlisten ab. Um Verteilerlisten in der Excel-Tabelle von Kontakten unterscheiden zu können, weisen wir den entsprechenden Zellen eine graue Hintergrundfarbe zu.

Am Ende der Prozedur wird die Breite der Spalten angepasst: `AutoFit`. Danach werden die Verweise freigegeben.

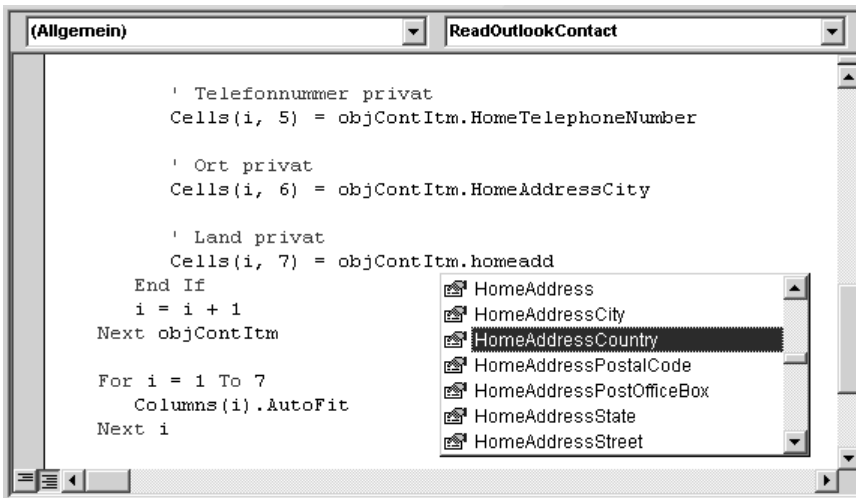


Abbildung 357: Eigenschaften und Methoden zu den Kontakten

Beim Eintippen der Variablen `objContItm` werden in einem DropDown-Feld die einzelnen Eigenschaften und Methoden zu den Kontakten angezeigt (siehe Abbildung 357). Sie können hier wahlweise weitere oder andere Felder auswählen.

```

' =====
' Auf CD      Buchdaten\Beispiele\12_Extern\
' Dateiname   12_05_Outlook.xls
' Modul       mdl_01_ReadOutlookContact
' =====

' Verweis auf Microsoft Outlook x.x Object Library muss gesetzt sein
Sub ReadOutlookContact()
    Dim objOLApp As Outlook.Application
    Dim objNameSpace As Outlook.NameSpace
    Dim objContItem As Object
    Dim i As Integer

    Set objOLApp = CreateObject("Outlook.Application")
    Set objNameSpace = GetNamespaces("MAPI")

    i = 1
    Worksheets.Add
    For Each objContItem In objNameSpace. _
        GetDefaultFolder(olFolderContacts). _
        Items
        If objContItem.Class = olContact Then
            ' Vorname und Nachname
            Cells(i, 1) = objContItem.FirstName & " " & _
                objContItem.LastName

            ' E-Mail-Adresse 1
            Cells(i, 2) = objContItem.EmailAddress
        End If
    Next objContItem
End Sub
  
```



```
' Geschäftsadresse
Cells(i, 3) = objContItem.BusinessAddress

' Telefonnummer Geschäft
Cells(i, 4) = objContItem.BusinessTelephoneNumber

' Telefonnummer privat
Cells(i, 5) = objContItem.HomeTelephoneNumber

' Ort privat
Cells(i, 6) = objContItem.HomeAddressCity

' Land privat
Cells(i, 7) = objContItem.HomeAddressCountry
Else
Cells(i, 1) = "Verteilerliste: " & objContItem.DLName
Cells(i, 1).Interior.ColorIndex = 15
End If
i = i + 1
Next objContItem

' Spaltenbreite automatisch anpassen
Columns("A:G").AutoFit

Set objContItem = Nothing
Set objNameSpace = Nothing
End Sub
```

425 Outlook-Kalender auslesen

In diesem Beispiel werden sämtliche Einträge aus dem Kalender ausgelesen. Einer der wichtigsten Bestandteile des Codes ist auch hier die saubere Deklaration der Variablen. Wir gehen diesmal mit der Deklaration noch etwas weiter. Es werden sämtliche Outlook-Komponenten, die später in der Schleife verwendet werden, dimensioniert und referenziert.

Man muss sich mit den Objekten von Outlook auskennen, um eine entsprechende Prozedur erstellen zu können. Der entscheidende Unterschied zur vorangegangenen Prozedur besteht darin, dass diesmal der Kalender `oIFolderCalendar` ausgelesen wird.

Vor dem Beginn der Schleife wird ein Tabellenblatt eingefügt, damit die Daten in ein leeres Blatt geschrieben werden können. Sie schließen damit die Gefahr aus, dass bestehende Daten überschrieben werden.

Innerhalb der Schleife werden verschiedene Felder aus dem Kalender ausgelesen. Wir haben uns erlaubt, die Beschreibung hier direkt im Code unterzubringen.

Am Ende der Prozedur wird die Spaltenbreite angepasst (`AutoFit`). Danach werden die Verweise freigegeben.

```
'=====
' Auf CD      Buchdaten\Beispiele\12_Extern\
' Dateiname   12_05_Outlook.xls
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

```

' Modul      mdl_02_ReadOutlookCalendar
' =====

Sub ReadOutlookCalendar()
    Dim objOLApp As Outlook.Application
    Dim objNameSpace As Outlook.NameSpace
    Dim objFolder As Outlook.MAPIFolder
    Dim objItem As Outlook.Items
    Dim objAppItem As Outlook.AppointmentItem
    Dim i As Integer

    Set objOLApp = CreateObject("Outlook.Application")
    Set objNameSpace = objOLApp.GetNamespace("MAPI")
    Set objFolder = objNameSpace.GetDefaultFolder(olFolderCalendar)
    Set objItem = objFolder.Items

    i = 1
    Worksheets.Add
    For Each objAppItem In objItem
        ' Betreff
        Cells(i, 1) = objAppItem.Subject

        ' Ort
        Cells(i, 2) = objAppItem.Location

        ' Kategorie
        Cells(i, 3) = objAppItem.Categories

        ' Ganztägig oder Start und Endzeit
        If objAppItem.AllDayEvent = True Then
            Cells(i, 4) = "Ganztägig"
        Else
            Cells(i, 5) = objAppItem.Start
            Cells(i, 6) = objAppItem.End
        End If

        ' Dauer berechnen. Da die Ausgabe in Stunden erfolgt,
        ' wird hier eine entsprechende Umwandlung in Stunden
        ' und Tage vorgenommen
        If objAppItem.Duration >= 1440 Then
            Cells(i, 7) = objAppItem.Duration / 1440 & " Tag(e)"
        ElseIf objAppItem.Duration >= 60 Then
            Cells(i, 7) = objAppItem.Duration / 60 & " Stunde(n)"
        Else
            Cells(i, 7) = objAppItem.Duration & " Stunden"
        End If
        i = i + 1
    Next objAppItem

    ' Spaltenbreite anpassen
    Columns("A:G").AutoFit

    Set objNameSpace = Nothing

```

```
Set objFolder = Nothing
Set objItem = Nothing
End Sub
```

426 Outlook-Posteingang auslesen

Genauso wie Sie die Kontakte oder den Kalender auslesen können, haben Sie per VBA die Möglichkeit, den Inhalt Ihres Posteingangs in eine Excel-Tabelle zu schreiben. Der Standardordner für den Posteingang lautet `olFolderInbox`.

Auch hier wird erst sauber dimensioniert und referenziert. Danach wird ein Tabellenblatt eingefügt, um zu verhindern, dass auf einem bestehenden Blatt Daten überschrieben werden. Innerhalb der `For...Next`-Schleife werden die gewünschten Daten aufbereitet, um diese in das neu angelegte Tabellenblatt zu schreiben. Innerhalb der Schleife wird zudem geprüft, ob es sich bei den Daten um eine Mail (`olMail`) handelt.

Am Ende der Prozedur werden die vier Spalten, welche mit den Outlook-Einträgen gefüllt wurden, in der Breite angepasst (`AutoFit`). Danach werden die Verweise freigegeben.

```
'=====
' Auf CD      Buchdaten\Beispiele\12_Extern\
' Dateiname  12_05_Outlook.xls
' Modul      mdl_03_ReadOutlookInbox
'=====

' Verweis auf Microsoft Outlook x.x Object Library muss gesetzt sein
Sub ReadOutlookInbox()
    Dim objOLApp As Outlook.Application
    Dim objNameSpace As Outlook.NameSpace
    Dim objFolder As Outlook.MAPIFolder
    Dim objItems As Outlook.Items
    Dim objItem As Object
    Dim i As Integer

    Set objOLApp = CreateObject("Outlook.Application")
    Set objNameSpace = objOLApp.GetNamespace("MAPI")
    Set objFolder = objNameSpace.GetDefaultFolder(olFolderInbox)
    Set objItems = objFolder.Items

    Worksheets.Add
    i = 1
    For Each objItem In objItems
        If objItem.Class = olMail Then
            Cells(i, 1) = objItem.SenderName      ' Absender
            Cells(i, 2) = objItem.Subject         ' Betreff
            Cells(i, 3) = objItem.ReceivedTime    ' Erhalten am
            Cells(i, 4) = objItem.Size            ' E-Mail-Größe
        End If
        i = i + 1
    Next

    Columns("A:D").AutoFit
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

```

Set objNameSpace = Nothing
Set objFolder = Nothing
Set objItems = Nothing
End Sub

```

427 Outlook-Aufgaben auslesen

In diesem Rezept werden wir den Inhalt des Ordners AUFGABEN in ein Tabellenblatt schreiben. Zu Beginn der Prozedur wird dimensioniert und referenziert. Wenn Sie sich eine Weile mit Outlook-VBA auseinander gesetzt haben, werden Sie feststellen, dass die Beispiele immer nach demselben Muster entstehen.

Auch in diesem Rezept fügen wir zuerst ein leeres Tabellenblatt ein, damit die Daten in dieses Blatt geschrieben werden können. Wir arbeiten mit einer For...Next-Schleife, die einige der zur Verfügung stehenden Felder ausliest.

Beim Auslesen des Status arbeiten wir mit einer Select...Case-Anweisung. Die zur Verfügung stehenden Werte werden in einen aussagekräftigen Text umgewandelt. Es stehen insgesamt fünf Werte 0-4 zur Verfügung.

```

' =====
' Auf CD      Buchdaten\Beispiele\12_Extern\
' Dateiname   12_05_Outlook.xls
' Modul       mdl_04_ReadOutlookTasks
' =====

' Verweis auf Microsoft Outlook x.x Object Library muss gesetzt sein
Sub ReadOutlookTasks()
    Dim objOLApp As Outlook.Application
    Dim objNameSpace As Outlook.NameSpace
    Dim objFolder As Outlook.MAPIFolder
    Dim objItems As Outlook.Items
    Dim objItem As Object
    Dim i As Integer

    Set objOLApp = CreateObject("Outlook.Application")
    Set objNameSpace = objOLApp.GetNamespace("MAPI")
    Set objFolder = objNameSpace.GetDefaultFolder(olFolderTasks)
    Set objItems = objFolder.Items

    Worksheets.Add
    i = 1
    For Each objItem In objItems
        Cells(i, 1) = objItem.Subject      ' Betreff
        Cells(i, 2) = objItem.DueDate      ' Fällig am
        Cells(i, 3) = objItem.StartDate    ' Startdatum
        Select Case objItem.Status          ' Status
            Case Is = 0
                Cells(i, 4) = "Nicht begonnen"
            Case Is = 1
                Cells(i, 4) = "In Bearbeitung"
            Case Is = 2
                Cells(i, 4) = "Erledigt"

```

```
Case Is = 3
  Cells(i, 4) = "Wartet auf jemand anderen"
Case Is = 4
  Cells(i, 4) = "Zurückgestellt"
End Select
i = i + 1
Next

Columns("A:D").AutoFit

Set objOLApp = Nothing
Set objNameSpace = Nothing
Set objFolder = Nothing
Set objItems = Nothing
End Sub
```

428 Outlook-Aufgaben-Fenster anzeigen

In diesem Rezept wird gezeigt, wie eine neue Aufgabe erstellt werden kann bzw. wie es möglich ist, das Aufgabenfenster zu öffnen.

Nach der Variablendeklaration und Referenzierung wird ein neues Item erzeugt: Add. Jedes Item-Element gehört einer Klasse an. Eine solche Klasse beginnt jeweils mit der Abkürzung *IPM* (Inter Personal Message). Auf diese Einleitung folgt der Klassenname. In unserem Beispiel lautet der Name *Task* für Aufgabe. Für sämtliche Ordner stehen solche Klassen zur Verfügung. Sie können mit diesem Wissen nicht nur eine neue Aufgabe erstellen, sondern z.B. auch das Dialogfenster für einen neuen Kontakt oder einen neuen Termin anzeigen lassen. Der nachfolgenden Tabelle können Sie die verschiedenen Klassen entnehmen.

Klasse	Beschreibung
(»IPM.Appointment«)	Kalender
(»IPM.Contact«)	Kontakt
(»IPM.DistList«)	Verteilerliste
(»IPM.Activity«)	Journaleintrag
(»IPM.Note«)	E-Mail
(»IPM.StickyNote«)	Notiz
(»IPM.Post«)	Diskussion
(»IPM.Task«)	Aufgabe

Tabelle 98: Outlook-Elemente erzeugen

Um das gewünschte Dialogfenster anzeigen zu lassen, verwenden wir den Befehl `Display` (Anzeigen).

```
'=====
' Auf CD      Buchdaten\Beispiele\12_Extern\
' Dateiname  12_05_Outlook.xls
' Modul      mdl_05_CreateOutlookTask
'=====
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

```
' Verweis auf Microsoft Outlook x.x Object Library muss gesetzt sein
Sub DisplayOutlookTask()
    Dim objOLApp As Outlook.Application
    Dim objNameSpace As Outlook.NameSpace
    Dim objFolder As Outlook.MAPIFolder
    Dim objItems As Outlook.Items
    Dim objItem As Object
    Dim i As Integer

    Set objOLApp = CreateObject("Outlook.Application")
    Set objNameSpace = objOLApp.GetNamespace("MAPI")
    Set objFolder = objNameSpace.GetDefaultFolder(olFolderNotes)
    Set objItems = objFolder.Items
    Set objItem = objItems.Add("IPM.Task")

    objItem.Display

    Set objOLApp = Nothing
    Set objNameSpace = Nothing
    Set objFolder = Nothing
    Set objItems = Nothing
End Sub
```

429 Outlook-Notiz erstellen

Wenden wir uns dem allerletzten Outlook-Beispiel zu. Diesmal soll aus Excel heraus eine Notiz erzeugt werden. Nach der Variablendeklaration und Referenzierung erzeugen wir das Outlook-Objekt und dann das `Item` für eine Notiz.



Abbildung 358: Outlook-Notiz erstellen

Unsere Prozedur ist so aufgebaut, dass der Notiz direkt im Code ein Text zugewiesen wird (`Body`). Danach wird die Farbe `Color` der Notiz ausgewählt und schließlich verwenden wir noch den Befehl `Display`, um die Notiz auf dem Bildschirm anzuzeigen.

Farb-Konstante zur <code>Color</code> -Eigenschaft	Beschreibung
<code>olBlue</code>	Blau
<code>olYellow</code>	Gelb
<code>olGreen</code>	Grün

Tabelle 99: Farben für Notizen

Farb-Konstante zur Color-Eigenschaft	Beschreibung
olPink	Rosa
olWhite	Weiß

Tabelle 99: Farben für Notizen (Forts.)

```
' =====
' Auf CD      Buchdaten\Beispiele\12_Extern\
' Dateiname  12_05_Outlook.xls
' Modul      mdl_06_CreateOutlookNote
' =====

' Verweis auf Microsoft Outlook x.x Object Library muss gesetzt sein
Sub CreateOutlookNote()
    Dim objOLApp As Outlook.Application
    Dim objNote As Outlook.NoteItem

    Set objOLApp = CreateObject("Outlook.Application")
    Set objNote = objOLApp.CreateItem(olNoteItem)

    With objNote
        .Body = "EINKAUFSZETTEL:" & vbCrLf & vbCrLf & _
            "2 Liter Milch" & vbCrLf & _
            "1 Kilogramm Brot" & vbCrLf & _
            "1 Stück Käse" & vbCrLf & _
            "500 Gramm Butter"
        .Color = olPink
        .Display
    End With

    Set objOLApp = Nothing
    Set objNote = Nothing
End Sub
```

Hinweis

Rezepte zum Thema E-Mail-Versand finden Sie in der Kategorie »Web/Mail«.

430 Access: Ein Query erstellen

Um an dieser Stelle nicht zu weit ausholen zu müssen, setzen wir gewisse Grundkenntnisse in Access oder anderen relationalen Datenbanken voraus. Für unsere Beispiele haben wir eine sehr einfache Access-Datenbank mit zwei Tabellen erstellt, die in einer 1:n-Beziehung miteinander verknüpft sind (siehe Abbildung 359). Die Datenbank trägt den Namen *myDatabase.mdb*. Wenn Sie, wie zu Beginn dieser Kategorie beschrieben, den Ordner *Codebook_12_Test* ins Laufwerk C:\ kopiert haben, steht Ihnen dort die Datei zur Verfügung.

Grundlagen
Allgemein
Datum und Zeit
Abfragen
Steuerungselemente
Befehlslisten
Objekte
Diagramme
Ereignisse
User-Formen
Web/Mail
Extern
Gemischtes
Spezial

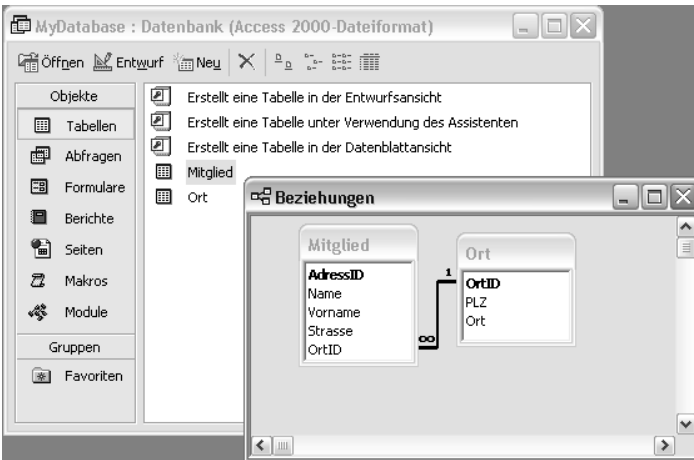
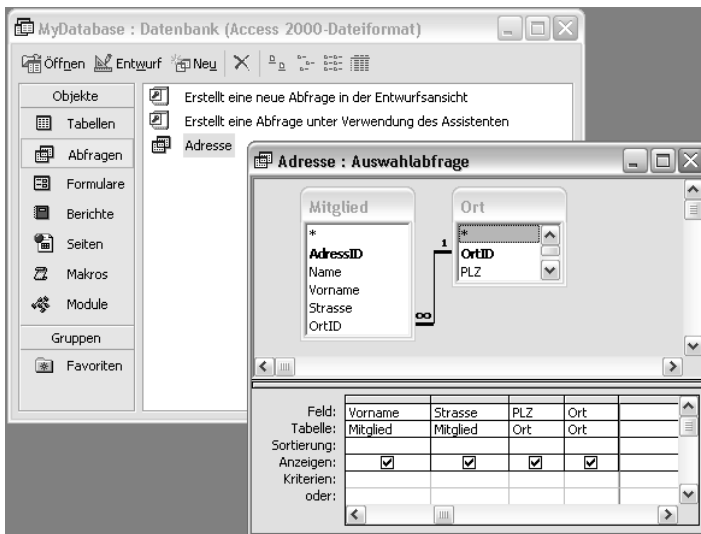


Abbildung 359: 1:n-Beziehung zwischen den beiden Tabellen (Relation)

Über diese beiden Tabellen wurde eine einzelne Abfrage mit dem Namen Adresse erstellt, die die Inhalte beider Tabellen zusammenfasst und damit die kompletten Adressen ausgibt (siehe Abbildung 360).



	Name	Vorname	Strasse	PLZ	Ort
►	Breden	Melanie	Gartenweg 8	30000	Bachhausen
■	Can	Zülfü	Bachstrasse 17	20000	Neustadt
■	Meier	Karl	Klugweg 5	10000	Altdorf
■	Muster	Fritz	Hohlgasse 9	40000	Oberthal
■	Weber	Monika	Dorfstrasse 7	10000	Altdorf
*					

Datensatz: 1 von 5

Abbildung 360: Abfrage über die beiden Tabellen

Bevor wir uns dem VBA zuwenden, werden wir erläutern, wie diese Daten ohne Programmierung nach Excel übertragen werden können. Wenn Sie lediglich die Daten in eine Excel-Tabelle kopieren möchten, ist der Vorgang sehr einfach. Markieren Sie in Access das Ergebnis der Abfrage, d.h., markieren Sie alle Datensätze, die nach dem Ausführen der Abfrage angezeigt werden. Kopieren Sie diese. Sie können dazu, wie gewohnt, die Schaltfläche für das Kopieren in der Symbolleiste verwenden oder mit der Tastenkombination **Strg + C** arbeiten. Wechseln Sie nach Excel und fügen Sie die Daten dort ein: **Strg + V**. Der Nachteil bei diesem Verfahren ist, dass so keine Verknüpfung zur Original-Datenbank besteht. Das bedeutet: Wenn Daten in Access verändert werden, werden diese in Excel *nicht* aktualisiert.

Sollte dies für Ihre Zwecke jedoch erforderlich sein, empfiehlt es sich, ein Query über die Abfrage zu erstellen. Um ein Query zu erstellen, klicken Sie in Excel auf den Menüpunkt: DATEN | EXTERNE DATEN IMPORTIEREN | NEUE ABFRAGE ERSTELLEN. Dabei öffnet sich das Dialogfenster DATENQUELLE AUSWÄHLEN. Aktivieren Sie darin den Eintrag Microsoft ACCESS-DATENBANK* und klicken Sie dann auf die Schaltfläche OK.

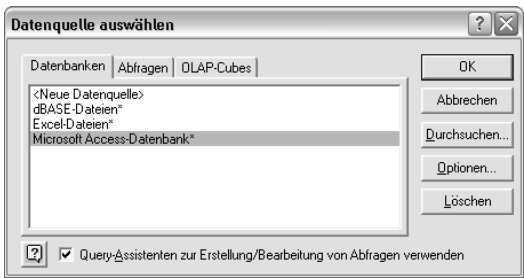


Abbildung 361: Datenquelle auswählen

Wählen Sie nun die Datenbank aus, indem Sie den Pfad und die Datei einstellen. Bestätigen Sie dann mit OK.

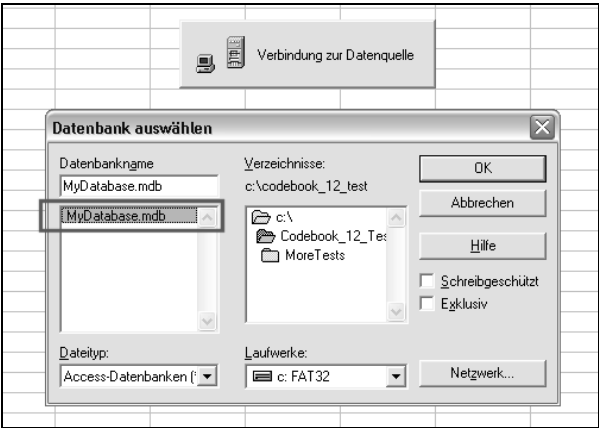


Abbildung 362: Datenbank auswählen

Die Abfrage und die beiden Tabellen werden angezeigt. Sie können nun direkt die Spalten aus der Abfrage auswählen. Mittels Doppelklick auf die Felder werden diese vom linken Fenster ins rechte übertragen. Klicken Sie dann in jedem Fenster des Assistenten auf die Schaltfläche WEITER. Im letzten Fenster klicken Sie auf die Schaltfläche FERTIG STELLEN.

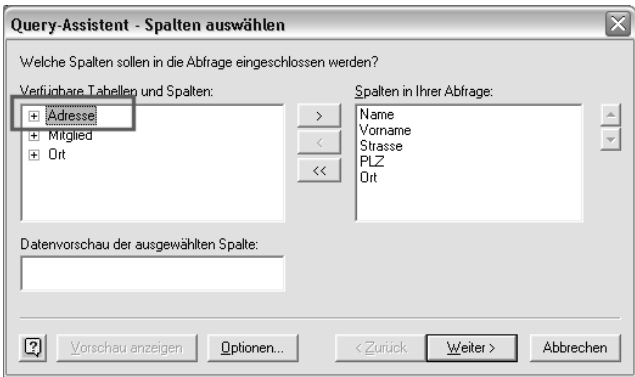


Abbildung 363: Spalten auswählen

Wählen Sie die Excel-Zelle aus, ab welcher die Daten übertragen werden sollen.



Abbildung 364: Daten importieren

Der gesamte Inhalt der Access-Datenbank ist nun in der Excel-Tabelle verfügbar. Der Vorteil dabei ist, dass eine Verbindung zur Quelle, also der Access-Datenbank besteht. Wie Sie sehen können, wird nun eine zusätzliche Symbolleiste angezeigt. Um die Daten in Excel zu aktualisieren, klicken Sie erst in die Tabelle und dann auf das rote Ausrufezeichen in der Symbolleiste EXTERNE DATEN. Die Daten werden damit aktualisiert.

```
' =====  
' Auf CD      Buchdaten\Beispiele\12_Extern\  
' Dateiname  12_06_Query.xls  
' =====
```

	A	B	C	D	E
1	Name	Vorname	Strasse	Ort	PLZ
2	Breden	Melanie	Gartenweg 8	Bachhausen	30000
3	Can	Zülfü	Bachstrasse 17	Neustadt	20000
4	Meier	Karl			10000
5	Muster	Fritz			40000
6	Weber	Monika	Dorfstrasse 7	Altdorf	10000
7					

Abbildung 365: Importierte Datenbank

Das Microsoft Query kann auch nachträglich eingesehen werden. Klicken Sie dazu in der Symbolleiste EXTERNE DATEN auf die erste Schaltfläche. Der Query-Assistent wird geöffnet. Klicken Sie im Assistenten auf die Schaltfläche ABBRECHEN. Es wird dann eine Meldung angezeigt (siehe Abbildung 366). Klicken Sie in diesem Dialogfenster auf die Schaltfläche JA.



Abbildung 366: Query bearbeiten

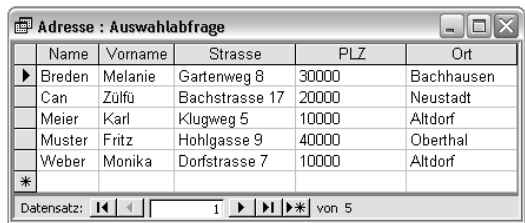
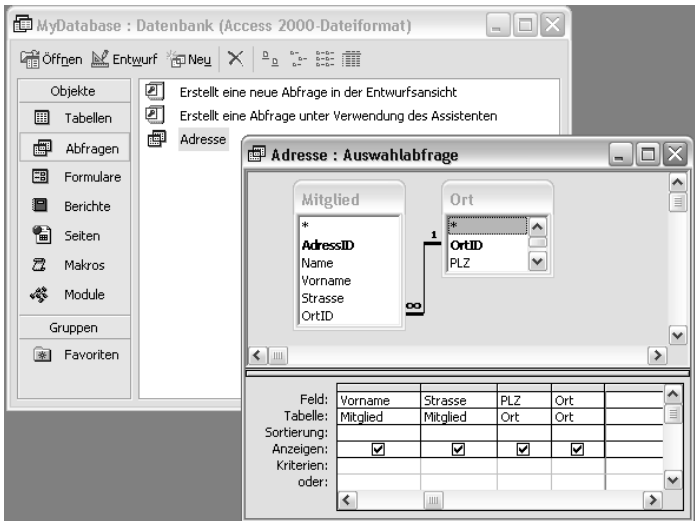


Abbildung 367: Abfrage über die beiden Tabellen

Das Programm »Microsoft Query« wird nun angezeigt (siehe Abbildung 367). Sie haben darin die Möglichkeit, verschiedene Änderungen vorzunehmen, z.B. eine Spalte auf- oder absteigend zu sortieren und vieles mehr.

Ein Klick auf die Schaltfläche SQL bewirkt, dass die verborgenen SQL-Anweisungen angezeigt werden.

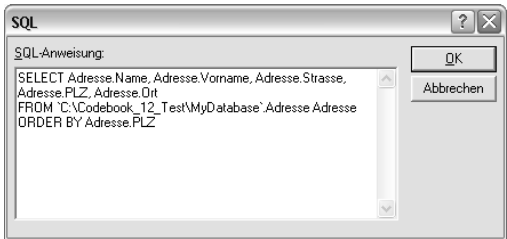


Abbildung 368: SQL-Anweisungen

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

In diesem Fenster können Änderungen vorgenommen werden. Voraussetzung dazu sind Kenntnisse im Umgang mit SQL.

Der Code in diesem Fenster enthält drei SQL-Befehle:

SELECT: Damit können Sie die Spalten aus der Tabelle oder Abfrage angeben. Bei jeder Spalte, die angezeigt werden soll, geben Sie zuerst den Namen der Tabelle oder Abfrage ein und dann den Namen der Spalte.

FROM: Hier wird die Quelle der Datenbank hinterlegt.

ORDER BY: Sortierung.

Um ein solches Query zu speichern, klicken Sie in der Symbolleiste auf die Schaltfläche, die eine Diskette darstellt. Es entsteht dabei eine Datei mit der Erweiterung *.dqy.

431 Import aus einer Access-Tabelle

Das Thema Import aus Access ist alles in allem sehr umfangreich. Dazu ist eigens in der Codebook-Reihe ein Buch entstanden.

Hinweis

Das Buch nennt sich »Das Access-VBA Codebook« und ist ebenfalls beim Verlag Addison-Wesley erschienen. Die ISBN-Nr. lautet 3-8273-1953-6.

Wir haben beschlossen, Ihnen ein Beispiel zum Import sowie eins zum Export aufzuzeigen. Falls die Beispiele hier für Ihren Bedarf nicht ausreichen, empfehlen wir Ihnen, sich gezielt Literatur zum Thema ADO (ActiveX Data Objects) zu beschaffen. Eine sehr gute Seite zu diesem Thema ist im Internet zu finden unter der Adresse: <http://www.erlandsendata.no/english/vba/adodao/index.htm>. Der Inhalt der Seite ist in englischer Sprache verfasst. Eine weitere interessante Webseite zu diesem Thema (in Deutsch) finden Sie unter der Adresse: http://www.activevb.de/tutorials/tut_ado_db/adodb.html.

Es gibt grundsätzlich zwei Bibliotheken, die Sie verwenden können, um auf Access zuzugreifen. Zum einen ist es die bereits erwähnte ADO, zum anderen DAO (Data Access Objects). Bei DAO handelt es sich um den Vorgänger von ADO. Im Gegensatz zu DAO wird ADO laufend weiterentwickelt. Dies ist unschwer an den verschiedenen Versionen zu erkennen, die bereits vorliegen (siehe Abbildung 369).

Hinweis

Um Daten zwischen Excel und Access austauschen zu können, muss die ADO-Bibliothek aktiviert sein. Beide der nachfolgenden Beispiele greifen auf diese Bibliothek zu. Aktivieren Sie die Bibliothek, indem Sie im VBA-Editor auf den Menüpunkte EXTRAS | VERWEISE klicken und dann das entsprechende Kontrollkästchen aktivieren.

In unserem ersten Beispiel besteht eine Access-Datenbank mit dem Namen *MyDatabase.mdb*. Die Datenbank befindet sich im Pfad *C:\Codebook_12_Test*, sofern Sie den entsprechenden Ordner, wie zu Beginn der Kategorie beschrieben, auf Ihr Laufwerk C:\ kopiert haben.

Sämtliche Datensätze aus der Access-Tabelle MITGLIED sollen in ein Tabellenblatt nach Excel übertragen werden.

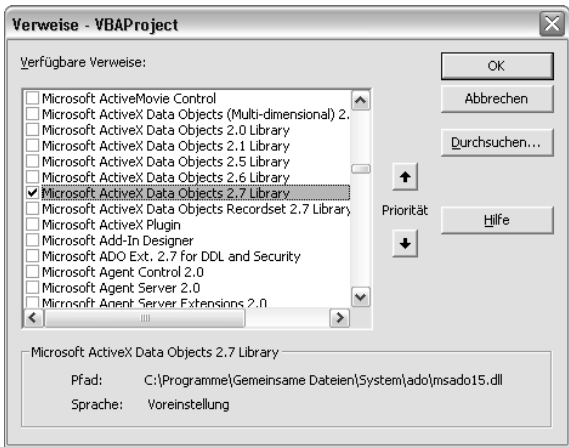
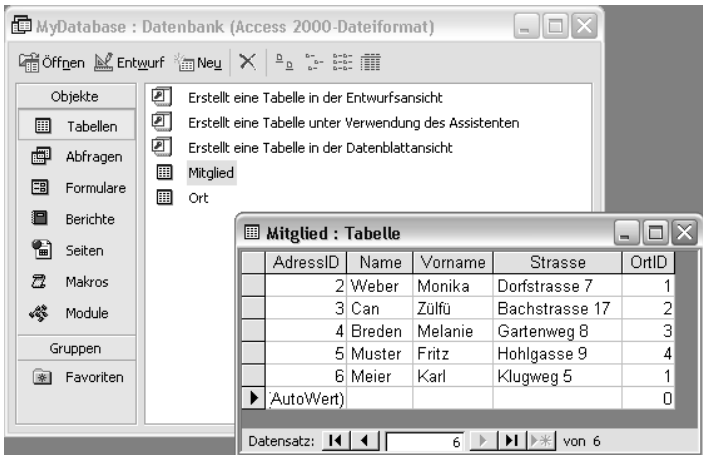


Abbildung 369: Verweis auf ADO-Bibliothek aktivieren



	A	B	C	D	E
1		2 Weber	Monika	Dorfstrasse 7	1
2		3 Can	Zülfü	Bachstrasse	2
3		4 Breden	Melanie	Gartenweg 8	3
4		5 Muster	Fritz	Hohlgasse 9	4
5		6 Meier	Karl	Klugweg 5	1
6					

Abbildung 370: Daten einer Access-Tabelle in eine Excel-Tabelle schreiben

Für den Aufbau der Verbindung nach Access sind zwei Objekte erforderlich. Diese werden gleich zu Beginn der Prozedur deklariert. Dabei wird auf die Bibliothek von ADO (ADODB) Bezug genommen. Es handelt sich einerseits um das Objekt `Connection`, das für die Verbindung zu Access eingesetzt wird, und andererseits um das Objekt `Recordset`, das für die Datensätze innerhalb der Tabelle zuständig ist. Beide Objekte werden nach der Variablendeklaration als `New` (Neues Objekt) referenziert.

Nach der Referenzierung wird zuerst die Datenbank geöffnet (`objConnection.Open`). Dabei werden zwei Parameter mitgeliefert. Der erste gibt den Provider an und der zweite die Datenquelle.

Grundlagen

Allgemein

Datum
Zeit

ungen

Steuer
elemente

Befehl
leisten

Objekt

Diagramm

Ereignisse

User-
Form

Web/
Mail

Extern

Ge-
misch-
tes

Specia

Nun kann die Tabelle geöffnet werden (`objRecSet.Open`). Dabei werden vier Parameter angegeben. Zuerst wird der Name der Access-Tabelle hinterlegt (`Mitglied`). Dann folgt die Verbindung und schließlich die Eigenschaften `CursorType` und `LockType`. Mittels `CursorType` kann das Verhalten des Cursors festgelegt werden. Über `LockType` wird die Sicherheitsstufe des Recordsets bestimmt:

CursorType-Konstanten	Beschreibung
<code>adLockReadOnly</code>	Nur Lesen. Es können keine Veränderungen vorgenommen werden (Standard).
<code>adLockPessimistic</code>	Die Daten werden beim Öffnen der Tabelle für andere Benutzer gesperrt.
<code>adLockOptimistic</code>	Die Daten werden erst beim Aktualisieren (Update) gesperrt.
<code>adLockBatchOptimistic</code>	Gleichzeitiges Ändern von Datensätzen möglich.

Tabelle 100: *CursorType-Konstanten*

LockType-Konstanten	Beschreibung
<code>adOpenForwardOnly</code>	Der Cursor bewegt sich nur vorwärts. Für andere Benutzer unsichtbar. Dies ist die schnellste Einstellung.
<code>adOpenDynamic</code>	Bewegungen in alle Richtungen sind möglich. Die Änderungen werden den anderen Benutzern angezeigt. Dies ist die langsamste Einstellung.
<code>adOpenKeyset</code>	Es wird für jeden Datensatz ein Schlüssel erstellt. Nur das Einfügen wird den anderen Benutzern nicht angezeigt. Er ist schneller als die Static- oder Dynamic-Einstellung.
<code>adOpenStatic</code>	Die Bewegung des Cursors ist in alle Richtungen möglich. Es können keine Daten verändert werden. Relativ langsam beim Einlesen des ersten Datensatzes, nicht jedoch bei den weiteren.

Tabelle 101: *LockType-Konstanten*

Die Daten können nun aus dem Recordset kopiert (`CopyFromRecordset`) und in Excel eingefügt werden. Da in einem Recordset alle Felder enthalten sind, reicht als Ziel die Angabe der Startzelle, in unserem Beispiel die Zelle A1.

Nach dem Kopieren müssen die offenen Verbindungen geschlossen (`Close`) und die referenzierten Objekte freigegeben werden.

```

' =====
' Auf CD      Buchdaten\Beispiele\12_Extern\
' Dateiname   12_07_Access.xls
' Tabelle     12_Extern_Access_Import
' Modul       mdl_01_ImportFromAccess
' =====

' Verweis auf Microsoft ActiveX Data Objects 2.7 Library
' muss gesetzt sein

```

```
Sub ImportFromAccess()  
    Dim objConnection As ADODB.Connection  
    Dim objRecSet As ADODB.Recordset  
  
    ' Verbindung und Datensätze referenzieren  
    Set objConnection = New ADODB.Connection  
    Set objRecSet = New ADODB.Recordset  
  
    ' Datenbank öffnen  
    objConnection.Open "Provider=Microsoft.Jet.OLEDB.4.0;" & _  
        "Data Source =" & _  
        "C:\Codebook_12_Test\MyDatabase.mdb;"  
  
    ' Daten von Access nach Excel übertragen  
    With objRecSet  
        ' Tabelle "Mitglied" öffnen  
        objRecSet.Open "Mitglied", _  
            objConnection, _  
            CursorType:=adOpenKeyset, _  
            LockType:=adLockOptimistic  
  
        ' Die Access-Felder nach Excel kopieren  
        Worksheets(1).[A1].CopyFromRecordset objRecSet  
    End With  
  
    ' Offene Verbindungen schließen  
    objRecSet.Close  
    objConnection.Close  
  
    ' Verweise freigeben  
    Set objRecSet = Nothing  
    Set objConnection = Nothing  
  
    MsgBox "Ende der Übertragung"  
End Sub
```

Grundlagen
Allgemein
Datum und Zeit
Formeln
Diagramme
Steuerungselemente
Befehlszeile
Objektmodell
Diagramme
Ereignisse
User-Formen
Web/Mail
Extern

432 Export in eine Access-Tabelle

In diesem Rezept besteht eine Access-Datenbank mit dem Namen *MyDatabase.mdb*. Die Datenbank befindet sich im Pfad *C:\Codebook_12_Test*, sofern Sie den entsprechenden Ordner, wie zu Beginn der Kategorie beschrieben, auf Ihr Laufwerk *C:* kopiert haben.

In Excel erfasste Datensätze werden an eine bestehende Access-Tabelle angefügt:

Das Verfahren dieser Prozedur ähnelt sehr dem vorangegangenen Beispiel. Zuerst werden die Objekte dimensioniert. Zusätzlich verwenden wir hier einen Zähler *i*, um in der Excel-Tabelle die Zeilen zu durchlaufen. Danach werden die neuen Objekte referenziert. Schließlich wird die Access-Datenbank und dann die Tabelle geöffnet. Das Übertragen von Excel nach Access kann nun beginnen. Im zweiten Tabellenblatt unserer Excel-Datei werden die Daten Zeile für Zeile und Zelle für Zelle ausgelesen. Eine *Do...Loop*-Schleife führt diesen Vorgang so lange aus, bis eine leere Zelle in der ersten Spalte gefunden wird.

Gemischtes
Spezial

	A	B	
1	PLZ	Ort	
2	50000	Meerhafen	
3	60000	Seedorf	
4	70000	Flusshausen	
5	80000	Bachstadt	
6			



Abbildung 371: Daten aus Excel nach Access in eine Tabelle übertragen

Mit einer With-Anweisung wird innerhalb des Recordsets von Access jeweils ein neuer Datensatz erzeugt (AddNew). Darin werden die einzelnen Felder gefüllt. Nach dem Schreiben eines Datensatzes wird dieser aktualisiert (Update).

Nach dem Übertragen der vorhandenen Datensätze werden die Verbindungen geschlossen (Close) und die referenzierten Objekte freigegeben.

```
' =====
' Auf CD      Buchdaten\Beispiele\12_Extern\
' Dateiname   12_07_Access.xls
' Tabelle     12_Extern_Access_Export
' Modul       mdl_02_ExportToAccess
' =====
```

```
' Verweis auf Microsoft ActiveX Data Objects 2.7 Library
' muss gesetzt sein
```

```
Sub ExportToAccess()
```

```
    Dim objConnection As ADODB.Connection
    Dim objRecSet As ADODB.Recordset
    Dim i As Integer
```

```
    ' Verbindung und Datensätze referenzieren
    Set objConnection = New ADODB.Connection
    Set objRecSet = New ADODB.Recordset
```



```
' Datenbank öffnen
objConnection.Open "Provider=Microsoft.Jet.OLEDB.4.0;" & _
                  "Data Source =" & _
                  "C:\Codebook_12_Test\MyDatabase.mdb;"

' Tabelle "Ort" öffnen
objRecSet.Open "Ort", _
              objConnection, _
              CursorType:=adOpenKeyset, _
              LockType:=adLockOptimistic

' Daten von Excel nach Access übertragen
i = 2
Do While Worksheets(2).Cells(i, 1) <> ""
    With objRecSet
        .AddNew                ' Neuer Datensatz
        .Fields("PLZ") = Cells(i, 1) ' Feld "PLZ"
        .Fields("Ort") = Cells(i, 2) ' Feld "Ort"
        .Update                ' Aktualisieren
    End With
    i = i + 1
Loop

' Offene Verbindungen schließen
objRecSet.Close
objConnection.Close

' Verweise freigeben
Set objRecSet = Nothing
Set objConnection = Nothing

MsgBox "Ende der Übertragung"
End Sub
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

433 Export nach Zwischenablage

Um in die Zwischenablage schreiben zu können, ist es erforderlich, dass der Verweis auf die Bibliothek »Microsoft Forms x.x Object Library« gesetzt ist, denn dort ist das DataObjekt-Objekt zu finden, das benötigt wird.

Um den genannten Verweis zu setzen, klicken Sie im VBA-Editor auf den Menüpunkt EXTRAS | VERWEISE und wählen die entsprechende Bibliothek aus (siehe Abbildung 372).

Die Zwischenablage können Sie auch füllen, ohne eine Prozedur zu verwenden. Um Text in die Zwischenablage zu schreiben, verwenden Sie in der Symbolleiste die Schaltfläche für KOPIEREN oder die Tastenkombination **[Strg] + [C]**, zuvor muss der Bereich markiert werden. Um den Inhalt aus der Zwischenablage wieder einzufügen, verwenden Sie in der Symbolleiste die Schaltfläche EINFÜGEN oder die Tastenkombination **[Strg] + [V]**.

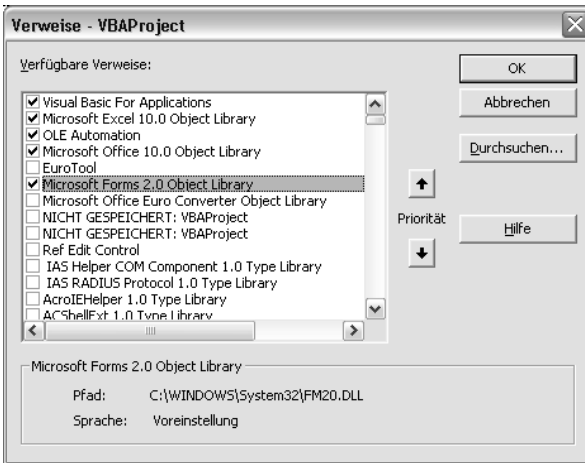


Abbildung 372: Verweis setzen

Die Zwischenablage und das `DataObject` sind nicht ein und dasselbe. Es bestehen Unterschiede, wie z.B.:

- ▶ Die Zwischenablage kann zwar Text sowie Bilder aufnehmen, das `DataObject` jedoch nicht. Es ist derzeit lediglich in der Lage, Text zu speichern.
- ▶ In der Zwischenablage bleiben die Daten erhalten, solange das Betriebssystem ausgeführt wird. Das ist mit dem `DataObject` nicht möglich. Der Inhalt des `DataObject` wird gelöscht, sobald die Anwendung geschlossen wird.

In unserem Beispiel wird zu Beginn der Prozedur, direkt nach der Variablendeklaration, ein neues `DataObject` erzeugt (`New DataObject`). Danach wird der Inhalt des Bereiches A1:B2 in die Zwischenablage geschrieben. Die Zellen werden dabei beginnend bei A1 zeilenweise von links nach rechts gelesen und der Variablen `strData` übergeben. Nach dem Auslesen jeder Zelle wird ein Leerzeichen angehängt, damit ein Abstand entsteht. Dieser Vorgang erfolgt innerhalb einer `For...Next`-Schleife.

Mittels `SetText` wird der Text aus der Variablen `strData` ausgelesen und anschließend an die Zwischenablage übergeben (`PutInClipboard`). Am Ende der Prozedur wird der Verweis auf das Datenobjekt entfernt, um den Speicherplatz wieder freizugeben.

Der Text befindet sich nun in der Zwischenablage und kann an einem beliebigen Ort, auch in einer anderen Anwendung, eingefügt werden.

```
' =====
' Auf CD      Buchdaten\Beispiele\12_Extern\
' Dateiname  12_08_Clipboard.xls
' Modul      mdl_01_WriteDataToClipboard
' =====

' Microsoft Forms 2.0 Object Library setzen
Sub WriteDataToClipboard()
    Dim c As Range
    Dim strData As String
    Dim MyData As DataObject
```

```
Set MyData = New DataObject

For Each c In Range("A1:B2")
    strData = strData & c & " "
Next c

With MyData
    .SetText strData
    .PutInClipboard
End With

Set MyData = Nothing
End Sub
```

434 Import aus Zwischenablage

Wie Sie nun wissen, kann Text mittels des Objekts DataObject in die Zwischenablage geschrieben werden. Der Inhalt, der so in die Zwischenablage übertragen wurde, kann auch wieder ausgelesen werden. Dies kann wahlweise in Excel oder einer anderen Anwendung geschehen.

Unser nächstes Beispiel baut auf dem vorangegangenen auf. In der ersten Hälfte der Prozedur werden die Daten in die Zwischenablage geschrieben. In der zweiten Hälfte der Prozedur wird ein neues Tabellenblatt erzeugt. In dieses werden die Daten geschrieben. Um die Daten auszulesen, verwenden wir die Methode GetFromClipboard. Mittels der Methode GetText werden die ausgelesenen Daten in die Zelle A1 geschrieben.

```
'=====
' Auf CD      Buchdaten\Beispiele\12_Extern\
' Dateiname  12_08_Clipboard.xls
' Modul      mdl_02_ReadDataFromClipboard
'=====

' Microsoft Forms 2.0 Object Library setzen
Sub ReadDataFromClipboard()
    Dim c As Range
    Dim strData As String
    Dim MyData As DataObject

    Set MyData = New DataObject

    ' Daten in Variable schreiben
    For Each c In Range("A1:B2")
        strData = strData & c & " "
    Next c

    ' Daten in Zwischenablage übertragen
    With MyData
        .SetText strData
        .PutInClipboard
    End With
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

```
' Tabellenblatt hinzufügen
Worksheets.Add

' Daten aus der Zwischenablage an Variable übergeben
MyData.GetFromClipboard

' Daten aus der Variablen ins neue
' Tabellenblatt schreiben
ActiveSheet.Range("A1") = MyData.GetText

' Inhalt der Variablen löschen
MyData.Clear

Set MyData = Nothing
End Sub
```

435 VBE – Entwicklungsumgebung

Der VB-Editor (VBE) ist die Entwicklungsumgebung für Ihre VBA-Projekte (Excel-Dateien). Auf die Entwicklungsumgebung können Sie, ebenso wie auf andere Objekte, wie Arbeitsmappen, Tabellen usw., über VBA-Befehle mit dem Ziel des Auslesens oder der Manipulation zugreifen.

Achtung

Ab Excel 2002 (XP) sind die Sicherheitsbestimmungen gegenüber den Vorgängerversionen verschärft worden. Um mit Makros auf Visual-Basic-Projekte zuzugreifen, müssen Sie in der Excel-Umgebung im Menü EXTRAS | MAKRO | SICHERHEIT im Register VERTRAUENSWÜRDIGE QUELLEN die Option ZUGRIFF AUF VISUAL BASIC PROJEKT VERTRAUEN aktivieren. Ist diese Option deaktiviert, gibt Microsoft Office eine Warnmeldung aus, wenn Makros versuchen, auf das Visual-Basic-Projekt zuzugreifen.

Eine gute Übersicht über die Elemente der Entwicklungsumgebung und den Zugriff darauf bietet die Microsoft-Hilfedatei *VBOB6.CHM* (siehe Abbildung 373). Diese Datei wird ab der Version Excel 2000 automatisch installiert. In vorherigen Versionen steht die Datei *veenob3.hlp* zur Verfügung. Üblicherweise finden Sie die Hilfedatei *VBOB6.CHM* im Ordner *C:\Programme\Gemeinsame Dateien\Microsoft Shared\VBA\VBA6\1031*.

Tipp

Um die Größe des verwendeten Speichers klein zu halten, werden standardmäßig nur die Bibliotheken in der Online-Hilfe geladen und zur Verfügung gestellt, deren Verweise auch gesetzt sind. Sobald der Verweis zu der *Visual Basic for Applications Extensibility*-Bibliothek gesetzt ist, ist auch der Zugriff zur oben genannten Hilfedatei direkt über die Online-Hilfe möglich. Um eine kompakte Übersicht des Themas zu erhalten, ist es jedoch nützlich, die VBE-Hilfedatei separat zu öffnen.

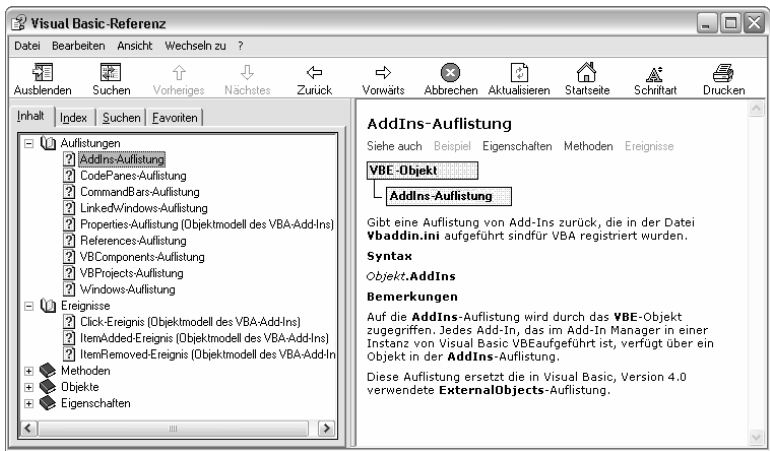


Abbildung 373: Microsoft-VBE-Hilfedatei VBOB6.CHM

Damit Sie mit den Objekten des Visual-Basic-Editors arbeiten können, müssen Sie zunächst einen Verweis auf seine Typenbibliothek erstellen. Die Elemente der VBA-Bibliothek, wie Objekte, Eigenschaften, Ereignisse, Konstanten usw., sind in den *Visual Basic for Applications Extensibilitys* hinterlegt. In Projekten, in denen Sie auf die VBA zugreifen, müssen Sie in der Entwicklungsumgebung über das Menü *Extras | Verweise* einen Verweis zu den »Extensibilities« setzen. Der Verweis wird mit der Arbeitsmappe gespeichert. Wenn Sie einen Verweis zu dieser geschaffen haben, können Sie sich über den Objektkatalog (F2) die Elemente der VBA anzeigen lassen. Markieren Sie eine Klasse und drücken Sie die Taste (F1), um die Microsoft Excel Online-Hilfe zu dem entsprechenden Thema aufzurufen.

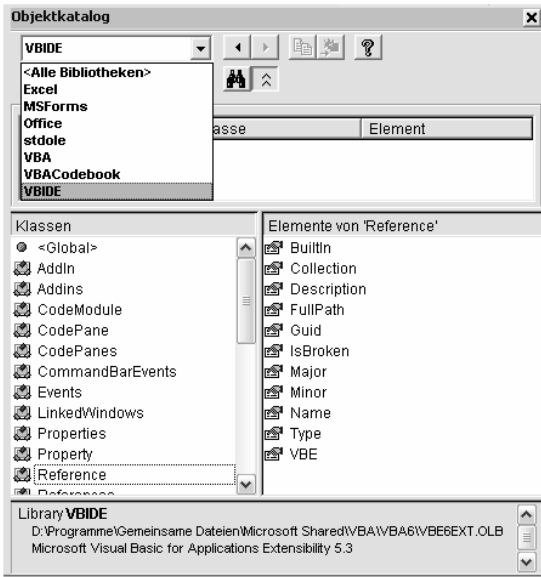


Abbildung 374: Elemente der VBA-Bibliothek

436 Verweise auslesen

Die verfügbaren Verweise im aktiven VBA-Projekt sehen Sie im Menü *EXTRAS | VERWEISE* (siehe Abbildung 375). Nachstehende Prozedur `ActiveVBEReferences` listet nur die aktiven der verfügbaren Verweise dieser Liste auf. Es ist nicht möglich, die Liste aller verfügbaren Verweise auszulesen, um gezielt einzelne Verweise aktivieren zu können.

Über eine Schaltfläche im Tabellenblatt der Beispieldatei wird die Prozedur `ActiveVBEReferences` aufgerufen. Bevor in dieser Prozedur ein neues Tabellenblatt mit dem Namen des VBA-Projektes in die Arbeitsmappe eingefügt wird, wird ein eventuell bereits vorhandenes Blatt gleichen Namens gelöscht. Über ein Array werden die Spaltenüberschriften für die aufzuführenden Eigenschaften eines jeden Verweises in die erste Zeile geschrieben.

- ▶ Die `Description`-Eigenschaft gibt beim `Reference`-Objekt den beschreibenden Namen des Verweises zurück.
- ▶ Die `Name`-Eigenschaft liefert den Namen des Verweises.
- ▶ Die `BuiltIn`-Eigenschaft gibt einen Wert vom Typ `Boolean` zurück, der angibt, ob ein Verweis ein Standardverweis ist, der nicht entfernt werden kann.

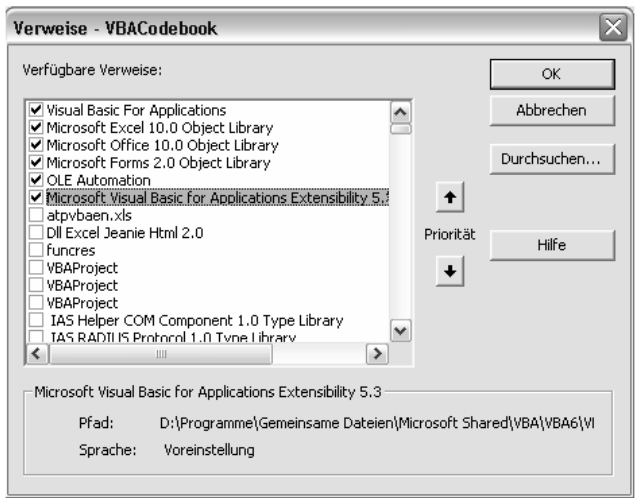


Abbildung 375: Aktivierte Verweise des selektierten VBA-Projektes

- ▶ Die Klassen- ID eines Objekts wird über die GUID-Eigenschaft ermittelt.
- ▶ Die von der Major-Eigenschaft zurückgegebene Zahl entspricht der Hauptversionsnummer, die in der referenzierten Klassenbibliothek gespeichert ist.
- ▶ Die Minor-Eigenschaft liefert die Nebenversionsnummer der Klassenbibliothek, auf die verwiesen wird.
- ▶ Über die FullPath-Eigenschaft ermitteln Sie den Pfad und den Dateinamen der Klassenbibliothek, auf die verwiesen wird.

```
'=====
' Auf CD      Buchdaten\Beispiele\13_Gemischtes\
' Dateiname  13_01_VBAProject.xls
' Modul      mdl_01_ReadReferences
'=====
```

```
Sub ActiveVBEReferences()
    Dim objRef As Object
    Dim intRow As Integer

    On Error Resume Next
    Application.DisplayAlerts = False
    Worksheets(ThisWorkbook.vbProject.Name).Delete
    Application.DisplayAlerts = True
    On Error GoTo 0

    Sheets.Add: ActiveSheet.Name = ThisWorkbook.vbProject.Name

    With ActiveSheet
        .Range("A1:G1") = Array("Beschreibung", "Name", _
                                "Standardverweis", "GUID", _
                                "HauptNr.", "NebenNr.", _
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

```

                                "LW:\Pfad\Datei")
.Rows(1).Font.Bold = True
intRow = 1

For Each objRef In Application.VBE.ActiveVBPProject.References
    intRow = intRow + 1
    .Cells(intRow, "A").Value = objRef.Description
    .Cells(intRow, "B").Value = objRef.Name
    .Cells(intRow, "C").Value = objRef.BuiltIn
    .Cells(intRow, "D").Value = objRef.GUID
    .Cells(intRow, "E").Value = objRef.Major
    .Cells(intRow, "F").Value = objRef.Minor
    .Cells(intRow, "G").Value = objRef.FullPath
Next
.Columns.EntireColumn.AutoFit
End With
End Sub

```

437 Verweise aktivieren

Verweise werden in der Regel von Hand über das Menü EXTRAS | VERWEISE gesetzt. Diesen Vorgang können Sie jedoch auch über eine Prozedur erledigen lassen. Die im Dialogfenster VERWEISE aktivierten Verweise (siehe Abbildung 375) werden über die References-Auflistung angesprochen. Die AddFromGuid-Methode fügt der References-Auflistung einen Verweis hinzu, wobei der global eindeutige Bezeichner (GUID) des Verweises verwendet wird. Dieser enthält die Klassen-ID eines Objekts. Die AddFromGuid-Methode durchsucht die Registrierung, um den hinzuzufügenden Verweis zu ermitteln.

Die folgende Prozedur setzt einen Verweis auf die *Microsoft Visual Basic for Application Extensibility 5.3*-Bibliothek. Die Fehlerroutine `On Error Resume Next` ist zwingend erforderlich, denn Excel würde eine Fehlermeldung ausgeben, wenn der angegebene Verweis bereits aktiviert wäre.

Tipp Um die Fehlerroutine `On Error Resume Next` zu umgehen, also die Zeile wegzulassen, können Sie mit der References-Auflistung aus dem vorherigen Rezept den Aktivierungsstatus eines Verweises vorab prüfen.

```

' =====
' Auf CD      Buchdaten\Beispiele\13_Gemischtes\
' Dateiname   13_01_VBAProject.xls
' Modul       mdl_02_ActivateReferences
' =====

Sub VBEActivate()
    Dim objVBE As Object

    On Error Resume Next
    ' Bibliothek Microsoft Visual Basic for Application
    ' Extensibility 5.3
    Set objVBE = Application.VBE.ActiveVBPProject.References._
        AddFromGuid("{0002E157-0000-0000-C000-000000000046}", 5, 3)

```


End Sub	Grund-lagen
Wenn Sie den genauen Pfad der zu aktivierenden Verweise kennen, können Sie auch mit der Add-FromFile-Methode die benötigten Verweise setzen.	Allge-mein
<pre>' ===== ' Auf CD Buchdaten\Beispiele\13_Gemischtes\ ' Dateiname 13_01_VBAProject.xls ' Modul mdl_02_ActivateReferences ' ===== Sub ReferencesActivate() Dim objVBE As Object Set objVBE = Application.VBE.ActiveVBProject.References On Error Resume Next ' Bibliothek OLE Automation objVBE.AddFromFile "D:\WINDOWS.0\System32\stdole2.tlb" ' Bibliothek Microsoft Visual Basic for Application ' Extensibility 5.3 objVBE.AddFromFile _ "D:\Programme\Gemeinsame Dateien\Microsoft Shared" & _ "\VBA\VBA6\VBE6EXT.OLB" End Sub</pre>	Datu Zeit tungen Steuer elemen Befehl leisten Objekt Dia-gramm Ereig-nisse
438 Verweise deaktivieren	User-Form

Um Verweise zu deaktivieren, verwenden Sie die Remove-Methode der References-Auflistung. Zur Identifizierung geben Sie den zu löschenden Verweisnamen an. In folgendem Beispiel wird der Verweis auf die *Microsoft Visual Basic for Application Extensibility 5.3*- Bibliothek entfernt.

<pre>' ===== ' Auf CD Buchdaten\Beispiele\13_Gemischtes\ ' Dateiname 13_01_VBAProject.xls ' Modul mdl_03_DeactivateReferences ' ===== Sub DeactivateReferences() Dim objVBE As Object Set objVBE = Application.VBE.ActiveVBProject.References On Error Resume Next ' Bibliothek Microsoft Visual Basic for Application ' Extensibility 5.3 objVBE.Remove objVBE("VBIDE") End Sub</pre>	Web/Mail Extern Ge-misch tes Specia

439 VBA-Projekteigenschaften auslesen

Im Dialogfenster PROJEKTEIGENSCHAFTEN eines jeden VBA-Projektes können Sie die Eigenschaften des Projektes, wie zum Beispiel den Namen, festlegen (siehe Abbildung 376). Sie erreichen den Dialog über den Menüpunkt EXTRAS | EIGENSCHAFTEN VON VBAPROJECT oder durch einen Rechtsklick im Projekt-Explorer innerhalb des Projektes und wählen den Menüeintrag EIGENSCHAFTEN VON VBAPROJECT.

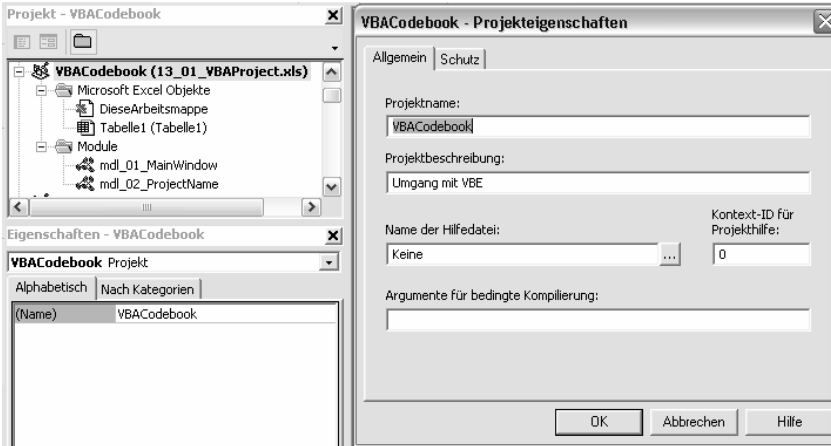


Abbildung 376: Eigenschaften des aktiven VBA-Projektes

Folgende Prozedur ProjectProperties listet die wichtigsten Eigenschaften des aktuellen Projektes im Direktfenster auf:

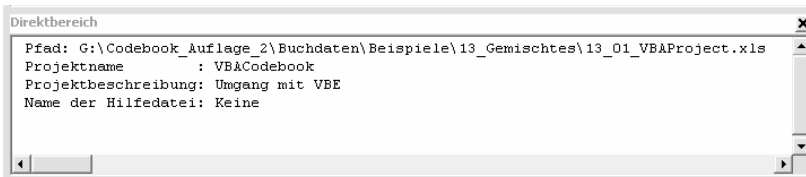


Abbildung 377: Eigenschaften des aktiven VBA-Projektes im Direktbereich auflisten

```
' =====
' Auf CD      Buchdaten\Beispiele\13_Gemischtes\
' Dateiname   13_01_VBAPROJECT.xls
' Modul       mdl_05_ProjectProperties
' =====
```

```
Sub ProjectProperties()
    Dim vbProject As Object

    Set vbProject = _
        Application.VBE.ActiveVBProject.VBComponents.Parent

    Debug.Print "Pfad: " & vbProject.FileName _
```

```
        & Chr(13) & _
        "Projektname           : " & vbProject.Name _
        & Chr(13) & _
        "Projektbeschreibung: " & vbProject.Description _
        & Chr(13) & _
        "Name der Hilfedatei: " & vbProject.HelpFile

End Sub
```

440 Projektname auslesen und ändern

Mit VBA ist es ohne weiteres möglich, nicht nur die Eigenschaften eines Projektes auszulesen, sondern auch Änderungen daran vorzunehmen. Folgende Syntax ändert den Namen des Visual-Basic-Projektes, in welchem der Code ausgeführt wird:

```
ThisWorkbook.VBProject.Name = "TestProject"
```

Im Tabellenblatt der Beispieldatei befinden zwei Schaltflächen, welche den aktuellen Projekt-namen der aktiven Arbeitsmappe in einem Meldungsfenster anzeigen bzw. den Namen über die Eingabe in einer InputBox ändern.

```
'=====
' Auf CD      Buchdaten\Beispiele\13_Gemischtes\
' Dateiname   13_01_VBAProject.xls
' Modul       mdl_05_ProjectProperties
'=====

Sub ProjectName()
    MsgBox Application.VBE.ActiveVBProject.VBComponents.Parent.Name
End Sub

'=====
' Auf CD      Buchdaten\Beispiele\13_Gemischtes\
' Dateiname   13_01_VBAProject.xls
' Modul       mdl_05_ProjectProperties
'=====

Sub ChangeProjectName()
    Dim strVBName As String
    Dim strName As String
    Dim objVBProject As Object

    Set objVBProject = _
        Application.VBE.ActiveVBProject.VBComponents.Parent

    strVBName = objVBProject.Name

    strName = InputBox("Geben Sie einen neuen Namen für das " & _
        strVBName & "-Projekt ein.", _
        "Neuer VBAProject Name für " & strVBName)

    If strName <> "" Then
```

Grund-lagen

Allge-mein

Datu-Zeit

tungen

Steuer-elemen

Befehl-leisten

Objekt

Dia-gramm

Ereig-nisse

User-Form

Web/Mail

Extern

Ge-mischtes

Specia

```

' Projektname ändern
objVBProject.Name = strName

MsgBox "Der Projekt Name " & strVBName & _
      " wurde in " & strName & " geändert."

Else
  MsgBox "Der Projekt Name " & strVBName & _
        " wurde nicht geändert."

End If
End Sub

```

441 Ist Projekt-Ansicht gesperrt?

Über die *Protection*-Eigenschaft des *VBProject*-Objektes können Sie ermitteln, ob die Projekt-Ansicht einer Excel-Arbeitsmappe gesperrt ist. Die Konstante *vbext_pp_locked* bzw. der Wert 1 gibt an, dass das angegebene Projekt gesperrt ist. Ist das angegebene Projekt nicht gesperrt, gibt die *Protection*-Eigenschaft den Wert 0 entsprechend *vbext_pp_none* zurück.

Geben Sie in die *InputBox* der folgenden Prozedur den Dateinamen ein, dessen Projektschutz Sie überprüfen wollen. Das VBA-Projekt der Beispielmappe ist mit dem Passwort *CODEBOOK* gesperrt.

```

' =====
' Auf CD      Buchdaten\Beispiele\13_Gemischtes\
' Dateiname   13_01_VBAProject.xls
' Modul       mdl_06_IsProjectLocked
' =====

Sub IsProjectLocked()
  Dim strWorkbook As String

  strWorkbook = InputBox( _
    "Geben Sie den Namen der Arbeitsmappe ein, " & _
    "dessen Projektschutz überprüft werden soll.", _
    Default:=ThisWorkbook.Name)

  If strWorkbook <> "" Then

    MsgBox _
      "Datei " & ThisWorkbook.Name & " ist " & _
      IIf(Workbooks(strWorkbook).vbProject.Protection _
        = vbext_pp_locked, _
        "gesperrt.", "nicht gesperrt.")

  End If
End Sub

```

442 Mit VBA Projektschutz aufheben

Um ein VBA-Projekt zu entsperren und das Passwort zu löschen, können Sie einerseits mit einigen Mausklicks den bereits weiter oben beschriebenen Weg über den Menüpunkt **EXTRAS | EIGENSCHAFTEN VON VBAPROJECT** gehen und im Dialog **PROJEKTEIGENSCHAFTEN** die Änderungen vornehmen. Diese Befehlsfolge können Sie aber auch mit Ihrer Tastatur erreichen. Wenn Sie sich

in der Tabellenansicht von Excel befinden, sind dazu nachfolgende Tastenanschläge und Texteingaben erforderlich. Diese können Sie in die SendKeys-Anweisung integrieren und so mittels VBA den Projektschutz eines VBA-Projektes aufheben bzw. setzen. Die SendKeys-Anweisung sendet eine Tastenfolge (die aus einem oder mehreren Tastenanschlägen bestehen kann) an das aktive Fenster, als ob sie über die Tastatur eingegeben worden wäre.

Befehl	Tastatur	SendKey
VBA-Editor anzeigen		%{F11}
Menü EXTRAS EIGENSCHAFTEN		%xi
Zu Registerreitern springen		+{TAB}
Register SCHUTZ auswählen		{RIGHT}
Zur CHECKBOX springen		{TAB}
Kontrollkästchen aktivieren / deaktivieren		Leertaste
Zur KENNWORTZEILE springen		{TAB}
Kennwort löschen		{BACKSPACE}
Zur KENNWORT BESTÄTIGEN-Zeile springen		{TAB}
Kennwort löschen		{BACKSPACE}
Zur Schaltfläche OK springen		{TAB}
Schaltfläche OK auslösen		{ENTER}
VBA-Editor schließen		%{q}

Tabelle 102: Tastaturanschläge und SendKey-Befehle zum Aufheben des Projektschutzes

Hinweis

Eine Übersicht der Tastenanschläge für die SendKeys-Anweisung und weitere Erläuterungen finden Sie im Anhang unter dem Stichwort »OnKey-Methode«. Die Zeichencodes und Handhabung dieser beiden Methoden sind identisch.

Das VBA-Projekt unserer Beispieldatei ist mit dem Passwort »Codebook« gesperrt. Öffnen Sie die Datei und heben Sie den Schutz mit folgender Prozedur auf. Dadurch wird die Ansicht auf die Komponenten des Projektes freigegeben.

```
'=====
' Auf CD      Buchdaten\Beispiele\13_Gemischtes\
' Dateiname  13_01_VBAProject.xls
' Modul      mdl_07_VBAPassword
'=====

Sub OpenVBAProject()
    If Val(Application.Version) > 8 Then
        SendKeys _
            "%{F11}%xi" & "Codebook" & "{ENTER 2}%{q}"
    End If
End Sub
```

Grundlagen
Allgemein
Datum und Zeit
Formeln
Diagramme
Diagramm
Ereignisse
User-Form
Web/Mail
Extern
Gemischtes
Spezial

Hinweis

Damit unsere Prozeduren keine Fehlermeldung aufgrund einer früheren Version auslösen, wird über die Val-Funktion die verwendete Excel-Version Application.Version ausgelesen und der Code nur in Versionen ab Excel 9.0 (Excel 2000) ausgeführt.

In Excel 97 wird der Menüpunkt EIGENSCHAFTEN VON VBAPROJECT mit der Tastenkombination **[Alt] + [S]** aufgerufen. Um die Prozeduren aus diesem Beispiel in dieser Version anzuwenden, ändern Sie die Zeichenfolge von %xi auf %xs.

Im Tabellenblatt wurden zwei Schaltflächen die Prozeduren UnlockedVBAProject und LockedVBAProject zur Änderung des Sperrstatus des VBA-Projektes der Beispielmappe zugewiesen. Die einzelnen Schritte der SENDKEYS-Anweisung sind in Bruchteilen einer Sekunde auf dem Bildschirm sichtbar. Das lässt sich auch nicht mit der Deaktivierung der Bildschirmaktualisierung, der ScreenUpdating-Eigenschaft verhindern, da die SendKeys-Methode ansonsten keinen Zugriff auf die angesprochenen Befehle ausüben kann.

Um den Sperrstatus zu ändern, verwenden Sie die folgenden Prozeduren. Beachten Sie, dass vorab der Projektschutz aufgehoben werden muss, damit die Ansicht auf das Projekt möglich ist.

Achtung

Wenn Sie versuchen, ein bereits entsperartes VBProjekt zu entsperren, wird eine Fehlermeldung ausgegeben und der Debugger gestartet. Dies lässt sich auch nicht mit einer Fehler-routine verhindern. Aus diesem Grund wird der aktuelle Status des Projektschutzes zu Ihrer Information im Tabellenblatt in die Zelle H26 geschrieben.

```
'=====
' Auf CD      Buchdaten\Beispiele\13_Gemischtes\
' Dateiname   13_01_VBAProject.xls
' Modul       mdl_07_VBAPassword
'=====
```

```
Sub UnlockedVBAProject()
    If Val(Application.Version) > 8 Then
        SendKeys _
            "%{F11}%xi+{TAB}{RIGHT}{TAB} {TAB}" & _
            "{BACKSPACE}{TAB}{BACKSPACE}{TAB}{ENTER}%{q}"
    End If
    Range("H26").Value = "Unlocked"
End Sub
```

```
Sub LockedVBAProject()
    If Val(Application.Version) > 8 Then
        SendKeys _
            "%{F11}%xi+{TAB}{RIGHT}{TAB} {TAB}" & _
            "Codebook" & "{TAB}" & "Codebook" & "{TAB}{ENTER}%{q}"
    End If
    Range("H26").Value = "Locked"
End Sub
```

443 Entwicklungsumgebung ein- und ausblenden

Die Entwicklungsumgebung (VBA-Editor) können Sie zum Beispiel mit der Tastenkombination **[Alt] + [F11]** oder über die VISUAL BASIC-Symboleiste aufrufen, das heißt einblenden. Das Schließen der Entwicklungsumgebung ist standardmäßig nur möglich, wenn Sie sich in dieser Ansicht

finden. Klicken Sie dazu auf das Schließen-Kreuz in der Titelzeile der Entwicklungsumgebung, oder drücken Sie die Tastenkombination **[Alt]+[Q]**.

Folgende Prozeduren blenden die Entwicklungsumgebung ein oder aus. Im Tabellenblatt der Beispieldatei wurden diese Prozeduren zwei Schaltflächen zugewiesen.

```
'=====
' Auf CD      Buchdaten\Beispiele\13_Gemischtes\
' Dateiname  13_02_VBEWindows.xls
' Modul      mdl_01_MainWindow
'=====

Sub OpenMainWindow()
    With Application.VBE.MainWindow
        .Visible = True
        .SetFocus
    End With
End Sub

Sub CloseMainWindow()
    Application.VBE.MainWindow.Visible = False
End Sub
```

444 Codemodul-Fenster auflisten

Es ist sogar möglich, im Visual-Basic-Editor über die **Windows-Auflistung** ein bestimmtes Codemodul anzuzeigen. Die folgende Prozedur listet alle im VBE sichtbaren Fenster im Direktbereich auf. Das aktive Codemodul-Fenster wird nur über seine Bezeichnung, also ohne vorangestellten Dateinamen ausgegeben und angesprochen (siehe Abbildung 378).

Achtung

Wenn im Dialogfenster PROJEKTEIGENSCHAFTEN, Register SCHUTZ die Option PROJEKT FÜR ANZEIGE GESPERRT aktiviert ist, kann für dieses VBA-Projekt keine Windows-Auflistung und kein namentlicher Zugriff auf die enthaltenen Codemodule erfolgen.

Tipp

Die Prozedur können Sie auch direkt über das Direktfenster starten. Schreiben Sie folgende Zeile in den Direktbereich und bestätigen Sie die Eingabe mit **[↵]**:
run "VBEWindows"

```
'=====
' Auf CD      Buchdaten\Beispiele\13_Gemischtes\
' Dateiname  13_02_VBEWindows.xls
' Modul      mdl_02_VBEWindows
'=====

Sub VBEWindows()
    Dim VBEWin As Object

    For Each VBEWin In Application.VBE.Windows
        Debug.Print VBEWin.Caption
    Next VBEWin
End Sub
```

Grund-lagen

Allge-mein

Datu Zeit

tungen

Steuer elemen

Befehl leisten

Objekt

Dia-gramm

Ereig-nisse

User-Form

Web/Mail

Extern

Ge-misch tes

Specia

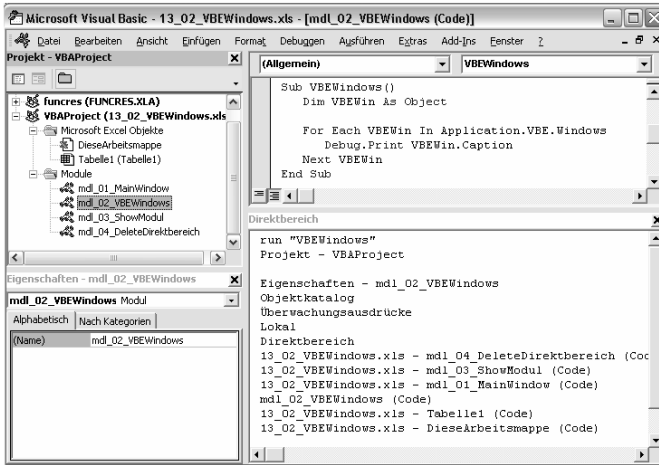


Abbildung 378: Fenster-Auflistung der Entwicklungsumgebung

445 Codemodul-Fenster anzeigen

Um gezielt ein Codemodul anzuzeigen, stellen Sie zunächst sicher, dass der Editor mit `Application.VBE.MainWindow.Visible = True` eingeblendet ist. Anschließend wird über eine Zeichenkette, welche aus dem Namen der Arbeitsmappe sowie der genauen Fenster-Bezeichnung besteht, mit der `SetFocus`-Eigenschaft der Focus auf das angegebene Modul gesetzt. Wenn das angegebene Modul das aktive Fenster im VBE ist, wird ein Fehler erzeugt. Dieser wird über die Sprungmarke `ERRORHANDLER` abgefangen. Es erfolgt eine erneute Zuweisung einer Zeichenkette an die Variable `strWindows`. Diesmal setzt sich der Fensternamen ohne vorangestellten Dateinamen zusammen. Der Codebereich des genannten Moduls wird angezeigt.

Hinweis

Wenn Sie mit VBA das Fenster anzeigen wollen, welches gerade aktiv ist, wird lediglich der Modulname ohne Dateiname angegeben.

```
' =====
' Auf CD      Buchdaten\Beispiele\13_Gemischtes\
' Dateiname   13_02_VBEWindows.xls
' Modul       mdl_03_ShowModul
' =====
```

```
Sub ShowModul()
    Dim strCodeMod As String
    Dim strWindows As String

    strCodeMod = "mdl_03_ShowModul"

    Err.Clear
    strWindows = ThisWorkbook.Name & " - " & strCodeMod & " (Code)"

    If Err.Number > 0 Then
```



```

ERRORHANDLER:
    strWindows = strCodeMod & " (Code)"
End If

With Application.VBE
    .MainWindow.Visible = True
    On Error GoTo ERRORHANDLER
    .Windows(strWindows).SetFocus
End With
End Sub

```

Die Windows-Auflistung liefert außer den Codemodulen auch alle Fenster der Entwicklungsumgebung zurück. Dabei spielt es keine Rolle, ob diese sichtbar sind. Auf diese Fenster können Sie mit dem ermittelten Namen zugreifen:

- Projekt – VBACodebook (*Projekt-Explorer*)
- Eigenschaften – mdl_02_VBEWindows (*Eigenschaftsfenster*)
- Objektkatalog
- Überwachungsausdrücke
- Lokal
- Direktbereich

Am Beispiel des DIREKTBEREICHs stellen wir Ihnen in nachfolgendem Rezept eine Prozedur zur Verwendung des Window-Objektes vor.

446 Text in Direktbereich löschen

Dass Sie mit der Print-Methode des Debug-Objektes Text-Ausgaben zur Laufzeit zum DIREKT-FENSTER senden können, ist weithin bekannt. Um den Text im DIREKTBEREICH wieder zu löschen, markieren Sie einzelne Zeilen im Direktfenster oder drücken im aktivierten Direktfenster die Tastenkombination **[Strg] + [a]**, um alle Zeilen zu markieren. Drücken Sie anschließend die Taste **[Entf]**, um den markierten Text im Direktfenster zu löschen.

Nachfolgende Prozedur DeleteDirektbereich schreibt zunächst einen Text in das Direktfenster. Über die Windows-Auflistung wird der Zugriff auf den DIREKTBEREICH an eine With-Anweisung übergeben. Nachdem über die Visible-Eigenschaft das Direktfenster sichtbar gemacht wurde, unterbricht die Stop-Anweisung die Ausführung der laufenden Prozedur. Dadurch geben wir Ihnen die Möglichkeit, die Ausführung des Debug.Print-Befehls zu überprüfen. Drücken Sie die Taste **[F5]**, um das Makro zu Ende laufen zu lassen. Wenn Sie die Prozedur schrittweise mit der Taste **[F8]** weiterlaufen lassen, erhält das Direktfenster nicht den Focus, da der Focus im Codemodul der Prozedur verbleibt, in welchem die ausführende Prozedur steht. Die SendKeys-Anweisung führt die Tastenfolge **[Strg] + [a] + [Entf]** `"^a{DEL}"` aus und könnte durch die Stop-Anweisung letztendlich sogar den Code der eigenen Prozedur löschen. Wenn Sie die Prozedur in realen Projekten einsetzen, löschen Sie die Stop-Anweisung oder kommentieren diese durch ein vorangestelltes Hochkomma (') aus.

Tipp

Um alle aufgeführten Elemente im ÜBERWACHUNGSFENSTER zu löschen, sprechen Sie statt des Direktbereichs dieses Fenster an:

```
With Application.VBE.Windows("Überwachungsausdrücke")
```

Grundlagen

Allgemein

Datum
Zeit

tungen

Steuer
elemente

Befehl
leisten

Objekt

Diagramm

Ereignisse

User-
Form

Web/
Mail

Extern

Gemischtes

Spezial

```
'=====
' Auf CD      Buchdaten\Beispiele\13_Gemischtes\
' Dateiname   13_02_VBEWindows.xls
' Modul       mdl_04_DeleteDirektbereich
'=====

Sub DeleteDirektbereich()
    Debug.Print "Das Excel-VBA Codebook"

    With Application.VBE.Windows("Direktbereich")
        .Visible = True
        Stop ' weiter mit <F5>
        .SetFocus
        SendKeys "^a{DEL}"
    End With
End Sub
```

447 Alle Projekte auslesen

Um alle VBA-Projekte der aktiven Excel-Anwendung und die darin gespeicherten Komponenten auszulesen, müssen einige Fehlerquellen berücksichtigt werden.

Zu jedem Projekt soll in der Spalte A des neu eingefügten Tabellenblattes ALLVBEPROJECTS der komplette Pfad und Dateiname ausgegeben werden. Über den Aufruf AddSheet "AllVBEPProjects" wird ein neues Tabellenblatt vor dem sichtbaren Blatt der aktiven Arbeitsmappe eingefügt. In dieser Prozedur wird ein eventuell vorhandenes Tabellenblatt gleichen Namens gelöscht. Damit Microsoft Excel vor dem Löschvorgang keine Warnungen und Meldungen anzeigt, werden diese über die DisplayAlerts-Eigenschaft zunächst deaktiviert und abschließend wieder aktiviert. Ein Tabellenblatt mit dem im Argument strShName angegebenen Namen wird neu angelegt. In einigen der folgenden Beispiele wird vor einer Auflistung ein neues Tabellenblatt eingefügt. Zur Vereinfachung wird jeweils die Prozedur AddSheet herangezogen.

```
'=====
' Auf CD      Buchdaten\Beispiele\13_Gemischtes\
' Dateiname   13_03_VBProjectsAndModules.xls
' Modul       mdl_01_AddSheet
'=====

Sub AddSheet(strShName As String)
    On Error Resume Next
    Application.DisplayAlerts = False

    With ActiveWorkbook
        .Sheets(strShName).Delete
        .Sheets.Add
        .ActiveSheet.Name = strShName
    End With

    Application.DisplayAlerts = True
End Sub
```

Wenn die Arbeitsmappe und somit das VBA-Projekt noch nicht gespeichert wurde, kann die File-Name-Eigenschaft nicht ausgewertet werden. Dieser Fehler wird mit der Fehlerroutine On Error Resume Next zunächst abgefangen. Im Anschluss daran wird mit Err.Number überprüft, ob der Versuch, den Pfadnamen in eine Zelle zu schreiben, einen Fehler ausgelöst hat. In dem Fall wird der Vermerk »Projekt nicht gespeichert« in die Zelle geschrieben.

Es ist nicht möglich, auf die Komponenten eines gesperrten Projekts zuzugreifen. Über die Protection-Eigenschaft des VBProject-Objekts wird dementsprechend der Sperrstatus überprüft. In einer For Each...Next-Anweisung werden alle Komponenten eines VBProjektes durchlaufen. Die Eigenschaften Window und Caption des CodePane-Objekts werden verwendet, um den Titel des angegebenen Codebereichs zurückzugeben.

```
'=====
' Auf CD      Buchdaten\Beispiele\13_Gemischtes\
' Dateiname   13_03_VBProjectsAndModules.xls
' Modul       mdl_01_AllVBProjects
'=====
```

```
Sub AllVBProjects()
    Dim VBProj As VBProject
    Dim VBComp As VBComponent
    Dim intI As Integer

    intI = 1
    AddSheet "AllVBProjects"

    With ActiveSheet
        For Each VBProj In Application.VBE.VBProjects

            With .Cells(intI, "A")
                Err.Clear
                On Error Resume Next
                .Value = VBProj.Filename
                If Err.Number > 0 Then
                    .Value = "Projekt nicht gespeichert"
                End If
                intI = intI + 1
            End With

            If VBProj.Protection = vbext_pp_locked Then
                .Cells(intI, "B").Value = "Projekt gesperrt"
                intI = intI + 1
            Else
                For Each VBComp In VBProj.VBComponents
                    .Cells(intI, "B").Value = _
                        VBComp.CodeModule.CodePane.Window.Caption
                    intI = intI + 1
                Next VBComp
            End If
            intI = intI + 1

        Next VBProj
    End With
End Sub
```

Grundlagen

Allgemein

Datum
Zeit

Formeln

Steuerelemente

Befehle
leisten

Objekte

Diagramme

Ereignisse

User-Formen

Web/
Mail

Extern

Gemischtes

Spezial

448 Module und Prozeduren des aktiven VBA-Projektes auslesen

Zu jeder Excel-Arbeitsmappe gehört ein VBA-Projekt. In der deutschen Excel-Version wird dies VBAPROJECT genannt und bezeichnet eine Gruppe von Modulen. Zu jedem VBA-Projekt gehört standardmäßig ein Dokumentmodul DIESEARBEITSMAPPE sowie für jedes Tabellen- und Diagrammblatt der Arbeitsmappe ein eigenes Dokumentmodul mit dem Standardnamen des Blattes (TABELLE1, DIAGRAMM1). Des Weiteren können beliebig viele Standardmodule, Klassenmodule oder Formulare in ein VBA-PROJEKT eingefügt werden.

Konstante	Wert	Typ	Beschreibung
vbext_ct_StdModule	1	Standardmodul	(Code-Modul) Enthält nur Prozeduren, Typen, Daten-deklarationen und Datendefinitionen.
vbext_ct_ClassModule	2	Klassenmodul	Komponente zur Erstellung und Definierung neuer Klassen
vbext_ct_MSForm	3	Microsoft-Formular	Komponente für UserForms
vbext_ct_ActiveXDesigner	11	ActiveX Designer	
vbext_ct_Document	100	Dokumentmodul	Komponente für Tabellen-, Diagrammblätter oder »DieseArbeitsmappe«

Tabelle 103: Verfügbare Komponenten eines VBA-Projektes

In diesem Beispiel werden alle Namen der Prozeduren in den vorhandenen Klassen- und Standardmodulen mit Nennung der Module und der Modultypen in dem eingefügten Tabellenblatt LISTMODULES ausgegeben. Über eine For Each...Next-Anweisung wird jede Komponente (Modul) des aktiven VBA-Projektes durchlaufen. In Spalte A des eingefügten Blattes wird der Name der jeweiligen Komponente geschrieben. Die Funktion CompTypeToName wandelt die Konstante des Komponententyps (z.B. vbext_ct_Document) in eine Zeichenkette Dokumentmodul um. Diese wird neben dem Namen in Spalte B geschrieben.

Über die CodeModule-Eigenschaft wird auf den Code innerhalb der Komponente zugegriffen. In der folgenden For...Next-Anweisung wird über die CountOfLines-Eigenschaft des CodeModule-Objektes jede Codezeile des angegebenen Moduls durchlaufen. Die ProcOfLine-Eigenschaft des CodeModule-Objekts gibt den Namen der Prozedur zurück, in der sich die festgelegte Zeile befindet. Ein ermittelter Prozedurname wird in der Variablen strSubName gespeichert und in die Spalte C geschrieben.

```
'=====
' Auf CD      Buchdaten\Beispiele\13_Gemischtes\
' Dateiname   13_03_VBProjectsAndModules.xls
' Modul       mdl_03_ModulesAndSubs
'=====

Sub ModulesAndSubs()
    Dim vbComp As VBIDE.VBComponent
```

```
Dim iCounter As Integer
Dim iRow As Integer
Dim strSubName As String

AddSheet "ListModules"

With Range("A1:C1")
    .Value = Array("Code-Modul", "Typ", "Prozeduren")
    .Font.Bold = True
End With

For Each vbComp In ThisWorkbook.VBProject.VBComponents
    iCounter = iCounter + 2
    Cells(iCounter, "A") = vbComp.Name
    Cells(iCounter, "B") = CompTypeToName(vbComp)

    With vbComp.CodeModule
        For iRow = 1 To .CountOfLines
            If .ProcOfLine(iRow, 0) <> "" And _
                .ProcOfLine(iRow, 0) <> strSubName Then
                strSubName = .ProcOfLine(iRow, 0)
                Cells(iCounter, "C") = strSubName
                iCounter = iCounter + 1
            End If
        Next iRow
    End With

    iRow = 0
Next vbComp
Columns.AutoFit
End Sub
```

```
'=====
' Auf CD      Buchdaten\Beispiele\13_Gemischtes\
' Dateiname  13_03_VBProjectsAndModules.xls
' Modul      mdl_03_ModulesAndSubs
'=====
```

```
Function CompTypeToName(vbeComp As VBComponent) As String
    Select Case vbeComp.Type
        Case vbext_ct_ActiveXDesigner
            CompTypeToName = "ActiveX Designer"
        Case vbext_ct_ClassModule
            CompTypeToName = "Klassenmodul"
        Case vbext_ct_Document
            CompTypeToName = "Dokumentmodul"
        Case vbext_ct_MSForm
            CompTypeToName = "Microsoft-Formular"
        Case vbext_ct_StdModule
            CompTypeToName = "Standardmodul"
        Case Else
    End Select
End Function
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

449 Code einer Prozedur auslesen

Um den Code dieses Beispiels auszulesen, muss Ihnen der Name des Codemoduls `mdl_04_ReadCode`, in welchem der Code hinterlegt ist, sowie der Prozedurname `ReadCode` bekannt sein. Diese werden in den Variablen `strModName` und `strSubName` hinterlegt. Nachdem die Komponente mit der `Set`-Anweisung an die Variable `vbComp` übergeben wurde, werden in einer `With`-Anweisung verschiedene Eigenschaften des `CodeModule`-Objektes ermittelt.

Damit der Code in einem Meldungsfenster angezeigt werden kann, benötigen wir die `Lines`-Eigenschaft. Diese gibt eine Zeichenfolge zurück, welche die angegebene Anzahl von Code-Zeilen enthält. Der erste Parameter erwartet die Startzeile `intStart` der angegebenen Prozedur. Diese wird mit der `ProcBodyLine`-Eigenschaft des `CodeModule`-Objektes ermittelt. Sie gibt die erste Zeile einer Prozedur zurück. Die erste Zeile einer Prozedur ist die Zeile, in der die `Sub`-, `Function`- oder `Property`-Anweisung enthalten ist. Die Anzahl der Zeilen `intCount` für die `Lines`-Eigenschaft erfahren Sie über die `ProcCountLines`-Eigenschaft. Sowohl diese als auch die `ProcBodyLine`-Eigenschaft erwarten als zweiten Parameter die Art der zu suchenden Prozedur.

Sie können eine der folgenden Konstanten für das Argument `ProzArt` verwenden:

Konstante	Beschreibung
<code>vbext_pk_Get</code>	Legt eine Prozedur fest, die den Wert einer Eigenschaft zurückgibt
<code>vbext_pk_Let</code>	Legt eine Prozedur fest, die einer Eigenschaft einen Wert zuweist
<code>vbext_pk_Set</code>	Legt eine Prozedur fest, die einen Verweis auf ein Objekt angibt
<code>vbext_pk_Proc</code>	Legt alle Prozeduren mit Ausnahme von Eigenschaften-Prozeduren fest

Tabelle 104: Konstanten zum Ermitteln von Prozeduren

Da Eigenschaften-Prozeduren mehrere Darstellungen im Modul haben können, müssen Sie die Art der zu suchenden Prozedur festlegen. Alle Prozeduren, abgesehen von Eigenschaften-Prozeduren (`Sub`- und `Function`-Prozeduren), verwenden `vbext_pk_Proc`.



Abbildung 379: VBA-Code im Meldungsfenster anzeigen

```
' =====
' Auf CD      Buchdaten\Beispiele\13_Gemischtes\
' Dateiname  13_03_VBProjectsAndModules.xls
' Modul      mdl_04_ReadCode
' =====

Sub ReadCode()
    Dim vbComp As VBIDE.VBComponent
    Dim strModName As String
    Dim strSubName As String
    Dim intStart As Integer
    Dim intCount As Integer

    strModName = "mdl_04_ReadCode"
    strSubName = "ReadCode"

    Set vbComp = _
        ThisWorkbook.VBProject.VBComponents(strModName)

    With vbComp.CodeModule
        intStart = .ProcBodyLine(strSubName, vbext_pk_Proc)
        intCount = .ProcCountLines(strSubName, vbext_pk_Proc)

        MsgBox Prompt:=.Lines(intStart, intCount), _
            Title:=.Name
    End With
End Sub
```

450 Blatt- und Codenamen einer Tabelle auslesen

In Microsoft Excel hat jedes Blatt neben dem Excel-Blattnamen auch einen VBA-Codenamen. Letzterer kann nur über die VBA-Entwicklungsumgebung verändert werden. Der Blattname kann sich vom Codenamen unterscheiden. Beim Öffnen einer neuen Excel-Arbeitsmappe oder beim Erstellen eines neuen Blattes stimmen Blattname und Codename überein TABELLE1 (TABELLE1). Die Änderung des Blattnamens wirkt sich jedoch nicht auf den Codenamen aus, und die Änderung des Codenamens, mit Hilfe des EIGENSCHAFTEN-Fensters im Visual-Basic-Editor, ändert nicht den Blattnamen.

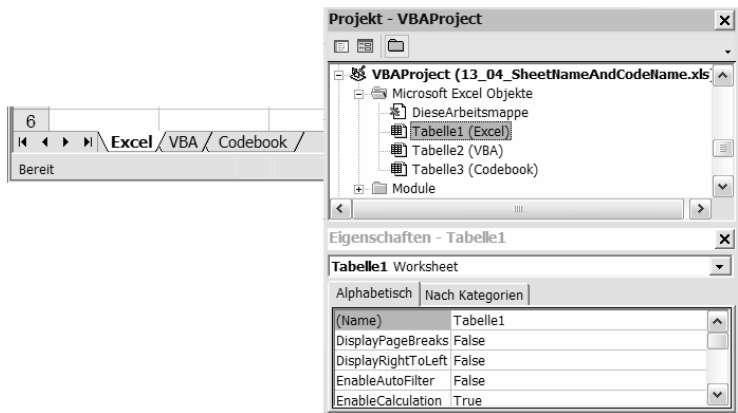


Abbildung 380: Excel- und VBA-Blattnamen

Grundlagen

Allgemein

Datum
Zeit

Formeln

Steuerelemente

Befehle
leisten

Objekte

Diagramme

Ereignisse

User-Formen

Web/Mail

Extern

Gemischtes

Spezial

Standardmäßig hat jede Arbeitsmappe ein Blatt TABELLE1 an erster Stelle. Dieses kann auf drei Arten angesprochen werden:

```
Worksheets(1).Select      ' über den Index ansprechen
Worksheets("Tabelle1").Select  ' über den Blattnamen ansprechen
Tabelle1.Select           ' über den Codenamen ansprechen
```

Folgende Prozedur zeigt den Index des aktiven Blattes in der Titelzeile eines Meldungsfensters an. Der Blatt- und Codename werden im Meldungstext bekannt gegeben:



Abbildung 381: Index, Blatt- und Codename des aktiven Blattes

```
'=====
' Auf CD      Buchdaten\Beispiele\13_Gemischtes\
' Dateiname  13_04_SheetNameAndCodeName.xls
' Modul      mdl_01_SheetNames
'=====
```

```
Sub SheetNames()
    With ActiveSheet
        MsgBox _
            "Blattname:" & .Name & Chr(10) & _
            "Codename: " & .CodeName, _
            Title:="Blatt " & .Index
    End With
End Sub
```

Wird nun das erste Blatt TABELLE1 von Hand oder mittels VBA in EXCEL umbenannt

```
Worksheets("Tabelle1").Name = "Excel"
```

funktioniert die `Select`-Methode über die `Index`-Eigenschaft noch, nicht jedoch die zweite Methode über den Blattnamen. Da sich der Codename aber nicht geändert hat, kann das Blatt über seinen Codenamen einwandfrei identifiziert werden.

Wenn das Blatt TABELLE1 verschoben wird, liefert die `Index`-Eigenschaft ein falsches Blatt, da das erste Blatt in der Registerfolge selektiert wird. Die anderen beiden Methoden funktionieren hingegen tadellos.

Als Fazit daraus sehen wir, dass die Verwendung des Codenamens im VBA-Programm eine Rücksichtnahme auf die entsprechenden Benutzeraktionen (Verschieben/Umbenennen) erspart. Einzig das Löschen des Blattes würde beim Versuch, das Blatt anzusprechen, eine Fehlermeldung auslösen.

451 Codennamen einer Tabelle ändern

Um den Code lesbarer zu machen, empfiehlt es sich, zu Beginn die Blätter mit sprechenden Namen auszustatten. Markieren Sie im Projekt-Explorer des Visual-Basic-Editors das gewünschte Blatt und ändern Sie die dazugehörige Eigenschaft (NAME) (siehe Abbildung 380). Der Codename muss mit einem Buchstaben beginnen und darf keine Leerzeichen enthalten. Wenn Sie einem Blatt einen benutzerdefinierten Codennamen, z.B. MEINBLATT geben, wird beim Kopieren dieses Blattes ein neuer Codename vergeben, der sich aber recht gut nützen lässt, da die Kopien die Codennamen MEINBLATT1, MEINBLATT2 usw. erhalten. Achten Sie bei der Namensvergabe darauf, dass die Namen kurz gehalten sind, da lange Namen Probleme bereiten können.

Den Codennamen eines Arbeitsblattes können Sie zum einen über den Blattnamen ändern. Damit Excel keine Fehlermeldung ausgibt, wenn das genannte Blatt nicht existiert, werden in einer For Each...Next-Anweisung alle Komponenten der Arbeitsmappe durchlaufen. Das Tabellenblatt EXCEL erhält den neuen Codennamen NEUERCODENAME1.

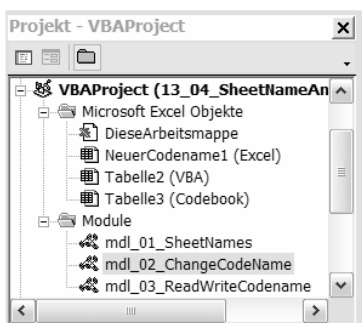


Abbildung 382: Geänderter Codename

```
' =====
' Auf CD      Buchdaten\Beispiele\13_Gemischtes\
' Dateiname  13_04_SheetNameAndCodeName.xls
' Modul      mdl_02_ChangeCodename
' =====
```

```
Sub ChangeCodenameOverSheetname()
    Dim objVBComp As Object

    For Each objVBComp In ThisWorkbook.VBProject.VBComponents

        If objVBComp.Properties("Name").Value = "Excel" Then
            objVBComp.Properties("_Codename").Value = "NeuerCodename1"
            Exit Sub
        End If

    Next

    MsgBox "Das Blatt mit dem Blattnamen ""Tabelle1"" existiert nicht"
End Sub
```

Ebenfalls können Sie den Codennamen eines Excel-Arbeitsblattes auch über den bekannten alten Codennamen TABELLE2 ändern. Zum Abfangen einer Fehlermeldung, falls das genannte Blatt nicht

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

existiert, wird in folgender Prozedur anstelle einer `For Each...Next`-Anweisung eine Fehleroutine mit der Sprungmarke `ERRORHANDLER` angewandt.

```
' =====
' Auf CD      Buchdaten\Beispiele\13_Gemischtes\
' Dateiname   13_04_SheetNameAndCodeName.xls
' Modul       mdl_02_ChangeCodeName
' =====

Sub ChangeCodenameOverCodename()
    On Error GoTo ERRORHANDLER

    ThisWorkbook.VBProject.VBComponents("Tabelle2") _
        .Properties("_CodeName").Value = "NeuerCodename2"

Exit Sub

ERRORHANDLER:
    MsgBox "Das Blatt mit dem Codenamen ""Tabelle2"" existiert nicht"
End Sub
```

Eine wesentliche Vereinfachung zum Auslesen und zur Änderung von Codenamen stellen wir Ihnen mit nachfolgender Funktion vor. Die Funktion `ReadWriteCodename` erwartet als erstes Argument `strSheetName` den Blattnamen eines Excel-Arbeitsblattes. Geben Sie im zweiten Argument `strCodeName` eine Zeichenfolge ein, wird diese als neuer Codename für das angegebene Blatt verwendet.

```
' =====
' Auf CD      Buchdaten\Beispiele\13_Gemischtes\
' Dateiname   13_04_SheetNameAndCodeName.xls
' Modul       mdl_03_ReadWriteCodename
' =====

Public Function ReadWriteCodename(strSheetName As String, _
                                   Optional strCodeName As String) _
                                   As String

    Dim intCount As Integer
    Dim objVBProj As Object

    On Error Resume Next
    Set objVBProj = ThisWorkbook.VBProject

    For intCount = 1 To objVBProj.VBComponents.Count
        With objVBProj.VBComponents(intCount)
            If .Properties("Name") = strSheetName Then
                ' Codename ändern
                If strCodeName <> "" Then
                    .Properties("_CodeName") = strCodeName
                End If
                ' Codename auslesen
                ReadWriteCodename = .Properties("_CodeName")
            End If
        End With
    Next intCount
End Function
```

```
End With
Next
End Function
```

Wenden Sie diese Funktion zum Auslesen bzw. Ändern von Codennamen in einer Prozedur oder im Direktfenster an. Folgende Anweisung im Direktfenster liefert den Codennamen des Blattes EXCEL. Schreiben Sie die Anweisung in das Direktfenster und bestätigen Sie die Eingabe mit der -Taste

```
?ReadWriteCodename("Excel")
```

Um den Codennamen eines Excel-Arbeitsblattes zu ändern, geben Sie nach dem Blattnamen den neuen Codennamen an.

```
?ReadWriteCodename("Excel","NeuerCodename")
```

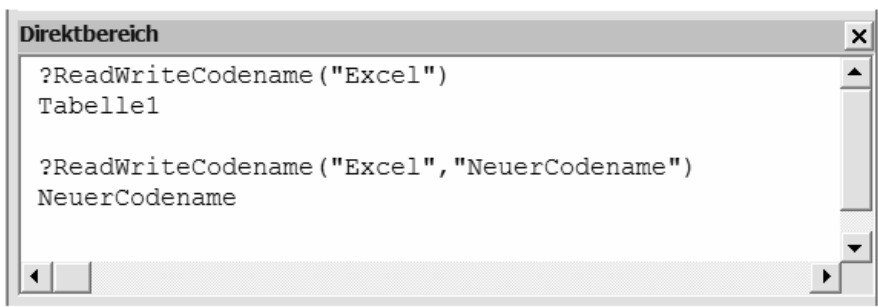


Abbildung 383: Codename über das Direktfenster ändern

Über die Prozedur ReadCodename erhalten Sie ein Meldungsfenster mit dem Namen und Codennamen des aktiven Tabellenblattes. Die Prozedur WriteCodename ändert den Codennamen entsprechend Ihrer Eingabe in der InputBox.

```
'=====
' Auf CD      Buchdaten\Beispiele\13_Gemischtes\
' Dateiname   13_04_SheetNameAndCodeName.xls
' Modul       mdl_03_ReadWriteCodename
'=====

Sub ReadCodename()
    With ActiveSheet
        MsgBox ReadWriteCodename(.Name), , _
            "Codename des Blattes " & .Name
    End With
End Sub

'=====
' Auf CD      Buchdaten\Beispiele\13_Gemischtes\
' Dateiname   13_04_SheetNameAndCodeName.xls
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

```
' Modul      mdl_03_ReadWriteCodename
'=====
Sub WriteCodename()
    Dim strInput As String

    With ActiveSheet
        strInput = InputBox( _
            "Geben Sie einen neuen Codenamen für " & .Name & " ein")
        If strInput = "" Then Exit Sub

        MsgBox ReadWriteCodename(.Name, strInput), , _
            "Codename des Blattes " & .Name
    End With
End Sub
```

452 Codenamen nach Tabellennamen benennen

Mit nachfolgender Prozedur erhalten alle Tabellenblätter der Beispieldatei den Codenamen nach ihrem jeweiligen Blattnamen. Zusätzlich wird dem Codenamen die Zeichenfolge CN_ vorangestellt. Das Tabellenblatt EXCEL bekommt zum Beispiel dadurch den Codenamen CN_EXCEL. Die Fehleroutine On Error Resume Next haben wir eingesetzt, falls ein Blattname Leerzeichen enthält, zu lang ist, oder mit einer Ziffer beginnt. Diese Faktoren würden bei der Vergabe eines neuen Codenamens zu einer Fehlermeldung führen.

```
'=====
' Auf CD      Buchdaten\Beispiele\13_Gemischtes\
' Dateiname   13_04_SheetNameAndCodeName.xls
' Modul      mdl_02_ChangeCodeName
'=====

Sub ChangeAllCodeNames()
    Dim objSh As Object

    On Error Resume Next
    For Each objSh In ThisWorkbook.Application.Sheets
        With ThisWorkbook.VBProject.VBComponents(objSh.CodeName)
            .Properties("_CodeName").Value = "CN_" & objSh.Name
        End With
    Next
End Sub
```

453 Modul erstellen

Um Ihrem VBA-Projekt eine neue Standardkomponente hinzuzufügen, gibt es von Hand mehrere Möglichkeiten. Wählen Sie in der MENÜLEISTE des VBA-Editors das Menü EINFÜGEN, um ein USERFORM, MODUL oder KLASSENMODUL in das markierte VBA-Projekt einzufügen. Im Kontextmenü des VBA-Projektes (Rechtsklick im Projekt-Explorer innerhalb des VBA-Projektes) finden Sie alle Aktionen, welche Sie hier ausführen können.

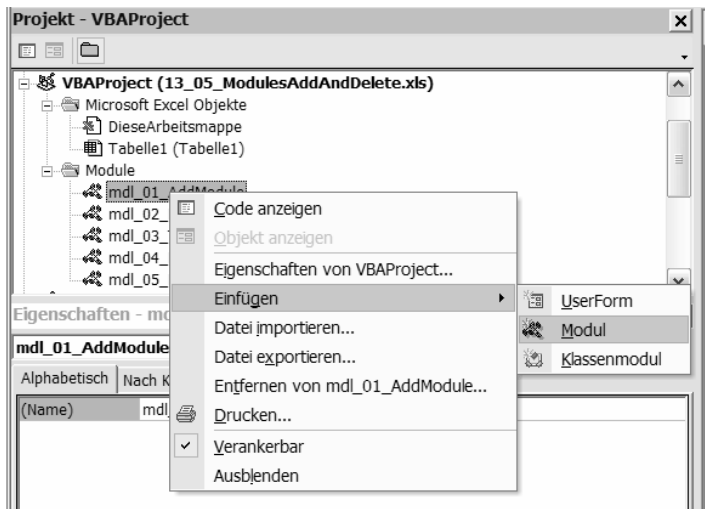


Abbildung 384: Kontextmenü des VBA-Projektes

Verwenden Sie in VBA die Add-Methode der VBComponents-Auflistung, um dem VBA-Projekt eine neue Komponente hinzuzufügen. Sie können eine der folgenden Komponenten verwenden:

Konstante	Beschreibung
Vbext_ct_StdModule	Fügt der Auflistung ein Standardmodul hinzu.
Vbext_ct_MSForm	Fügt der Auflistung ein Formular hinzu.
vbext_ct_ClassModule	Fügt der Auflistung ein Klassenmodul hinzu.

Tabelle 105: Neue Komponenten für das VBA-Projekt

Wenn Sie einem VBA-Projekt eine neues Standardmodul hinzufügen, erhält dieses zunächst den Namen `MODUL` mit fortlaufender Nummerierung, zum Beispiel `MODUL1`, `MODUL2` usw. Diese Vorgehensweise gilt ebenfalls für UserForms und Klassenmodule.

```
' =====
' Auf CD   Buchdaten\Beispiele\13_Gemischtes\
' Dateiname 13_05_ModulesAddAndDelete.xls
' Modul     mdl_01_AddModule
' =====

Sub AddModule()
    ActiveWorkbook.VBProject.VBComponents.Add (vbext_ct_StdModule)
End Sub
```

454 Modul mit Namensvergabe erstellen

Bei standardmäßiger Modulbenennung werden nur einstellige fortlaufende Nummerierungen vorgenommen. Wenn Sie das zehnte Modul `MODUL10` erstellen, reiht es sich im VBA-Projekt zwischen `MODUL1` und `MODUL2` ein. Folgende Prozedur zählt alle vorhandenen Standardmodule

Grund-lagen

Allge-mein

Datu-Zeit

tungen

Steuer-elemen

Befehl-leisten

Objekt

Dia-gramm

Ereig-nisse

User-Form

Web/Mail

Extern

Ge-mischtes

Specia

und vergibt für die neue Komponente eine fortlaufende zweistellige Nummerierung, MODUL01, MODUL02 usw. Zusätzlich wird die Zeichenkette NEWMODULE für die Namensvergabe vergeben.

```
' =====
' Auf CD      Buchdaten\Beispiele\13_Gemischtes\
' Dateiname  13_05_ModulesAddAndDelete.xls
' Modul      mdl_01_AddModule
' =====

Sub AddNumberModule()
    Dim VBComp As VBAComponent
    Dim intCount As Integer

    For Each VBComp In ThisWorkbook.VBProject.VBComponents
        If VBComp.Type = 1 Then
            intCount = intCount + 1
        End If
    Next VBComp

    Set VBComp = _
        ThisWorkbook.VBProject.VBComponents.Add(vbext_ct_StdModule)

    VBComp.Name = "mdl_" & Format(intCount, "00_") & "NewModule"
    Application.Visible = True
End Sub
```

455 Modul löschen

Wenn Sie mit der `Remove`-Methode ein Modul löschen wollen, welches in dem angegebenen VBA-Projekt nicht vorhanden ist, wird eine Fehlermeldung ausgegeben. Diese können Sie, wie in unserem Beispiel, mit einer Fehlerroutine auffangen.

Hinweis

In Rezept 456 stellen wir Ihnen eine Funktion vor, welche vorhandene Module eines VBA-Projektes überprüft. Diese Funktion eignet sich dazu, die in diesem Beispiel angewandte Fehlerroutine zu ersetzen.

Achtung

Es ist nicht möglich, das Codemodul DIESEARBEITSMAPPE oder Dokumentenmodule von Tabellen- und Diagrammblättern zu löschen. Diese sind feste Bestandteile des VBA-Projektes. Lediglich die Prozeduren bzw. der Text in den Codemodulen kann gelöscht werden.

```
' =====
' Auf CD      Buchdaten\Beispiele\13_Gemischtes\
' Dateiname  13_05_ModulesAddAndDelete.xls
' Modul      mdl_02_DeleteModule
' =====

Sub DeleteModule1()
    On Error Resume Next
    With ThisWorkbook.VBProject
```

```
        .VBComponents.Remove .VBComponents("mdl_03_Test_Delete")
    End With
End Sub
```

456 Vorhandene Module überprüfen

Bevor Sie ein Modul erstellen oder löschen, können Sie über die VBComponents-Auflistung und dem Namen der Komponente feststellen, ob die genannte Komponente im angegebenen VBA-Projekt vorhanden ist. Die Funktion ModuleExists liefert als Ergebnis den booleschen Wert True oder False.

In unserer Beispieldatei liefert der folgende Aufruf im Direktbereich den Wert Wahr:

```
?ModuleExists("mdl_04_FunctionModuleExists")    ' liefert Wahr

'=====
' Auf CD      Buchdaten\Beispiele\13_Gemischtes\
' Dateiname   13_05_ModulesAddAndDelete.xls
' Modul       mdl_04_FunctionModuleExists
'=====

Function ModuleExists(strModule As String) As Boolean
    Dim VBComp As VBIDE.VBComponent

    For Each VBComp In ThisWorkbook.VBProject.VBComponents
        If VBComp.Name = strModule Then
            ModuleExists = True
            Exit For
        End If
    Next VBComp
End Function
```

Über eine Prozedur rufen Sie die Funktion wie folgt auf:

```
'=====
' Auf CD      Buchdaten\Beispiele\13_Gemischtes\
' Dateiname   13_05_ModulesAddAndDelete.xls
' Modul       mdl_04_FunctionModuleExists
'=====

Sub FindModule()
    Debug.Print ModuleExists("mdl_01_AddModule")
End Sub
```

In unserem vorherigen Beispiel haben wir die Fehlerroutine On Error Resume Next vor dem Löschen eines Standardmoduls vorangestellt. Die Funktion ModuleExists löst diese Fehlerbehandlung ab.

```
'=====
' Auf CD      Buchdaten\Beispiele\13_Gemischtes\
```

Grundlagen

Allgemein

Datum und Zeit

Formeln und Berechnungen

Steuerungselemente

Befehlsliste

Objekte

Diagramme

Ereignisse

User-Formen

Web und Mail

Extern

Gemischtes

Spezial

```
' Dateiname 13_05_ModulesAddAndDelete.xls
' Modul      mdl_04_FunctionModuleExists
' =====

Sub DeleteModule2()
    Dim VBComp As VBComponent

    If ModuleExists("mdl_03_Test_Delete") = True Then
        Set VBComp = _
            ThisWorkbook.VBProject.VBComponents("mdl_03_Test_Delete")
    Else
        MsgBox "Das zu löschende Modul existiert nicht", vbExclamation
        Exit Sub
    End If

    ThisWorkbook.VBProject.VBComponents.Remove VBComp
End Sub
```

457 Vorhandene Prozeduren überprüfen

Um die Existenz einer Prozedur zu ermitteln, muss in folgender Funktion der Name des Moduls bekannt sein. Dieser wird beim Aufruf der Funktion `ProcedureExists` an das Argument `strModule` übergeben. Liefert die Funktion `ModuleExists` den Wert Wahr, wird über die `ProcStartLine`-Eigenschaft ermittelt, in welcher Zeile des Codemoduls die festgelegte Prozedur `strProcedure` beginnt. Wird der angegebene Prozedurname nicht gefunden, gibt die `ProcStartLine` den Wert 0 zurück und die Funktion `ProcedureExists` liefert dementsprechend den Wert False.

Hinweis

Wenn es sich bei der gesuchten Prozedur um die erste Prozedur eines Moduls handelt, beginnt sie am Ende des allgemeinen Deklarationsabschnitts. Weitere Prozeduren beginnen in der ersten Zeile unterhalb der `End Sub`-Anweisung der vorhergehenden Prozedur.

```
' =====
' Auf CD      Buchdaten\Beispiele\13_Gemischtes\
' Dateiname 13_05_ModulesAddAndDelete.xls
' Modul      mdl_05_FunctionProcedureExists
' =====

Function ProcedureExists(strModule As String, _
                        strProcedure As String) As Boolean
    On Error Resume Next
    If ModuleExists(strModule) = True Then
        ProcedureExists = _
            ThisWorkbook.VBProject.VBComponents(strModule) _
                .CodeModule.ProcStartLine(strProcedure, vbext_pk_Proc) _
                > 0
    End If
End Function
```

Im Direktfenster dieser Beispieldatei liefert folgende Anweisung den Wert Wahr:

```
?ProcedureExists("mdl_05_FunctionModuleExists", "ProcedureExists")
```


Über eine Prozedur rufen Sie die Funktion wie folgt auf:

```
'=====
' Auf CD      Buchdaten\Beispiele\13_Gemischtes\
' Dateiname  13_05_ModulesAddAndDelete.xls
' Modul      mdl_03_FunctionModuleExists
'=====

Sub FindProcedure()
    Debug.Print ProcedureExists("mdl_01_AddModule", "AddModule")
End Sub
```

458 Prozedur in ein Modul schreiben und aufrufen

Die folgende Prozedur `AddProcedure` fügt dem Modul `MDL_02_NEWPROCEDURE` das Makro `NewProcedure` hinzu. Über die `CountOfLines`-Eigenschaft werden die vorhandenen Zeilen im angegebenen Modul ermittelt. Die neue Prozedur `NewProcedure` wird eine Zeile unterhalb der Anweisung `Option Explicit` eingefügt.

Üblicherweise rufen Sie Prozeduren aus anderen Prozeduren über die `Call`-Anweisung auf, oder geben einfach den Namen der aufzurufenden Prozedur im Code an. Da das aufzurufende Makro `NewProcedure` zu Beginn der Prozedur `AddProcedure` noch nicht existiert, würde Excel eine Fehlermeldung beim Kompilieren des VBA-Projektes ausgeben. Um die neue Prozedur nach deren Erstellung aufzurufen, muss die `Run`-Methode angewandt werden.

Hinweis

Wenn die einzufügende Prozedur bereits besteht, gibt Excel eine Fehlermeldung aus. Dies können Sie umgehen, indem vorab die Existenz dieser Prozedur überprüft wird. Wenden Sie dazu die Funktion `ProcedureExists` aus dem vorherigen Beispiel an.

```
'=====
' Auf CD      Buchdaten\Beispiele\13_Gemischtes\
' Dateiname  13_06_ProceduresAddAndDelete.xls
' Modul      mdl_01_AddProcedure
'=====

Sub AddProcedure()
    Dim strText As String
    Dim VBCodeMod As CodeModule

    strText = _
        "Sub NewProcedure" & Chr(13) & _
        "    MsgBox ""Hallo Codebook-Leser"" & Chr(13) & _
        "End sub"

    Set VBCodeMod = _
        ThisWorkbook.VBProject.VBComponents("mdl_02_NewProcedure") _
        .CodeModule

    With VBCodeMod
        .InsertLines .CountOfLines + 1, strText
    End With
End Sub
```

Grundlagen

Allgemein

Datum
Zeit

Formeln
Funktionen

Steuerungselemente

Befehle
Leistungen

Objekte

Diagramme

Ereignisse

User-Formen

Web/
Mail

Extern

Gemischtes

Spezial

End With

Application.Run "NewProcedure"

End Sub

459 Prozedur in eigenes Modul schreiben und aufrufen

Im Gegensatz zu dem vorherigen Beispiel wird nachfolgend die neue Prozedur `MyNewProcedure` in das gleiche Codemodul `MDL_03_ADDPROCINMYMODULE` wie die ausführende Prozedur `AddProcInMyModule` eingefügt. Um das neue Makro im Anschluss aufzurufen, versagt hier allerdings die `Run`-Methode, da das Modul während des Prozedurablaufes nicht kompiliert werden kann.

Um das Modul erneut zu kompilieren, wird die Kontrolle zunächst an Excel übergeben. Dies wird über die `OnTime`-Methode erreicht. Es ist nicht notwendig, eine Wartezeit anzugeben. Allein die Übergabe an Excel und zurück zum VBE ist ausreichend, um das Modul zu kompilieren. Im Anschluss daran wird das Modul wieder geladen und die in der `OnTime`-Methode angegebene Prozedur `MyNewProcedure` ausgeführt.

```
'=====
' Auf CD      Buchdaten\Beispiele\13_Gemischtes\
' Dateiname  13_06_ProceduresAddAndDelete.xls
' Modul      mdl_03_AddProcInMyModule
'=====

Sub AddProcInMyModule()
    Dim strText As String
    Dim VBCodeMod As CodeModule

    strText = _
        "Sub MyNewProcedure" & Chr(13) & _
        "    MsgBox ""Hier ist die neue Prozedur."" & vbcr & _" & _
        "    & Chr(13) & _
        "    ""Eingefügt am:" & "" & Date & "" & Chr(13) & _
        "End sub"

    Set VBCodeMod = ThisWorkbook.VBProject.VBComponents _
        ("mdl_03_AddProcInMyModule").CodeModule

    With VBCodeMod
        .InsertLines .CountOfLines + 2, strText
    End With

    Application.OnTime Now, "MyNewProcedure"
End Sub
```

460 Ereignis-Prozedur erstellen

Über die `CreateEventProc`-Methode des `CodeModule`-Objektes wird eine Ereignis-Prozedur erstellt. Diese stehen in DIESEARBEITSMAPPE, Dokumenten- (Arbeitsblätter) und Klassenmodulen zur Verfügung. Wenn im linken Auswahlfeld oberhalb des Codebereichs der Eintrag `WORKBOOK` ausgewählt wird, werden im rechten Auswahlfeld alle verfügbaren Ereignisse, welche die Arbeits-

mappe betreffen, angezeigt. Die `CreateEventProc`-Methode erwartet als erstes Argument das Ereignis, zum Beispiel `Open`, und als zweites Argument das Objekt, zum Beispiel `Workbook`.

In folgender Prozedur wird die Komponente `DIESEARBEITSMAPPE` an eine `With`-Anweisung übergeben. In diese Komponente wird die Codeschablone der genannten Ereignis-Prozedur in die erste freie Zeile eingetragen. Sie enthält alle erforderlichen Deklarationen und Parameter des angegebenen Ereignisses. Die Zeilennummer der Aufrufzeile im Codebereich plus 1 Zeile wird in der Variablen `lngStartLine` gespeichert. Dieser Wert dient als Startlinie zum Einfügen des Codes in die neue Ereignis-Prozedur.

Die `InsertLines`-Methode fügt eine oder mehrere Codezeilen an einer bestimmten Stelle in einen Codeblock ein. Durch die Angabe von `lngStartLine` wird der Code eine Zeile unterhalb der Aufrufzeile eingefügt.

```
' =====
' Auf CD      Buchdaten\Beispiele\13_Gemischtes\
' Dateiname  13_06_ProceduresAddAndDelete.xls
' Modul      mdl_04_AddEvent
' =====

Sub AddWorkbookEvent()
    Dim lngStartLine As Long

    With ThisWorkbook.VBProject.VBComponents("DieseArbeitsmappe") _
        .CodeModule

        ' Codeschablone einfügen
        lngStartLine = .CreateEventProc("Open", "Workbook") + 1

        ' Code in Ereignis-Prozedur einfügen
        .InsertLines lngStartLine, _
            "    MsgBox "Hallo Codebook Leser", vbInformation+vbOKOnly"

    End With
End Sub
```

Um einem Arbeitsblatt eine Ereignis-Prozedur hinzuzufügen, schreiben Sie in der `With`-Anweisung den Codenamen des betreffenden Tabellen- oder Diagrammblattes. In der `CreateEventProc`-Methode wählen Sie ein Ereignis das Tabellenblatt betreffend, zum Beispiel `SelectionChange`, und geben als Objekt `Worksheet` an.

Die Syntax des einzufügenden Codetextes haben wir in unserem Beispiel in kleinen Buchstaben geschrieben.

```
msgbox target.address(0,0),vbinformation+vbokonly
```

Wenn ein Text in Anführungszeichen dargestellt werden soll, muss dieser in jeweils drei folgenden Anführungszeichen eingeschlossen werden. Nachdem die Prozedur erstellt wurde, sehen Sie im Codemodul des Tabellenblattes die korrekte Ausschreibung der Syntax. Excel hat den Codetext selbstständig korrigiert. Daran sehen Sie, dass die Syntax einwandfrei war und erkannt wurde.

Grundlagen
Allgemein
Datum und Zeit
Formeln
Diagramme
Diagramm
Ereignisse
User-Form
Web/Mail
Extern
Ge-mischtes
Spezial

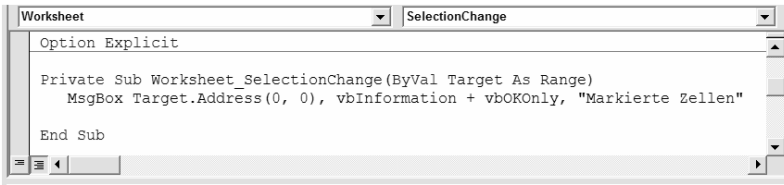


Abbildung 385: Korrigierte Ausschreibung der Syntax

```
'=====
' Auf CD      Buchdaten\Beispiele\13_Gemischtes\
' Dateiname   13_06_ProceduresAddAndDelete.xls
' Modul       mdl_04_AddEvent
'=====
```

```
Sub AddWorksheetEvent()
    Dim lngStartLine As Long

    With ThisWorkbook.VBProject.VBComponents _
        (Worksheets(1).CodeName).CodeModule

        lngStartLine = _
            .CreateEventProc("SelectionChange", "Worksheet") + 1

        .InsertLines lngStartLine, _
            "    msgbox target.address(0,0),vbinformation+vbokonly," & _
            """"Markierte Zellen""""

    End With
End Sub
```

461 Prozeduren löschen

Um einzelne Prozeduren eines Moduls zu löschen, setzen Sie zunächst einen Verweis auf die betreffende Komponente, in welcher die Prozedur hinterlegt ist. Über die `ProcStartLine`-Eigenschaft ermitteln Sie die Zeile, an der die festgelegte Prozedur beginnt. In der Variablen `lngCountLines` wird die Anzahl der Zeilen in der festgelegten Prozedur über die `ProcCountLines`-Eigenschaft ermittelt und gespeichert. Jetzt sind die Startzeile und die Anzahl der Zeilen bekannt, so dass mit der `DeleteLines`-Methode exakt die angegebene Prozedur gelöscht wird.

Hinweis

Wenn die zu löschende Prozedur im genannten Modul nicht vorhanden ist, gibt Excel eine Fehlermeldung aus. Dies können Sie umgehen, indem vorab die Existenz dieser Prozedur überprüft wird. Wenden Sie dazu die Funktion `ProcedureExists` aus dem Rezept 457 an.

```
'=====
' Auf CD      Buchdaten\Beispiele\13_Gemischtes\
' Dateiname   13_06_ProceduresAddAndDelete.xls
' Modul       mdl_05_DeleteProcedure
'=====
```

```
Sub DeleteProcedure()  
    Dim VBCodeMod As CodeModule  
    Dim lngStartLine As Long  
    Dim lngCountLines As Long  
  
    Set VBCodeMod = ThisWorkbook.VBProject.VBComponents _  
        ("mdl_02_NewProcedure").CodeModule  
  
    With VBCodeMod  
        lngStartLine = _  
            .ProcStartLine("NewProcedure", vbext_pk_Proc)  
  
        lngCountLines = _  
            .ProcCountLines("NewProcedure", vbext_pk_Proc)  
  
        .DeleteLines lngStartLine, lngCountLines  
    End With  
End Sub
```

Ereignis-Prozeduren löschen Sie, indem Sie einen Verweis auf das entsprechende Modul setzen und die genaue Bezeichnung des Ereignisses verwenden.

```
Set VBCodeMod = _  
ThisWorkbook.VBProject.VBComponents("DieseArbeitsmappe").CodeModule  
lngStartLine = .ProcStartLine("Worksheet_Open", vbext_pk_Proc)  
...
```

Um Ereignis-Prozeduren in Dokumentenmodulen der Arbeitsblätter zu löschen, setzen Sie einen Verweis auf die Komponente, indem Sie deren Codenamen verwenden:

```
Set VBCodeMod =  
    ThisWorkbook.VBProject.VBComponents(Worksheets(1).CodeName)  
    .CodeModule  
lngStartLine = _  
    .ProcStartLine("Worksheet_SelectionChange", vbext_pk_Proc)  
...
```

462 Code eines Moduls löschen

Um den Inhalt eines Moduls komplett zu löschen, verwenden Sie die `DeleteLines`-Methode. Geben Sie als Startzeile die erste Zeile an. Über die `CountOfLines`-Eigenschaft wird die Anzahl der Codezeilen im angegebenen Modul ermittelt.

```
'=====
```

' Auf CD Buchdaten\Beispiele\13_Gemischtes\
' Dateiname 13_06_ProceduresAddAndDelete.xls
' Modul mdl_06_DeleteModulCode

'=====

```
  
Sub DeleteModulCode()
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

```

With ThisWorkbook.VBProject.VBComponents _
    ("DieseArbeitsmappe").CodeModule
    .DeleteLines 1, .CountOfLines
End With
End Sub

```

463 Menü in VBE-Menüleiste erstellen

In der Kategorie »Befehlsleisten« finden Sie ausführliche Informationen, wie man in der Excel-Umgebung neue Symbolleisten, Menüs und Schaltflächen erstellt. Im VBA-Editor VBE ist dies ebenfalls möglich. Im folgenden Beispiel wird in der Menüleiste der Entwicklungsumgebung vor dem HILFE-Menü ein neues Menü CODEBOOK eingerichtet. Dieses enthält zwei Schaltflächen mit jeweils hinterlegten Prozeduren.

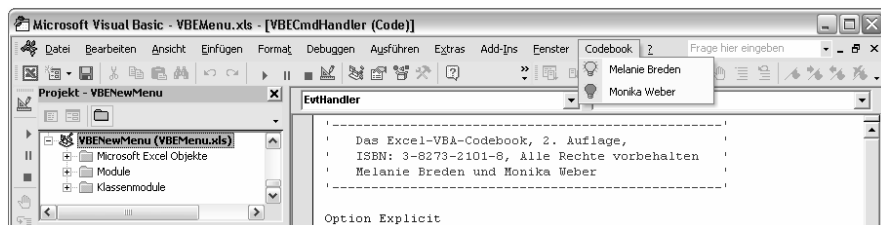


Abbildung 386: Neues Menü in der VBE-Menüleiste

Um benutzerdefinierte Steuerelemente in die Befehlsleisten der Entwicklungsumgebung zu integrieren, sind zwei Schritte nötig. Zum einen müssen die Steuerelemente mit ihren Beschriftungen und weiteren Eigenschaften erstellt werden. In der Excel-Umgebung wird die `OnAction`-Eigenschaft dazu verwendet, bei einem Schaltflächenklick die hinterlegte Prozedur aufzurufen. Im VBE muss dazu ein neues Ereignis `EvtHandler_Click` definiert werden. Dieses wird allerdings in einem Klassenmodul gespeichert, ein gewöhnliches Modul reicht dafür nicht aus.

Fügen Sie zunächst dem VBA-Projekt Ihrer Excel-Arbeitsmappe über **EINFÜGEN | KLASSENMODUL** ein neues Klassenmodul hinzu. Dieses erhält standardmäßig den Namen `KLASSE` mit fortlaufender Nummerierung, zum Beispiel `KLASSE1`. Es ist unbedingt erforderlich, den Klassennamen zu ändern, da im Modul `MDL_01_ADDNEWMENU` die Variable `mnuEvt` auf Basis dieser Klasse dimensioniert wird. Es könnte sonst zu Konflikten mit anderen vorhandenen Klassen kommen, da diese per Default automatisch erzeugte Namen tragen. Markieren Sie dazu das Klassenmodul und ändern Sie im Eigenschaftsfenster den Namen `Klasse1` in `VBECmdHandler` (siehe Abbildung 387). Wenn das Eigenschaftsfenster nicht sichtbar ist, zeigen Sie es über **ANSICHT | EIGENSCHAFTSFENSTER** oder die Tasten **[F4]** an.

Im Klassenmodul `VBECmdHandler` wird die öffentliche Variable `EvtHandler` vom Typ `CommandBarEvents` der `VBIDE`-Bibliothek deklariert. Das Schlüsselwort `WithEvents` gibt an, dass `EvtHandler` eine Objektvariable ist, die verwendet wird, um auf Ereignisse zu reagieren, die von einem `ActiveX`-Objekt ausgelöst wurden. Weitere Erläuterungen entnehmen Sie bitte dem kommentierten Code.

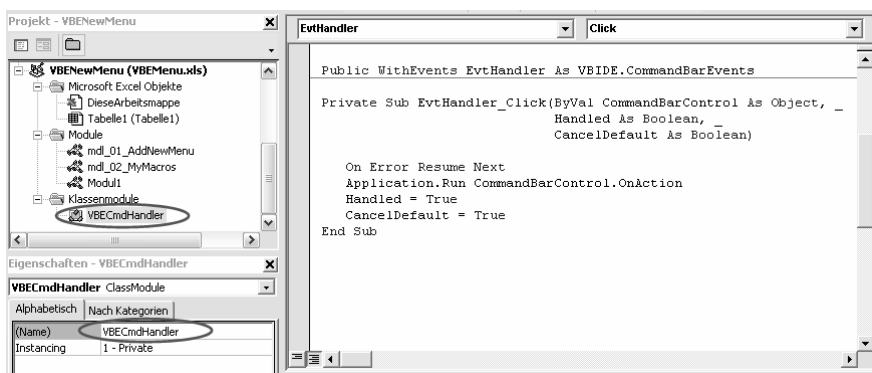


Abbildung 387: Klassenmodul mit angepasster Namensvergabe

```

' =====
' Auf CD      Buchdaten\Beispiele\13_Gemischtes\
' Dateiname   13_07_VBEMenu.xls
' Klasse      VBECmdHandler
' =====

Public WithEvents EvtHandler As VBA.CommandBarEvents

Private Sub EvtHandler_Click(ByVal CommandBarControl As Object, _
    Handled As Boolean, _
    CancelDefault As Boolean)

    ' Fehlerroutine, falls Prozedur nicht existiert
    On Error Resume Next

    ' Aufruf der hinterlegten Prozedur der angeklickten Schaltfläche
    Application.Run CommandBarControl.OnAction

    ' Ereignis wurde erfolgreich ausgeführt
    Handled = True
    CancelDefault = True
End Sub

```

Kommen wir nun zur Erstellung der neuen Steuerelemente und deren Handhabung. In einem Standardmodul werden zunächst die benötigten Variablen vor der ersten Prozedur `AddNewMenu` mit `Private` deklariert. Dadurch stehen sie allen Prozeduren dieses Moduls zur Verfügung, können aber nicht von anderen Modulen oder aus der Excel-Umgebung über **EXTRAS | MAKRO | MAKROS** heraus aufgerufen werden.

Damit bei einem Klick auf eine Schaltfläche des neuen Menüs **CODEBOOK** das `EvtHandler_Click`-Ereignis ausgelöst wird, muss zunächst eine öffentliche Variable `mnuEvt` der Objektklasse `VBECmdHandler` definiert werden. Wie Sie der Abbildung 388 entnehmen können, muss der Datentyp der Deklaration exakt mit der Bezeichnung der Objektklasse übereinstimmen, damit eine Zuordnung stattfinden kann.

Grundlagen

Allgemein

Datenzeit

tungen

Steuerelemente

Befehle leisten

Objekt

Diagramm

Ereignisse

User-Form

Web/Mail

Extern

Gemischtes

Spezial

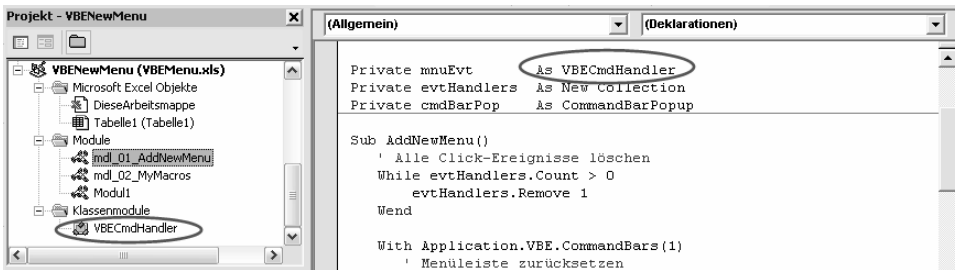


Abbildung 388: Variable einer Ereignisklasse definieren

Die Deklaration der Variablen `evtHandlers` als ein neues `Collection`-Objekt ermöglicht die Definition eigener Auflistungen. Dies dient in der Prozedur dazu, neue Objekte in Form von Schaltflächen `cmdBarBtn` in einer Auflistung zu speichern. Neue Objekte werden über die `ADD`-Methode hinzugefügt. Über die `Count`-Eigenschaft wird ermittelt, wie viele Objekte sich im `Collection`-Objekt befinden, welche dann mit der `Remove`-Methode entfernt werden. Wenn die Prozedur `AddNewMenu` aufgerufen wird, stellen wir so sicher, dass sich kein Objekt mehr in der `Collection`-Auflistung befindet. Ein Klick auf eine Schaltfläche würde sonst die hinterlegte Prozedur so oft aufrufen, wie sich das Objekt in der Auflistung befindet.

```
' =====
' Auf CD      Buchdaten\Beispiele\13_Gemischtes\
' Dateiname   13_07_VBEMenu.xls
' Modul       mdl_01_AddNewMenu
' =====
```

Option Private Module

```
Private mnuEvt As VBECmdHandler
Private evtHandlers As New Collection
Private cmdBarPop As CommandBarPopup
```

```
Sub AddNewMenu()
    ' Alle Click-Ereignisse löschen
    While evtHandlers.Count > 0
        evtHandlers.Remove 1
    Wend

    With Application.VBE.CommandBars(1)
        ' Menüleiste zurücksetzen
        .Reset
        ' Neues Menü in Menüleiste
        Set cmdBarPop = .Controls.Add( _
            Type:=msoControlPopup, _
            Before:=(.Controls.Count), _
            Temporary:=True)
        cmdBarPop.Caption = "Codebook"
    End With

    ' Erste Schaltfläche in Menü Codebook
    AddButton 351, "Melanie Breden", "Macro_Melanie"
```



```
' Zweite Schaltfläche in Menü Codebook
AddButton 352, "Monika Weber", "Macro_Monika"
End Sub
```

Die folgende Prozedur `AddButton(intFaceId, strCaption, strOnAction)` erstellt die Schaltflächen im Menü `CODEBOOK` der `VBE-MENÜLEISTE`. Sie erwartet zwingend die Angabe der `FaceId`-Nummer sowie die Bezeichnungen für die Schaltflächenbeschriftung `Caption` und den Prozedurnamen `OnAction`, der bei Klick auf die Schaltfläche ausgeführt werden soll. Diese werden beim Aufruf der Prozedur in den Variablen `intFaceId`, `strCaption` und `strOnAction` hinterlegt und an die Prozedur `AddButton` übergeben.

Nachdem dem Menü `CODEBOOK`, durch `cmdBarPop` dargestellt, eine neue Schaltfläche hinzugefügt wurde, wird mit der Anweisung

```
Set mnuEvt = New VBECmdHandler
```

eine neue Instanz innerhalb des `VBECmdHandler`-Objektes angelegt.

Anschließend wird mit der `Set`-Anweisung ein Verweis auf das gerade eingefügte Steuerelement, als ein neues Ereignis der `VBECmdHandler`-Klasse, gesetzt. Über die `Add`-Methode des `Collection`-Objektes wird die neue Instanz zur Auflistung des `evtHandlers`-Objektes hinzugefügt. Die Verbindung zwischen dem Schaltflächenklick und dem Aufrufen des `EvtHandler_Click`-Ereignisses ist somit hergestellt.

```
' =====
' Auf CD      Buchdaten\Beispiele\13_Gemischtes\
' Dateiname   13_07_VBEMenu.xls
' Modul       mdl_01_AddNewMenu
' =====
```

```
Sub AddButton(intFaceId As Integer, _
              strCaption As String, _
              strOnAction As String)

    Dim cmdBarBtn As CommandBarButton

    Set cmdBarBtn = cmdBarPop.Controls.Add
    With cmdBarBtn
        .Style = msoButtonIconAndCaption
        .FaceId = intFaceId
        .Caption = strCaption
        .OnAction = "'" & ThisWorkbook.Name & "'" & strOnAction
    End With
```

```
' Neue Instanz des VBECmdHandler-Objektes
Set mnuEvt = New VBECmdHandler
' Verbindung zwischen cmdBarBtn und Click-Ereignis herstellen
' In der Excel-Umgebung reicht das OnAction-Ereignis dafür aus
Set mnuEvt.EvtHandler = _
    Application.VBE.Events.CommandBarEvents(cmdBarBtn)
```

Grundlagen

Allgemein

Datum und Zeit

Formeln

Steuerelemente

Befehle

Objekte

Diagramme

Ereignisse

User-Formen

Web und Mail

Extern

Gemischtes

Spezial

```

' Neues Click-Ereignis in EvtHandler-Collection speichern
evtHandlers.Add mnuEvt
End Sub

```

Wenn Sie auf eine Schaltfläche im Menü CODEBOOK klicken, wird die jeweils hinterlegte Prozedur aufgerufen. Diese zeigen zunächst über die `SendKeys`-Anweisung das Direktfenster an und schreiben über die `Print`-Methode des `Debug`-Objektes einen Text in das Direktfenster. Durch die Anweisung `Option Private Module` sind diese Prozeduren in der Excel-Umgebung unter EXTRAS | MAKRO | MAKROS nicht in der Auflistung der verfügbaren Makros enthalten und können nur über die Schaltflächen im Menü CODEBOOK der VBE-MENÜLEISTE aufgerufen werden.

```

' =====
' Auf CD      Buchdaten\Beispiele\13_Gemischtes\
' Dateiname   13_07_VBEMenu.xls
' Modul       mdl_02_MyMacros
' =====

Option Private Module

Sub Macro_Melanie()
    ' Direktfenster anzeigen
    Application.VBE.Windows("Direktbereich").Visible = True
    Debug.Print "Melanie Breden " & Date
End Sub

Sub Macro_Monika()
    Application.VBE.Windows("Direktbereich").Visible = True
    Debug.Print "Monika Weber " & Date
End Sub

```

Wenn Sie diese Beispieldatei beenden, wird im `Workbook_Open`-Ereignis die Menüleiste der Entwicklungsumgebung zurückgesetzt und somit das Menü CODEBOOK gelöscht. Würden Sie bei geschlossener Beispieldatei auf eine Schaltfläche des benutzerdefinierten Menüs klicken, würde Excel die aufgerufene Prozedur nicht finden und eine Fehlermeldung ausgeben.

```

' =====
' Auf CD      Buchdaten\Beispiele\13_Gemischtes\
' Dateiname   13_07_VBEMenu.xls
' Ereignis    DieseArbeitsmappe
' =====

Private Sub Workbook_BeforeClose(Cancel As Boolean)
    Application.VBE.CommandBars(1).Reset
End Sub

```

464 Ereignis-Prozeduren für integrierte Ereignisse erstellen

Ereignis-Prozeduren sind Ihnen im Verlauf dieses Buches bereits an verschiedenen Stellen begegnet. In der Kategorie »Ereignisse« werden Ereignisse, welche das Arbeitsblatt bzw. die Arbeitsmappe betreffen, ausführlich behandelt.

In der Kategorie »UserForm« finden Sie ebenfalls Ereignisse, welche verwendet werden, um zum Beispiel auf das Laden der UserForm oder das Anklicken eines CommandButtons zu reagieren. Diese Ereignisse werden im Klassenmodul des jeweiligen Formulars verwaltet.

Alle diese Ereignisse haben eines gemeinsam: Sie besitzen jeweils ein eigenes Klassenmodul, worin die Ereignis-Prozeduren gespeichert werden. Beispielsweise kann das Worksheet_Activate-Ereignis in jedem Dokumentmodul vorhanden sein. Wenn der Anwender von einem Tabellenblatt zum anderen wechselt, wird das Worksheet_Activate-Ereignis des aktivierten Tabellenblattes ausgelöst.

Jede UserForm in einem Formularmodul weist ebenfalls eine eigene Instanz (Klassenmodul) auf. Das Initialize-Ereignis, welches beim Laden einer UserForm auftritt, wird nur in der Instanz ausgelöst, deren UserForm geladen wurde.

Diese Ereignisse lassen sich einfach erstellen. Öffnen Sie das entsprechende Klassenmodul und wählen Sie in der linken Dropdownliste OBJEKT den Eintrag WORKBOOK bzw. WORKSHEET. In einem Formular wählen Sie in der Objektliste die USERFORM selbst bzw. ein auf der UserForm vorhandenes Steuerelement. In der rechten DropDown-Liste PROZEDUR erstellen Sie die Code-schablone einer Ereignis-Prozedur durch einen Klick auf ein Ereignis in der Liste. Die Abbildung 389 zeigt dies am Beispiel eines Formular-Klassenmoduls.

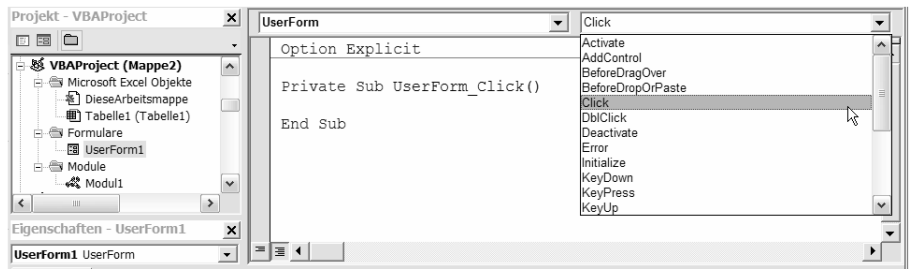


Abbildung 389: Integrierte Ereignisse einer UserForm

Excel kennt jedoch noch mehr Ereignisse, als in den bereits genannten Prozedurlisten der verschiedenen Klassenmodule zur Verfügung stehen. Das Application-Objekt, welches die gesamte Microsoft Excel-Anwendung darstellt, verfügt in der Version Excel 2002 über weitere 24 Ereignisse. Diese Zahl kann in vorherigen und künftigen Excel-Versionen abweichen. In der Online-Hilfe von Excel finden Sie unter dem Stichwort EREIGNISSE DES APPLICATION-OBJEKTS bzw. APPLICATION-OBJEKT Register EREIGNISSE alle in Ihrer Excel-Version zur Verfügung stehenden Application-Ereignisse mit weiteren Informationen. Ebenso listet der Objektbrowser im Visual-Basic-Editor von Microsoft Excel alle Ereignisse des Application-Objektes auf.

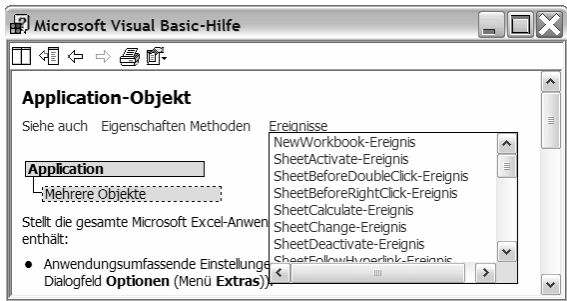


Abbildung 390: Integrierte Ereignisse in Excel 2002

Grundlagen

Allgemein

Datum
Zeit

ungen

Steuer
elemente

Befehl
leisten

Objekt

Diagramm

Ereignisse

User-
Form

Web/
Mail

Extern

Ge-
misch-
tes

Specia

Application-Ereignisse treten ein, wenn eine Arbeitsmappe erstellt oder geöffnet wird, wenn sich ein beliebiges Blatt in einer beliebigen geöffneten Arbeitsmappe ändert oder wenn eine Pivot-Table erstellt bzw. geöffnet wird.

Im Gegensatz zu den Ereignissen in den von Excel bereitgestellten Klassenmodulen, z.B. DIESEARBEITSMAPPE, ist diesen Application-Ereignissen kein eigenständiges Modul zugeordnet. Um Ereignis-Prozeduren für das Application-Objekt zu schreiben, müssen Sie unter Verwendung des Schlüsselwortes `WithEvents` in einem Klassenmodul ein neues Objekt erstellen.

Fügen Sie über das Menü EINFÜGEN | KLASSENMODUL ein neues Klassenmodul in Ihr VBA-Projekt ein. Zur Identifizierung der neuen Klasse ist es unbedingt erforderlich, dem Klassenmodul einen benutzerdefinierten, aussagekräftigen Namen zu geben. In unserem Beispiel haben wir im Eigenschaftsfenster die Name-Eigenschaft des neu eingefügten Klassenmoduls von KLASSE1 in CLSEVENT geändert.

Um anzugeben, dass eine Instanz eines Objektes auf Ereignisse reagieren soll, muss eine Objektvariable des betreffenden Typs mit dem Schlüsselwort `WithEvents` im neu eingefügten Klassenmodul CLSEVENT deklariert werden.

Das Schlüsselwort `WithEvents` gibt an, dass `x1App` eine Objektvariable ist, die verwendet wird, um auf Ereignisse vom Typ `Application` zu reagieren.

```
'=====
' Auf CD      Buchdaten\Beispiele\13_Gemischtes\
' Dateiname   13_08_ClassEventsBuiltIn.xls
' Klasse      clsEvent
'=====
```

```
Public WithEvents x1App As Application
```

Hinweis

In Excel-Versionen vor Excel 2000 wurde dem Datentyp `Application` der Objektkennzeichner `Excel` vorangestellt. Dies betrifft sowohl Modul- und Globalvariablen als auch Deklarierungen in der Aufrufzeile von Ereignis-Prozeduren.

Ohne Objektkennzeichner gibt die `Application`-Eigenschaft in Microsoft Excel ein `Application`-Objekt zurück, das die Anwendung »Microsoft Excel« repräsentiert. Mit Objektkennzeichner hingegen gibt diese Eigenschaft ein `Application`-Objekt zurück, das die Erstellungsanwendung des angegebenen Objekts repräsentiert. (Diese Eigenschaft eignet sich zur Ermittlung der zu einem OLE-Automatisierungsobjekt gehörigen Anwendung.)

Die Deklaration `Excel.Application` bedeutet also wörtlich `Excel.Microsoft Excel`. Unsere verwendeten Variablen und Verweise beziehen sich auf die eigene Anwendung »Excel«. Demnach können wir auf den Objektkennzeichner `Excel` verzichten.

Folgende Deklarierungen ohne Objektkennzeichner:

```
Public WithEvents x1App As Application
Private Sub x1App_NewWorkbook(ByVal Wb As Workbook)
```

und mit Objektkennzeichner:

```
Public WithEvents x1App As Excel.Application
Private Sub x1App_NewWorkbook(ByVal Wb As Excel.Workbook)
```

Beide Schreibarten sind jeweils in den von uns getesteten Excel-Versionen (Excel 97 bis Excel 2003) lauffähig.

Nachdem das neue Objekt `xlApp` mit Ereignissen deklariert wurde, wird es im Klassenmodul `CLSEVENT` im DropDown-Listenfeld `OBJEKT` angezeigt (Abbildung ■13.19).



Abbildung 391: Neues Element im Objektfeld

Wählen Sie im Feld `OBJEKT` das neue Objekt `xlApp` aus, damit die gültigen Ereignisse für dieses Objekt im DropDown-Listenfeld `PROZEDUR` angezeigt werden (siehe Abbildung 392). Klicken Sie im Feld `PROZEDUR` auf ein Ereignis, um dessen Codegerüst mitsamt den erforderlichen Deklarationen und Argumenten im Codefenster des Klassenmoduls `CLSEVENT` zu erstellen.

Im Objektkatalog wird das neue Element `xlApp` sowie die eingefügten Ereignis-Prozeduren als Elemente der Klasse `CLSEVENT` der Beispielmappe angezeigt.

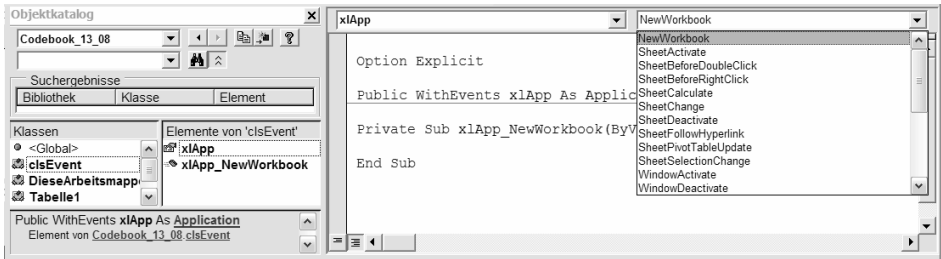


Abbildung 392: Integrierte Ereignisse des Application-Objektes im Prozedurfeld

Hinweis

Die integrierten Ereignisse des `xlApp`-Objektes stehen nur im Klassenmodul `clsEvent` zur Verfügung. Es können aber beliebig viele verschiedene Ereignisse in das Codefenster von `CLSEVENT` eingefügt werden.

Schreiben Sie die Anweisungen, welche beim Erstellen einer neuen Excel-Arbeitsmappe ausgeführt werden sollen, zwischen die Codeschablone.

Das Klassenmodul weist nun folgenden Code auf:

```
'=====
' Auf CD   Buchdaten\Beispiele\13_Gemischtes\
' Dateiname 13_08_ClassEventsBuiltIn.xls
' Klasse    clsEvent
'=====

Public WithEvents xlApp As Application
```

Grundlagen
Allgemein
Datum
Zeit
tungen
Steuer
elementen
Befehl
leisten
Objekt
Diagramm
Ereignisse
User-Form
Web/Mail
Extern
Gemischtes
Spezial

```
Private Sub xlApp_NewWorkbook(ByVal Wb As Workbook)
    MsgBox "Neue Datei '" & Wb.Name & "' wurde erstellt."
End Sub
```

Nun haben Sie die Objektvariable `xlApp` mit zugeordneten Ereignissen und eine Ereignis-Prozedur erstellt. Da `xlApp` noch keinen Wert auf die aktuelle Instanz der Anwendung enthält, müssen Sie ihr mit der `Set`-Anweisung einen Verweis zuweisen. Diesen Verweis setzen wir in der `Initialize`-Ereignis-Prozedur des Klassenmoduls `CLSEVENT`. Dieses Ereignis tritt ein, nachdem das Objekt geladen wurde, also beim Öffnen der Excel-Arbeitsmappe. Damit wird sichergestellt, dass der Verweis bei Bedarf zur Verfügung steht.

Der Verweis kann ebenso im `Workbook_Open`-Ereignis der Arbeitsmappe oder in einer Sub-Prozedur in einem beliebigen Standardmodul erfolgen. Sie müssen nur sicherstellen, dass er bei Bedarf zur Verfügung steht.

```
' =====
' Auf CD      Buchdaten\Beispiele\13_Gemischtes\
' Dateiname   13_08_ClassEventsBuiltIn.xls
' Klasse      clsEvent
' =====

Private Sub Class_Initialize()
    Set xlApp = Application
End Sub
```

Hinweis

Es ist nicht unbedingt erforderlich, den Verweis auf `xlApp` bei Beendigung der Excel-Arbeitsmappe zurückzusetzen. Dieser wird dann automatisch aus dem Arbeitsspeicher entfernt. Um zum Beispiel nur das Auslösen der Ereignis-Prozeduren des Klassenmoduls in der laufenden Anwendung zu unterdrücken, setzen Sie den Verweis auf `xlApp` zurück. Die Verbindung der Objektvariablen `xlApp` zum zugehörigen `Application`-Objekt (Microsoft Excel) heben Sie in einer `Set`-Anweisung über das Schlüsselwort `Nothing` auf.

```
Set xlApp = Nothing
```

Wenn Sie die `EnableEvents`-Eigenschaft auf `False` setzen, werden sowohl die Standardereignisse aus `DIESEARBEITSMAPPE` und den `DOKUMENTENMODULEN` als auch die `CLSEVENT`-Ereignisse deaktiviert. Mit `Application.EnableEvents = True` aktivieren Sie wieder alle Ereignisse in Microsoft Excel. Für die `CLSEVENT`-Ereignisse muss dann kein erneuter Verweis gesetzt werden.

Damit das `Initialize`-Ereignis in `CLSEVENT` ausgelöst wird, muss eine neue Instanz der `CLSEVENT`-Klasse erstellt werden. Dazu ist es notwendig, eine private Objektvariable `pEvents` auf Modulebene in einem Standardmodul zu deklarieren. Die Deklaration in einem Standardmodul ist zwingend notwendig, damit diese ereignisbereite `Application`-Objektvariable im Arbeitsspeicher verbleibt, bis sie zurückgesetzt oder das Projekt geschlossen wird. Da der Name der Klasse `CLSEVENT` lautet, wird die Objektvariable `pEvents` vom Typ `CLSEVENT` deklariert.

In der anschließenden Prozedur `INITIALIZECLSEVENT` wird mit der `Set`-Anweisung und dem Schlüsselwort `New` eine neue Instanz der Klasse `CLSEVENT` erstellt. Nachdem dieser Verweis ausgeführt wurde, reagiert Excel auf die Ereignisse, welche in `CLSEVENT` hinterlegt sind, und führt den hinterlegten Code aus.

```
'=====
' Auf CD      Buchdaten\Beispiele\13_Gemischtes\
' Dateiname  13_08_ClassEventsBuiltIn.xls
' Modul      mdl_01_InitClsEvent
'=====

Private pEvents As clsEvent

Public Sub InitializeClsEvent()
    ' Application-Ereignisse zur Verfügung stellen
    Set pEvents = New clsEvent
End Sub
```

Hinweis

Ereignisse für das Application-Objekt werden nicht dauerhaft gespeichert. Damit das Application-Objekt ereignisbereit wird, muss die Prozedur InitializeClsEvent bzw. der Verweis Set pEvents = New clsEvent bei jedem Öffnen von Excel ausgeführt werden.

Rufen Sie im Workbook_Open-Ereignis der Beispiel-Arbeitsmappe die Prozedur InitializeClsEvent auf, damit die in CLSEVENT hinterlegten Application-Ereignisse in allen geöffneten Excel-Arbeitsmappen überwacht werden.

```
'=====
' Auf CD      Buchdaten\Beispiele\13_Gemischtes\
' Dateiname  13_08_ClassEventsBuiltIn.xls
' Ereignis    DieseArbeitsmappe
'=====

Private Sub Workbook_Open()
    InitializeClsEvent
End Sub
```

465 Meldungen über alle Zellpositions-Änderungen

Wenn Sie die Zellposition in einem bestimmten Tabellenblatt überwachen möchten, verwenden Sie dafür das Worksheet_SelectionChange-Ereignis des Klassenmoduls der betreffenden TABELLE.

```
Private Sub Worksheet_SelectionChange(ByVal Target As Range)

Um jede Änderung der Zellposition in allen Tabellenblättern »einer« Excel-Arbeitsmappe zu überwachen, greifen Sie auf das Workbook_SheetSelectionChange-Ereignis in DIESEARBEITSMAPPE zu.

Private Sub Workbook_SheetSelectionChange(ByVal Sh As Object, _
                                           ByVal Target As Range)
```

Jetzt stellt sich die Frage, wie man jede Änderung der Zellposition auf jedem Tabellenblatt in allen geöffneten Excel-Arbeitsmappen überwachen kann. Nachdem Sie das vorherige Beispiel gelesen haben, ist dies nun kein Problem mehr. Verwenden Sie dazu das Application_SheetSelectionChange-Ereignis im Klassenmodul CLSEVENT.

Grund-lagen

Allge-mein

Datu Zeit

tungen

Steuer elemen

Befehl leisten

Objekt

Dia-gramm

Ereig-nisse

User-Form

Web/ Mail

Extern

Ge-misch tes

Specia

```
Private Sub xlApp_SheetSelectionChange(ByVal Sh As Object, _
                                       ByVal Target As Range)
```

Die notwendigen Variablen-Deklarierungen und Verweise in dieser Beispielmappe wurden aus dem vorherigen Beispiel übernommen. Sie sind für alle Application-Ereignisse gleichlautend. Die benutzerdefinierten Variablennamen sind frei wählbar. Um Sie nicht zu verwirren, haben wir die Namen CLSEVENT (für die neue Klasse), xlApp (für das Ereignisobjekt) und pEvents (zur Initialisierung einer neuen Klasseninstanz) nicht geändert.

In der Beispielmappe wurden alle drei vorgenannten SheetSelectionChange-Ereignisse in den verschiedenen Modulen hinterlegt. Wenn Sie sich in der Beispielmappe befinden, wird zuerst das Ereignis der Tabelle, anschließend das der Arbeitsmappe und zuletzt das Application-Ereignis aufgerufen. In anderen geöffneten Excel-Arbeitsmappen wird nur das Application-Ereignis ausgelöst. Bei jeder Änderung der Zellposition erscheint ein Meldungsfenster mit Angabe des Datei- und Tabellenblattnamens sowie der Zelladresse und Wert der ersten markierten Zelle.

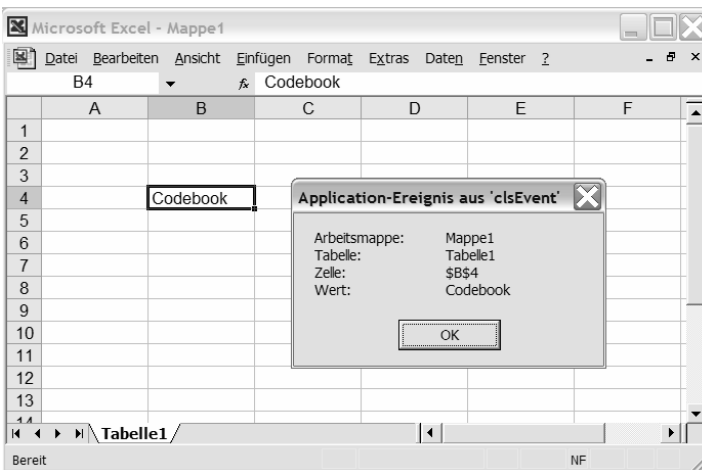


Abbildung 393: Application-Ereignis SheetSelectionChange

```
' =====
' Auf CD      Buchdaten\Beispiele\13_Gemischtes\
' Dateiname   13_09_xlApp_SheetSelectionChange.xls
' Ereignis    DieseArbeitsmappe
' =====
```

```
Private Sub Workbook_Open()
    Set pEvents = New clsEvent
End Sub
```

```
Private Sub Workbook_SheetSelectionChange(ByVal Sh As Object, _
                                           ByVal Target As Range)
```

```
    MsgBox _
        "Arbeitsmappe:" & vbTab & Me.Name & vbNewLine & _
        "Tabelle:      " & vbTab & Sh.Name & vbNewLine & _
```

```
"Zelle:      " & vbTab & vbTab & Target.Address & vbNewLine & _
"Wert:      " & vbTab & Target.Cells(1).Value, , _
"Ereignis aus 'DieseArbeitsmappe'"
End Sub

Private Sub Worksheet_SelectionChange(ByVal Target As Range)

    MsgBox _
    "Arbeitsmappe: " & vbTab & Me.Parent.Name & vbNewLine & _
    "Tabelle:     " & vbTab & Me.Name & vbNewLine & _
    "Zelle:      " & vbTab & vbTab & Target.Address & vbNewLine & _
    "Wert:      " & vbTab & Target.Cells(1).Value, , _
    "Ereignis aus 'Tabelle'"
End Sub

Public pEvents As clsEvent

Public WithEvents xlApp As Application

Private Sub Class_Initialize()
    Set xlApp = Application
End Sub

Private Sub xlApp_SheetSelectionChange(ByVal Sh As Object, _
                                       ByVal Target As Range)

    MsgBox _
    "Arbeitsmappe: " & vbTab & Sh.Parent.Name & vbNewLine & _
    "Tabelle:     " & vbTab & Sh.Name & vbNewLine & _
    "Zelle:      " & vbTab & vbTab & Target.Address & vbNewLine & _
    "Wert:      " & vbTab & Target.Cells(1).Value, , _
    "Application-Ereignis aus 'clsEvent'"
End Sub
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

466 Add-Ins

Add-Ins sind Excel-Dateien mit der Dateiendung *.*xla*. Einige Add-Ins werden von Microsoft bereits bei der Excel-Installation mitgeliefert und befinden sich im Verzeichnis C:\Programme\Microsoft Office\Office10\Makro.

Add-Ins dienen dazu, den Funktionsumfang von Excel zu erweitern. Sie können zum Beispiel Excel/VBA-Anwendungen oder eine Sammlung benutzerdefinierter Tabellen- oder VBA-Funktionen als Add-In speichern und dadurch in allen Excel-Arbeitsmappen zur Verfügung stellen.

Tipp

Um den Code Ihres Add-Ins zu schützen, sperren Sie das VBA-Projekt im Visual-Basic-Editor über das Menü EXTRAS | EIGENSCHAFTEN VON VBAPROJEKT Register SCHUTZ und vergeben ein Passwort. Der Schutz wird erst aktiv, wenn die Datei gespeichert und die Excel-Anwendung erneut geöffnet wird.

Ein Add-In erzeugen Sie durch Änderung des Excel-Dateityps einer bereits fertig gestellten Excel-Arbeitsmappe. Speichern Sie eine normale *.xls-Datei, die die Funktionen und Programme, die Sie bereitstellen wollen, enthält als *.xla-Datei ab. Wählen Sie dazu in der Excel-Ansicht im Menü DATEI | SPEICHERN UNTER in der Liste DATEITYP den untersten Eintrag *Microsoft-Excel Add-In (*.xla)* aus. Als Standardspeicherpfad schlägt Excel das Verzeichnis *C:\Dokumente und Einstellungen\<Benutzername>\Anwendungsdaten\Microsoft\AddIns* vor. Sie können jedoch auch einen beliebigen Speicherort wählen.

Wenn eine Excel-Arbeitsmappe als *.xla-Datei gespeichert wird, wird die *IsAddin*-Eigenschaft der Datei automatisch auf *True* eingestellt. Diese Einstellung ist das Hauptmerkmal eines Add-Ins.

Add-Ins und Excel-Arbeitsmappen, deren *IsAddin*-Eigenschaft auf *True* eingestellt ist, unterscheiden sich durch folgende Merkmale von normalen Excel-Arbeitsmappen:

- ▶ Das Arbeitsmappenfenster und vorhandene Tabellen- und Diagrammblätter sind nicht sichtbar.
- ▶ Sie werden nicht aufgefordert, die Arbeitsmappe zu speichern, falls Änderungen am VBA-Code vorgenommen werden, während die Arbeitsmappe geöffnet ist.
- ▶ Eventuell in der Arbeitsmappe enthaltene Makros werden nicht im Menü EXTRAS | MAKRO | MAKROS dargestellt.
- ▶ Makros der Arbeitsmappe können, obwohl sie nicht dargestellt werden, über das Makro-Dialogfeld ausgeführt werden. Geben Sie dazu den Prozedurnamen in der Zeile MAKRONAME: ein und klicken Sie auf AUSFÜHREN.
- ▶ Prozedurnamen müssen nicht mit dem Namen der Arbeitsmappe aufgerufen werden.
- ▶ Wenn Sie beim Öffnen der Arbeitsmappe die -Taste gedrückt halten, hat dies keine Wirkung.

Sie können auch in normalen *.xls-Dateien die *IsAddin*-Eigenschaft auf *True* setzen, um diesen Dateien alle Eigenschaften eines Add-Ins zuzuweisen. Stellen Sie dazu die *IsAddin*-Eigenschaft manuell im Eigenschaftsfenster von DIESEARBEITSMAPPE auf *True* (siehe Abbildung 394). Der Excel-Dateityp wird dabei allerdings nicht geändert.

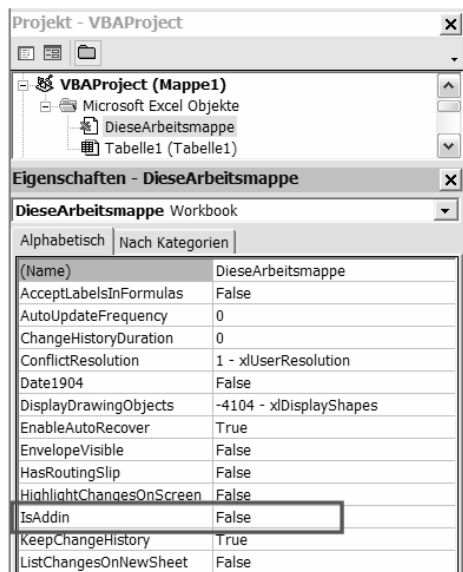


Abbildung 394: Excel-Arbeitsmappe Add-In deklarieren

Der Vorteil dabei liegt darin, dass die *.xls-Arbeitsmappe dann nicht in der Excel-Ansicht angezeigt wird. Nutzen Sie diese Eigenschaft, um Excel-Arbeitsmappen nur mit Makrounterstützung bearbeitbar zu machen. Setzen Sie dazu im Workbook_BeforeClose-Ereignis die IsAddin-Eigenschaft auf True und speichern Sie im Anschluss die Datei. Beim nächsten Öffnen der Arbeitsmappe mit Makrounterstützung erfolgt dann im Workbook_Open-Ereignis wieder die Aufhebung als Add-In durch die Anweisung ThisWorkbook.IsAddin = False.

Wird die Datei ohne Makrounterstützung oder mit gedrückter -Taste geöffnet, bleibt die IsAddin-Eigenschaft auf True und die Arbeitsmappe ist für den Excel-Anwender unbrauchbar. Sie können sie nur noch im Visual-Basic-Editor bearbeiten und dort im Eigenschaftsfenster von DIESEARBEITSMAPPE die IsAddin-Eigenschaft manuell auf False setzen. Dadurch werden die Arbeitsblätter wieder in der Excel-Anwendung sichtbar. Vorhandene Makros in dieser Datei werden durch diesen Vorgang jedoch nicht aktiviert.

467 Add-Ins hinzufügen und installieren

Sie stellen die Funktionen und Programme eines Add-Ins zur Verfügung, indem Sie es wie eine normale Excel-Arbeitsmappe über DATEI | ÖFFNEN oder mit einem Doppelklick auf die Datei im Explorer in Excel laden. Allerdings wird das Add-In bei der nächsten Excel-Sitzung dann nicht automatisch mitgeladen. Dazu muss das Add-In im ADD-INS-Manager im Menü EXTRAS | ADD-INS in der Liste der verfügbaren Add-Ins aufgelistet und installiert sein (siehe Abbildung 395).



Abbildung 395: Liste der verfügbaren und installierten Add-Ins

Über die Add-Methode fügen Sie der Liste der verfügbaren Add-Ins ein Add-In hinzu, installieren es jedoch nicht. Bei der Add-Methode müssen Sie den Dateinamen des Add-Ins verwenden und nicht den Titel, welcher in der Liste der verfügbaren Add-Ins angezeigt wird. Wenn das Add-In nicht im Microsoft-Verzeichnis gespeichert ist, müssen Sie dem Dateinamen den gesamten Pfad voranstellen.

Grundlagen

Allgemein

Datum
Zeit

Formeln

Steuerelemente

Befehle
leisten

Objekte

Diagramm

Ereignisse

User-Form

Web/
Mail

Extern

Gemischtes

Spezial

```
'=====
' Auf CD      Buchdaten\Beispiele\13_Gemischtes\
' Dateiname   13_10_CodebookAddIn.xla
' Modul       mdl_01_AddAndInstall
'=====

Sub AddAddInWithPath()
    AddIns.Add Filename:="C:\Eigene Dateien\MyAddin.xla"
End Sub
```

Wenn sich die Prozedur mit der Add-Methode in der eigenen Datei, also im Codebereich des Add-In befindet, verwenden Sie die `ThisWorkbook`-Eigenschaft zusammen mit der `FullName`-Eigenschaft. Diese Syntax gibt den Namen der Datei einschließlich Pfadangabe zurück, in der das aktuelle Makro ausgeführt wird.

```
'=====
' Auf CD      Buchdaten\Beispiele\13_Gemischtes\
' Dateiname   13_10_CodebookAddIn.xla
' Modul       mdl_01_AddAndInstall
'=====

Sub AddAddInFullname()
    AddIns.Add Filename:=ThisWorkbook.FullName
End Sub
```

Um das Add-In zu installieren, können Sie die `Installed`-Eigenschaft gleich an die Add-Methode anschließen.

```
'=====
' Auf CD      Buchdaten\Beispiele\13_Gemischtes\
' Dateiname   13_10_CodebookAddIn.xla
' Modul       mdl_01_AddAndInstall
'=====

Sub AddAndInstallAddIn()
    AddIns.Add(Filename:=ThisWorkbook.FullName).Installed = True
End Sub
```

Tipp

Excel gibt keine Fehlermeldungen aus, wenn Sie versuchen, ein Add-In zu der Liste der verfügbaren Add-Ins hinzuzufügen, welches bereits vorhanden ist. Ebenso bringt das Installieren eines bereits installierten Add-Ins keine Fehlermeldung. Dementsprechend können Sie ruhigen Gewissens beim Installieren die Add-Methode mit der `Installed`-Eigenschaft verbinden. Diese Vorgehensweise hat weiterhin den Vorteil, dass Sie nicht erst prüfen müssen, ob das Add-In über einen Titel verfügt.

Wenn Sie ein Add-In installieren wollen, welches bereits in der Liste der verfügbaren Add-Ins vorhanden ist, müssen Sie in dem `AddIn`-Objekt den Titel des Add-Ins angeben. Wurde kein Titel angegeben, verwendet Excel den Dateinamen ohne Dateiendung als Titel. Wenn Sie nicht genau

wissen, ob das Add-In einen Titel enthält, prüfen Sie dies vorab in einer If...Then...Else-Anweisung, bevor das Add-In installiert wird.

```
'=====
' Auf CD      Buchdaten\Beispiele\13_Gemischtes\
' Dateiname  13_10_CodebookAddIn.xla
' Modul      mdl_01_AddAndInstall
'=====

Sub InstallAddInTitle()
    Dim strAddIn As String

    strAddIn = ThisWorkbook.BuiltinDocumentProperties("Title")

    If strAddIn = "" Then
        strAddIn = Left( _
            ThisWorkbook.Name, Len(ThisWorkbook.Name) - 4)
    End If
    AddIns(strAddIn).Installed = True
End Sub
```

468 Add-In von Diskette auf Festplatte kopieren

Die Add-Methode der Add-Ins-Auflistung weist neben dem Parameter `Filename` einen zweiten Parameter `CopyFile` vom Typ `Boolean` auf. Falls sich das Add-In auf einem auswechselbaren Medium (Diskette oder CD) befindet und Sie den `CopyFile`-Wert auf `True` setzen, wird das Add-In auf die Festplatte in den Ordner `<Benutzername>\Anwendungsdaten\Microsoft\AddIns` kopiert.

Ist der `CopyFile`-Wert `False`, wird das Add-In nicht kopiert, sondern nur der Liste der verfügbaren Add-Ins hinzugefügt. Wird dieses Argument nicht angegeben, fragt Excel vor Ausführung des Befehls über ein Dialogfenster, ob die Datei kopiert werden soll.

Die folgende Prozedur kopiert die Add-In-Datei `MyAddIn.xla` vom Diskettenlaufwerk auf die Festplatte.

```
'=====
' Auf CD      Buchdaten\Beispiele\13_Gemischtes\
' Dateiname  13_10_CodebookAddIn.xla
' Modul      mdl_02_CopyAddIn
'=====

Sub CopyAddIn()
    AddIns.Add Filename:="A:\MyAddIn.xla", CopyFile:=True
End Sub
```

Die Prozedur `CopyAndInstallAddIn` kopiert zunächst das Add-In von der Diskette auf die Festplatte, fügt es der Liste der verfügbaren Add-Ins hinzu und installiert es zugleich.

Grundlagen

Allgemein

Datum
Zeit

Formeln

Steuerelemente

Befehle
Anweisungen

Objekte

Diagramme

Ereignisse

User-Formen

Web/
Mail

Extern

Gemischtes

Spezial

```
'=====
' Auf CD      Buchdaten\Beispiele\13_Gemischtes\
' Dateiname   13_10_CodebookAddIn.xla
' Modul       mdl_02_CopyAddIn
'=====

Sub CopyAndInstallAddIn()
    AddIns.Add(Filename:="A:\MyAddin.xla", _
               CopyFile:=True).Installed = True
End Sub
```

469 Add-Ins auflisten

Über das Menü EXTRAS | ADD-INS rufen Sie den ADD-INS-Manager auf (siehe Abbildung 395). In dieser Liste werden die Titel der Add-Ins angezeigt. Wenn im Eigenschaftsfenster unter DATEI | EIGENSCHAFTEN Register ZUSAMMENFASSUNG kein Titel angegeben wurde, wird der Dateiname des Add-Ins ohne Dateiendung angezeigt. Anhand des vorangestellten Kontrollkästchens sehen Sie, welche Add-Ins installiert sind und in der aktuellen Excel-Anwendung zur Verfügung stehen.

Was Ihnen in der Ansicht fehlt, sind die Dateiname sowie Speicherpfade, in welchen die Add-Ins auf Ihrer Festplatte gespeichert sind. Folgende Prozedur erstellt in der aktiven Arbeitsmappe ein neues Tabellenblatt und führt alle in der Liste aufgeführten Add-Ins mit ihrem Titel, Dateinamen, Pfad und Installationsstatus auf.

```
'=====
' Auf CD      Buchdaten\Beispiele\13_Gemischtes\
' Dateiname   13_10_CodebookAddIn.xla
' Modul       mdl_03_AddInList
'=====

Sub AddInList()
    Dim intI As Integer

    With Worksheets.Add
        .Rows(1).Font.Bold = True

        .Range("A1:D1").Value = _
            Array("Titel", "Name", "Pfad", "Installiert")

        For intI = 1 To AddIns.Count
            .Cells(intI + 1, 1) = AddIns(intI).Title
            .Cells(intI + 1, 2) = AddIns(intI).Name
            .Cells(intI + 1, 3) = AddIns(intI).Path
            .Cells(intI + 1, 4) = AddIns(intI).Installed
        Next

        .Columns.AutoFit
    End With
End Sub
```

470 Add-Ins über Explorer installieren

Wenn Sie eine Add-In-Datei über das Menü DATEI | ÖFFNEN oder im Explorer mit einem Doppelklick auf eine *.xla-Datei starten, wird diese in der aktuellen Excel-Anwendung geladen. Durch diesen Startvorgang wird die Add-In-Datei zwar geöffnet und alle Funktionen des Add-Ins stehen in allen geöffneten Arbeitsmappen zur Verfügung. Dabei wird das Add-In aber nicht zu der Liste der verfügbaren Add-Ins hinzugefügt und dementsprechend auch nicht installiert. Wenn Sie Excel schließen, steht dieses Add-In beim nächsten Excel-Start nicht mehr zur Verfügung, da es weder hinzugefügt noch installiert wurde.

Um in der Excel-Ansicht oder per VBA auf den Add-In-Manager unter EXTRAS | ADD-INS zuzugreifen, muss mindestens eine Excel-Arbeitsmappe geöffnet und ein Tabellenblatt sichtbar sein. Der Menüpunkt ist sonst ausgegraut und nicht anwählbar. Bevor Sie mit VBA eine Add-In-Datei der Liste der verfügbaren Add-Ins hinzufügen und installieren, muss ermittelt werden, ob die nötigen beschriebenen Voraussetzungen gegeben sind. Diese Prüfung führen wir mit der Funktion IsWorkbookOpen durch, welche in der Prozedur InstallAddIn aufgerufen und ausgewertet wird.

Hinweis

Die Funktion IsWorkbookOpen soll nur über VBA-Prozeduren aufgerufen und nicht im Funktions-Assistenten angezeigt und als Tabellenformel angewandt werden. Dementsprechend deklarieren wir das Modul, in welchem die Funktion gespeichert ist, mit der Option Private-Anweisung. Mit Option Private Module stehen Teile der Modulebenen, die nicht als Private deklariert wurden, anderen Modulen in dem Projekt zur Verfügung, anderen Projekten oder Anwendungen jedoch nicht.

```
' =====
' Auf CD      Buchdaten\Beispiele\13_Gemischtes\
' Dateiname   13_10_CodebookAddIn.xla
' Modul       mdl_04_InstallAddIn
' =====

Option Private Module

Function IsWorkbookOpen() As Boolean
    Dim intWb As Integer

    ' Wenn keine Datei geöffnet, bleibt Funktion auf False
    If Workbooks.Count = 0 Then Exit Function

    For intWb = 1 To Workbooks.Count
        ' Prüfung, ob geladene Excel-Datei sichtbar ist
        If Windows(Workbooks(intWb).Name).Visible = True Then
            IsWorkbookOpen = True
            Exit Function
        End If
    Next intWb
End Function
```

Die Vorgehensweise zur Installation eines Add-Ins beim Öffnen einer *.xla-Datei entnehmen Sie bitte den Kommentaren im folgenden Code.

Grundlagen
Allgemein
Datum und Zeit
Formeln
Diagramme
Objekte
Diagramm
Ereignisse
User-Form
Web/Mail
Extern
Gemischtes
Spezial

```
'=====
' Auf CD      Buchdaten\Beispiele\13_Gemischtes\
' Dateiname   13_10_CodebookAddIn.xla
' Modul       mdl_04_InstallAddIn
'=====
```

Option Private Module

```
Sub InstallAddIn()
    Dim blnInstalled As Boolean
    Dim WbDummy As Workbook
    Dim strAddIn As String

    ' Add-In Titel auslesen
    strAddIn = ThisWorkbook.BuiltinDocumentProperties("Title")

    ' Wenn kein Titel angegeben, wird Dateiname ohne Dateiendung
    ' als Titel verwendet
    If strAddIn = "" Then strAddIn = Left( _
        ThisWorkbook.Name, Len(ThisWorkbook.Name) - 4)

    ' Einlesen der Installed-Eigenschaft des AddIns in eine
    ' boolesche Variable
    On Error Resume Next
    blnInstalled = AddIns(strAddIn).Installed
    On Error GoTo 0

    ' Wenn Add-In bereits installiert ist, beende Prozedur
    If blnInstalled Then Exit Sub

    ' Rückfrage, ob Add-In installiert werden soll
    If MsgBox( _
        "Soll diese Arbeitsmappe als Add-In installiert werden?", _
        vbQuestion + vbYesNo, ThisWorkbook.Name) _
        = vbNo Then Exit Sub

    ' Warteschleife, um vorherige MsgBox zu schließen
    ' Ansonsten werden beide Meldungsfenster übereinander angezeigt
    Application.Wait (Now + TimeValue("00:00:01"))

    ' Ereignisse und Bildschirmaktualisierung ausschalten
    With Application
        .EnableEvents = False
        .ScreenUpdating = False
    End With

    ' wenn keine sichtbare Datei geöffnet ist, erstelle eine Datei
    If IsWorkbookOpen = False Then Set WbDummy = Workbooks.Add

    ' Add-In der Liste hinzufügen
    AddIns.Add Filename:=ThisWorkbook.FullName
```

```
' Add-In installieren
AddIns(strAddIn).Installed = True

' Wenn Datei erstellt wurde, schlieÙe diese ohne zu speichern
If Not WbDummy Is Nothing Then WbDummy.Close _
                                SaveChanges:=False

MsgBox _
    "Das Add-In wurde erfolgreich installiert.", , strAddIn

With Application
    .EnableEvents = True
    .ScreenUpdating = True
End With
End Sub
```

Im Workbook_Open-Ereignis erfolgt beim Öffnen der Add-In-Datei der Prozeduraufruf InstallAdd-In, um die Datei als Add-In zu installieren.

471 Add-In deinstallieren und aus Liste entfernen

Der Eintrag im ADD-INS-Manager bleibt auch nach dem Deaktivieren, also dem Entfernen des Häkchens, weiter erhalten.

Über den ADD-INS-Dialog unter EXTRAS | ADD-INS können Sie Add-Ins deaktivieren, indem Sie das Häkchen vor einem Eintrag entfernen. Die Datei wird dabei geschlossen und steht im Visual-Basic-Editor nicht mehr in der aktuellen Excel-Anwendung zur Verfügung. Auch bei einem Excel-Neustart bleibt der Eintrag in der Liste der verfügbaren Add-Ins bestehen. Sie können über den ADD-INS-Dialog also nur den Installationsstatus der Add-Ins steuern.

Folgende Schritte sind notwendig, um einen Eintrag aus der Liste der verfügbaren Add-Ins zu entfernen:

1. Schließen Sie Microsoft Excel
2. Löschen oder verschieben Sie das Add-In im Explorer oder benennen Sie es um
3. Starten Sie Microsoft Excel erneut
4. Microsoft Excel zeigt eine Fehlermeldung, dass das verschobene bzw. umbenannte Add-In nicht gefunden wurde
5. Versuchen Sie, das Add-In im Add-Ins-Manager zu aktivieren
6. Excel gibt daraufhin eine Fehlermeldung aus und fragt, ob das Add-In aus der Liste der verfügbaren Add-Ins gelöscht werden soll (siehe Abbildung 396)
7. Bestätigen Sie diesen Dialog mit JA, damit das Add-In aus der Liste entfernt wird
8. Beenden Sie den Add-Ins-Dialog über die Schaltfläche OK

Dieser Vorgang lässt sich teilweise mit VBA automatisieren. Der einzige Nachteil ist der, dass Excel keine Fehlermeldung ausgibt, wenn Sie versuchen, per VBA ein Add-In zu installieren, welches in der Liste der Add-Ins aufgeführt ist, dessen Name aber falsch ist. Dazu müssen Sie nach wie vor den Add-Ins-Dialog aufrufen und den Eintrag manuell anklicken. Erst dann fragt Excel, ob das nicht gefundene Add-In entfernt werden soll.

Grund-lagen

Allge-mein

Datu Zeit

tungen

Steuer elemen

Befehl leisten

Objekt

Dia-gramm

Ereig-nisse

User-Form

Web/ Mail

Extern

Ge-misch tes

Specia

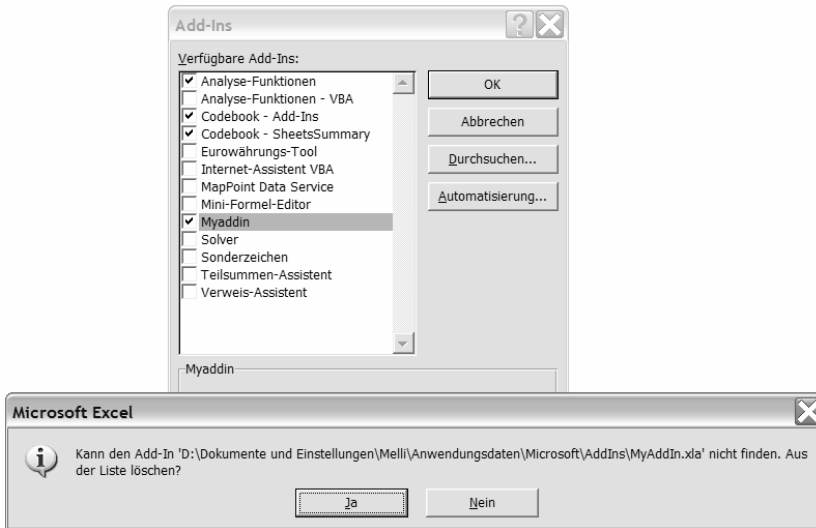


Abbildung 396: Entfernen eines Add-Ins aus der Liste der verfügbaren Add-Ins

```
' =====
' Auf CD      Buchdaten\Beispiele\13_Gemischtes\
' Dateiname   13_10_CodebookAddIn.xla
' Modul       mdl_05_DeleteAddIn
' =====
```

```
Sub DeleteAddIn()
    Dim strTitleName As String
    Dim strOldFile As String
    Dim strNewFile As String

    ' AddIn-Titel in Variable speichern
    strTitleName = "Myaddin"

    ' Pfad und Dateiname der Add-In-Datei
    strOldFile = AddIns(strTitleName).FullName

    ' Pfad und neuer Dateiname der Add-In-Datei
    strNewFile = AddIns(strTitleName).Path & "\del_" & _
        AddIns(strTitleName).Name

    ' Add-In deinstallieren (VBA-Projekt aus Speicher entfernen)
    AddIns(strTitleName).Installed = False

    ' Add-In im Explorer umbenennen
    Name strOldFile As strNewFile

    ' Excel gibt >>keine<< Fehlermeldung aus, wenn versucht wird,
    ' mit VBA ein unbekanntes Add-In zu installieren
    ' AddIns(strTitleName).Installed = True
```

```
' Add-Ins-Manager anzeigen
Application.Dialogs(xlDialogAddinManager).Show

' gewünschtes Add-In manuell aktivieren,
' damit Fehlermeldung erscheint
' und Add-In aus der Liste entfernt wird

' Add-In im Explorer zurück umbenennen
Name strNewFile As strOldFile

End Sub
```

472 Titel und Beschreibung einem Add-In hinzufügen

Den Titel und die Beschreibung (Kommentar) zu einem Add-In legen Sie im Eigenschaftsfenster der Excel-Datei über das Menü DATEI | EIGENSCHAFTEN fest. Um Änderungen an den Dokumenteneigenschaften eines Add-Ins zu tätigen, setzen Sie vorab im VBA-Editor die IsAddin-Eigenschaft auf False. Die Arbeitsmappe wird daraufhin in Excel angezeigt und Änderungen können vorgenommen werden. Diese werden erst übernommen, wenn Sie die IsAddin-Eigenschaft wieder auf True setzen und anschließend die Add-In-Datei speichern.

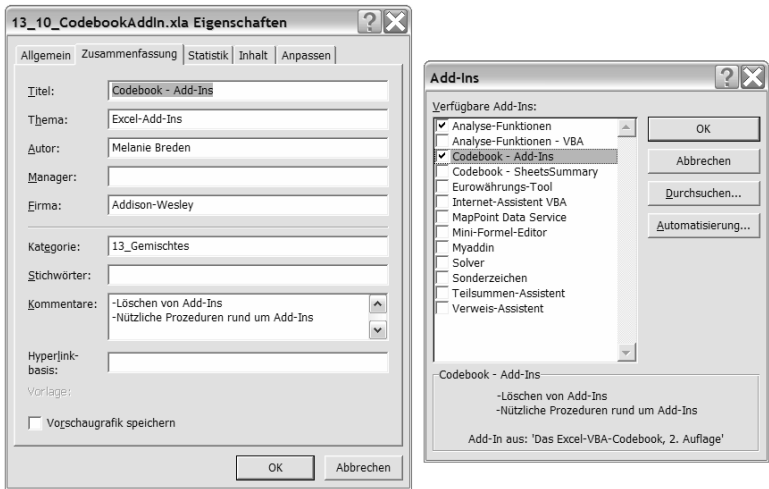


Abbildung 397: Titel und Beschreibung von Add-Ins festlegen

Wenn Sie die Dokumenteneigenschaften eines Add-Ins mit VBA ändern möchten, müssen Sie, wie über den manuellen Weg, vorab die IsAddin-Eigenschaft auf False stellen. Den Text für die Add-In-Beschreibung weisen Sie in der Prozedur AddInProperties der Variablen strComment zu. Normalerweise werden die einzelnen Zeilen im Fenster der Beschreibung zentriert dargestellt. Durch Hinzufügen von Leerzeichen nach einer Textzeile ist es aber möglich, eine linksbündige Ausrichtung zu erzielen. Nachdem die einzelnen Dokumenteneigenschaften festgelegt wurden, wird die IsAddin-Eigenschaft wieder auf True gesetzt und die Arbeitsmappe gespeichert. Anschließend können Sie im ADD-INS-Manager das Resultat sehen (siehe Abbildung 397).

Grundlagen

Allgemein

Datum
Zeit

ungen

Steuer
elemente

Befehl
leisten

Objekt

Diagramm

Ereignisse

User-
Form

Web/
Mail

Extern

Ge-
misch-
tes

Specia

```
'=====
' Auf CD      Buchdaten\Beispiele\13_Gemischtes\
' Dateiname   13_10_CodebookAddIn.xla
' Modul       mdl_06_AddInProperties
'=====

Sub AddInProperties()
    Dim objProp As Object
    Dim strComment As String

    ThisWorkbook.IsAddin = False

    Set objProp = ActiveWorkbook.BuiltinDocumentProperties

    strComment = _
        "-Löschen von Add-Ins" & vbCrLf & _
        "-Nützliche Prozeduren rund um Add-Ins" & vbCrLf & vbCrLf & _
        "Add-In aus: 'Das Excel-VBA-Codebook, 2. Auflage'"

    objProp("Title") = "Codebook - Add-Ins"
    objProp("Author") = "Melanie Breden"
    objProp("Subject") = "Excel-Add-Ins"
    objProp("Company") = "Addison-Wesley"
    objProp("Category") = "13_Gemischtes"
    objProp("Comments") = strComment

    ThisWorkbook.IsAddin = True
    ThisWorkbook.Save
End Sub
```

473 Code-Änderungen in Add-In speichern

Wenn Sie in normalen *.xls-Dateien Änderungen am VBA-Code vornehmen, erkennt Excel beim Beenden der Datei, ob diese Änderungen gespeichert wurden, und gibt eine entsprechende Meldung aus. Bei *.xla-Dateien, welche den IsAddin-Wert True aufweisen, erfolgt beim Beenden der Datei keine Abfrage nach einem Speichervorgang. Änderungen werden demnach nicht automatisch gespeichert. Wenn Sie in einem Add-In Codeänderungen vornehmen, müssen Sie Ihre Arbeit regelmäßig speichern, um Modifikationen in Ihrem Add-In zu übernehmen. Die automatische Speicherabfrage beim Beenden der Excel-Anwendung oder Deinstallieren eines Add-Ins kann man leicht über das Workbook_Before_Close-Ereignis der Add-In-Datei vornehmen. Fügen Sie den Prozedurnamen SaveAddIn dazu zwischen das Codegerüst des Ereignisses ein.

```
'=====
' Auf CD      Buchdaten\Beispiele\13_Gemischtes\
' Dateiname   13_10_CodebookAddIn.xla
' Modul       mdl_07_SaveAddIn
'=====

Sub SaveAddIn()
```

```
With ThisWorkbook
    If .Saved Then Exit Sub
    If MsgBox("Sollen Änderungen gespeichert werden?" _
        , vbYesNo, .Name) _
        = vbYes Then .Save
End With
End Sub
```

Grundlagen
Allgemein
Datum und Zeit

474 Funktionen in Add-In einbinden

Speichern Sie benutzerdefinierte Tabellen-Funktionen in einem Standardmodul einer Add-Ins-Datei. Diese Funktionen stehen bei installiertem Add-In in allen geöffneten Excel-Arbeitsmappen zur Verfügung. Der Dateiname des Add-In muss dem Funktionsnamen in der Zelle nicht vorangestellt werden.

Zur Umwandlung von arabischen in römische Zahlen steht in Excel die Funktion RÖMISCH() zur Verfügung. Umgekehrt gibt es dafür keine integrierte Funktion. In unserem Codebook Add-In haben wir exemplarisch für Funktionen die Funktion ARABISCH() integriert. Diese wandelt eine Zeichenkette aus römischen Ziffern in arabische Zahlen um. Sie können die Funktion ARABISCH() zum einen als Tabellen-Funktion in einer Zelle anwenden

```
=ARABISCH("MMIII")      ' liefert 2003
```

oder aber über eine Prozedur in VBA aufrufen.

```
'=====
' Auf CD      Buchdaten\Beispiele\13_Gemischtes\
' Dateiname   13_10_CodebookAddIn.xla
' Modul       mdl_08_FunctionARABISCH
'=====
```

```
Sub MyNumber()
    MsgBox ARABISCH("MMIII")
End Sub
```

```
'=====
' Auf CD      Buchdaten\Beispiele\13_Gemischtes\
' Dateiname   13_10_CodebookAddIn.xla
' Modul       mdl_08_FunctionARABISCH
'=====
```

```
Function ARABISCH(ByVal strRom As String) As Integer
    Dim intI As Integer
    Dim intTemp As Integer

    Do While Len(strRom) > 0
        Select Case Right(strRom, 1)
            Case "M"
                intTemp = 1000
            Case "D"
                intTemp = 500
```

Funktionen
Steuerelemente
Befehlsliste
Objekt
Diagramm
Ereignisse
User-Form
Web/Mail
Extern
Gemischtes
Spezial

```

Case "C"
    intTemp = 100
Case "L"
    intTemp = 50
Case "X"
    intTemp = 10
Case "V"
    intTemp = 5
Case "I"
    intTemp = 1
Case Else
End Select
strRom = Left(strRom, Len(strRom) - 1)

If intTemp >= intI Then
    intI = intTemp
Else
    intTemp = intTemp * (-1)
End If

ARABISCH = ARABISCH + intTemp
Loop
End Function

```

Die Funktion ARABISCH finden Sie bei installiertem Add-In im FUNKTIONSSASSISTENTEN in der Kategorie BENUTZERDEFINIERT.

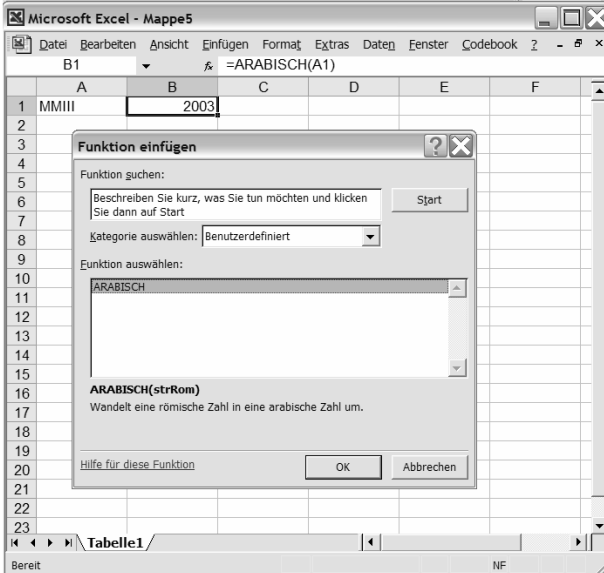


Abbildung 398: Tabellen-Funktionen über Add-In zur Verfügung stellen

Tipp

Den Hilfetext für die Funktion ARABISCH haben wir mit folgender Syntax im Direktfenster erstellt:

```
Application.MacroOptions "ARABISCH", _  
    "Wandelt eine römische Zahl in eine arabische Zahl um."  
Weitere Informationen zu benutzerdefinierten Tabellen-Funktionen finden Sie in der Kategorie »Einstieg«.
```

475 Makros aus Add-In aufrufen

In Add-Ins gespeicherte Tabellen-Funktionen lassen sich über den Funktionsassistenten leicht in Ihre Excel-Arbeitsmappe einfügen. Aber auch Ihre Prozeduren wollen aufgerufen und angewandt werden. Makros und Prozeduren aus Add-Ins werden nicht in der Liste der verfügbaren Makros über EXTRAS | MAKRO | MAKROS aufgeführt. Sie können aber dennoch darauf zugreifen, indem Sie in die oberste Zeile MAKRONAME den Namen der Prozedur eingeben und auf AUSFÜHREN klicken.

```
' =====  
' Auf CD      Buchdaten\Beispiele\13_Gemischtes\  
' Dateiname  13_10_CodebookAddIn.xla  
' Modul      mdl_09_TestMakro  
' =====  
  
Sub TestMakro()  
    MsgBox "'TestMakro' wurde aufgerufen"  
End Sub
```

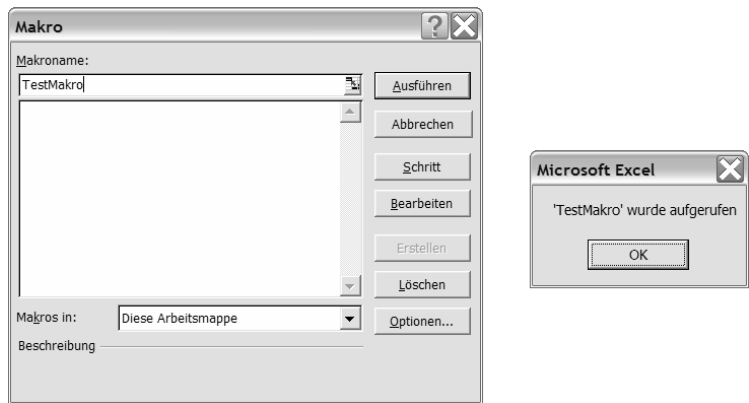


Abbildung 399: Add-In-Prozeduren manuell aufrufen

476 Makros aus Add-In über Schaltfläche aufrufen

Über eigene Menüs und Schaltflächen, welchen Sie Ihre Prozeduren aus dem Add-In zuweisen, können Sie bequem auf die in einem Add-In gespeicherten Prozeduren zugreifen. Die notwendige Prozedur AddCodebookMenu zur Menüerstellung wird beim Installieren des Add-Ins im Workbook_AddinInstall-Ereignis bzw. beim Öffnen der Datei im Workbook_Open-Ereignis aufgerufen. Wir verwenden beide Ereignis-Prozeduren, da beim Öffnen der Add-In-Datei über die aufgerufene Prozedur InstallAddIn zusätzlich überprüft wird, ob das Add-In bereits installiert ist, oder über die bereits vorgestellte Prozedur das Add-In automatisch installiert werden soll.

Grundlagen
Allgemein
Datum und Zeit
Formeln
Diagramme
Objekt
Diagramm
Ereignisse
User-Form
Web/Mail
Extern
Gemischtes
Spezial

```
'=====
' Auf CD      Buchdaten\Beispiele\13_Gemischtes\
' Dateiname   13_10_CodebookAddIn.xla
' Ereignis    DieseArbeitsmappe
'=====

Private Sub Workbook_AddinInstall()
    AddCodebookMenu
End Sub

Private Sub Workbook_Open()
    Dim cmdBarCtrl As CommandBarControl

    On Error Resume Next
    Set cmdBarCtrl = Application.CommandBars(1).Controls("Codebook")

    ' Wenn Menü vorhanden, beende Prozedur
    If Not cmdBarCtrl Is Nothing Then Exit Sub

    InstallAddIn
    AddCodebookMenu
End Sub
```

Beim Beenden der Excel-Anwendung werden die Add-In-Dateien automatisch geschlossen. Da das neue Menü temporär erstellt wurde, erübrigt sich ein explizites Löschen im Workbook_BeforeClose-Ereignis. Hier wird jedoch die Prozedur SaveAddIn zur Speicherabfrage bei Änderungen am VBA-Code aufgerufen.

```
'=====
' Auf CD      Buchdaten\Beispiele\13_Gemischtes\
' Dateiname   13_10_CodebookAddIn.xla
' Ereignis    DieseArbeitsmappe
'=====

Private Sub Workbook_BeforeClose(Cancel As Boolean)
    SaveAddIn
End Sub
```

Wird das Add-In während der Excel-Anwendung deinstalliert, muss das CODEBOOK-Menü entfernt werden, da die Prozeduren daraus dann nicht mehr zur Verfügung stehen. Der Prozeduraufruf DeleteCodebookMenu zum Entfernen des Menüs wird im Workbook_AddinUninstall-Ereignis hinterlegt.

```
'=====
' Auf CD      Buchdaten\Beispiele\13_Gemischtes\
' Dateiname   13_10_CodebookAddIn.xla
' Ereignis    DieseArbeitsmappe
'=====

Private Sub Workbook_AddinUninstall()
    DeleteCodebookMenu
End Sub
```

In unserem Beispiel-Add-In wird vor dem HILFE-Menü in der Menüleiste ein neues Menü CODEBOOK eingefügt. Diesem Menü wird die Schaltfläche ADD-INS zugewiesen, welche bei Klick das Formular frmAddIns zur Anzeige und Löschen von vorhandenen Add-Ins aufruft.

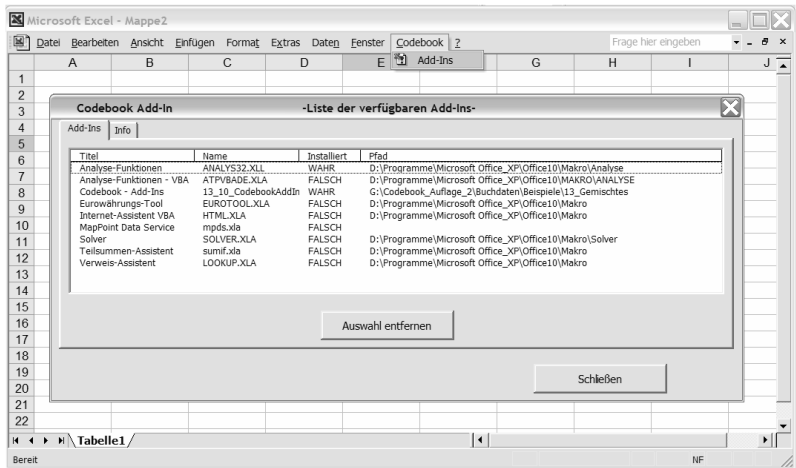


Abbildung 400: Aufgerufene UserForm des Codebook Add-Ins

```
' =====
' Auf CD      Buchdaten\Beispiele\13_Gemischtes\
' Dateiname  13_10_CodebookAddIn.xla
' Modul      mdl_10_AddCodebookMenu
' =====

Sub AddCodebookMenu()
    Dim CmdBar As CommandBar
    Dim CmdBarPop As CommandBarPopup

    Set CmdBar = Application.CommandBars(1)
    DeleteCodebookMenu
    Set CmdBarPop = CmdBar.Controls.Add( _
        Type:=msoControlPopup, _
        Before:=CmdBar.Controls.Count, _
        Temporary:=True)
    CmdBarPop.Caption = "&Codebook"

    With CmdBarPop.Controls.Add
        .Caption = "Add-Ins"
        .OnAction = "ShowFrmAddIn"
        .FaceId = 1952
    End With
End Sub

Sub DeleteCodebookMenu()
    On Error Resume Next
    Application.CommandBars(1).Controls("Codebook").Delete
End Sub
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

In unserem Beispiel-Add-In haben wir die Prozedur zum Löschen von Add-Ins in einem Formular integriert. Der Aufruf der UserForm erfolgt durch einen Klick auf die Schaltfläche ADD-INS im Menü CODEBOOK.

Da die Prozeduren der USERFORM auf die Liste der verfügbaren Add-Ins zugreifen, muss ein Tabellenblatt in der Excel-Anwendung sichtbar sein, da Excel sonst eine Fehlermeldung ausgeben würde. Über die bereits vorgestellte Funktion `IsWorkbookOpen` wird zunächst geprüft, ob eine Excel-Datei mit sichtbarem Tabellenblatt geöffnet ist, bevor das Formular `frmAddIns` angezeigt wird.

```
' =====
' Auf CD      Buchdaten\Beispiele\13_Gemischtes\
' Dateiname   13_10_CodebookAddIn.xla
' Modul       mdl_10_AddCodebookMenu
' =====

Sub ShowFrmAddIn()
    ' In Excel 97 ohne vbModeless
    If IsWorkbookOpen Then frmAddIns.Show vbModeless
End Sub
```

Tipp

Ausführliche Informationen zu Formularen finden Sie in der Kategorie »UserForm«.

Die nachfolgenden Prozeduren sind im Klassenmodul der USERFORM hinterlegt.

```
' =====
' Auf CD      Buchdaten\Beispiele\13_Gemischtes\
' Dateiname   13_10_CodebookAddIn.xla
' Formular     frmAddIns
' =====

Private mWks As Worksheet

Private Sub UserForm_Initialize()
    Dim intI As Integer

    Application.ScreenUpdating = False
    Set mWks = Worksheets.Add

    ' Quelle für ListBox
    Me.LbAddIns.RowSource = mWks.Name & "!A2:D" & AddIns.Count + 1

    ' Add-In-Liste auf neuem Tabellenblatt erstellen
    With mWks
        .Range("A1:D1").Value = _
            Array("Titel", "Name", "Installiert", "Pfad")

        For intI = 1 To AddIns.Count
            .Cells(intI + 1, 1) = AddIns(intI).Title
            .Cells(intI + 1, 2) = AddIns(intI).Name
```

```
        .Cells(intI + 1, 3) = AddIns(intI).Installed
        .Cells(intI + 1, 4) = AddIns(intI).Path
    Next

    ' Temporäres Tabellenblatt ausblenden
    .Visible = xlSheetVeryHidden
End With

Application.ScreenUpdating = True
End Sub

Private Sub CmbDelete_Click()
    Dim strTitleName As String
    Dim strOldFile As String
    Dim strNewFile As String

    On Error GoTo ERRORHANDLER
    ' Add-In-Titel auslesen
    strTitleName = LbAddIns.List(LbAddIns.ListIndex, 0)

    ' Form entladen
    Unload Me

    If ThisWorkbook.Name = AddIns(strTitleName).Name Then
        MsgBox "Das Add-In wird deinstalliert, " & _
            "aber nicht aus der Liste entfernt!"
    End If

    ' Add-In deinstallieren (VBA-Projekt aus Speicher entfernen)
    AddIns(strTitleName).Installed = False

    ' Pfad und Dateiname der Add-In-Datei
    strOldFile = AddIns(strTitleName).FullName

    ' Pfad und neuer Dateiname der Add-In-Datei
    strNewFile = AddIns(strTitleName).Path & "\del_" & _
        AddIns(strTitleName).Name

    ' Add-In im Explorer umbenennen
    Name strOldFile As strNewFile

    Application.Dialogs(xlDialogAddinManager).Show

    ' Add-In im Explorer zurück umbenennen
    Name strNewFile As strOldFile
    Exit Sub
ERRORHANDLER:
    MsgBox "Wählen Sie einen Eintrag aus!", vbCritical
End Sub

Private Sub CmbClose_Click()
    Unload Me
End Sub
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

```
Private Sub UserForm_Terminate()  
    If Not mWks Is Nothing Then  
        With Application  
            .ScreenUpdating = False  
            .DisplayAlerts = False  
        End With  
  
        ' Temporäres Tabellenblatt löschen  
        With mWks  
            .Visible = xlSheetVisible  
            .Delete  
        End With  
  
        With Application  
            .ScreenUpdating = True  
            .DisplayAlerts = True  
        End With  
  
        Set mWks = Nothing  
    End If  
End Sub
```

In dieser Kategorie werden wir Ihnen einige etwas komplexere Codebeispiele vorstellen. Stellenweise werden wir auf eine ausführliche Beschreibung verzichten. Dies jedoch nur, wenn die Themen bereits in den vorangegangenen Kategorien beschrieben wurden.

477 Ein kleines Puzzle

In diesem Beispiel zeigen wir Ihnen, wie Sie ein Puzzle in Excel programmieren können. Es werden neun Grafiken verwendet, die in der richtigen Reihenfolge zusammen ein Bild ergeben. Wir haben dabei eine einzige Grafik in neun gleich große Stücke unterteilt. Um eine Grafik in Teile zu zerlegen, verwenden Sie am besten ein geeignetes Grafikprogramm. Für unser Beispiel haben wir in Excel die einzelnen Stücke zugeschnitten und diese ins Grafikprogramm PaintShop Pro 7 übertragen. PaintShop Pro haben wir dazu verwendet, um die neun Einzelteile in je eine *.jpg-Datei zu speichern. Dadurch sind insgesamt zehn *.jpg-Dateien entstanden. Die ganze Grafik sowie die neun Einzelteile sind auf der CD im selben Verzeichnis wie die Beispieldatei zu finden. Jedes dieser Puzzle-Teile soll in einer eigenen Zelle erscheinen. Um das Puzzle zu mischen, haben wir einen Zufallsgenerator programmiert.

Der Zufallsgenerator wird später ein Bestandteil des gesamten Puzzlecodes sein. Wir erläutern hier an dieser Stelle jedoch zuerst den Zufallsgenerator.

Wichtig an unserem Zufallsgenerator ist, dass neun *verschiedene* Zahlen im Bereich von 1 bis 9 ermittelt werden müssen. Die Zahlen dürfen untereinander nicht identisch sein. Jede der neun Zahlen muss eine andere sein, denn sie werden später den Einzelbildern zugewiesen. Wenn eine Zahl doppelt vorhanden wäre, würde das bedeuten, dass das Puzzle doppelte Einzelbilder haben würde und andere Bilder wiederum fehlen würden.

Die äußere Schleife `For...Next` ist dazu da, neun Zahlen zu generieren. Die innere `Do...Loop`-Schleife ermittelt die Zufallszahlen. Die Zahlen werden an eine Variable mit dem Namen `strDummy` übergeben. In der inneren Schleife wird sichergestellt, dass die Zahlen nicht doppelt vorkommen. Wir verwenden dazu die Funktion `InStr`, welche die bereits vorhandenen Einträge im Dummy prüft. Wenn `InStr` den Wert 0 aufweist, dann bedeutet das, dass die Zahl, welche der Zufallsgenerator ermittelt hat, in der Variablen `strDummy` noch nicht vorhanden ist. Die Zahl wird dann an die Variable `strDummy` übergeben. Die `Do...Loop`-Schleife wird jeweils so lange durchlaufen, bis eine Zufallszahl gefunden wird, die in der Variablen `strDummy` noch nicht vorhanden ist.

Hinweis

Die Variable `strDummy` für den Dummy wurde vom Datentyp `String` deklariert, da die Zeichenkette theoretisch aus mehr als neun Zahlen bestehen könnte. In diesem Falle müsste ein Trennzeichen verwendet werden, damit beispielsweise die Zahl 10 nicht als 1 und 0 interpretiert würde.

Um die neun Zufallszahlen auszugeben, verwenden wir ein Meldungsfenster `MsgBox`.

Grundlagen
Allgemein
Datum und Zeit
Diagramme
Formeln
Objekte
Diagramm
Ereignisse
User-Form
Web/Mail
Extern
Gemischtes
Specials

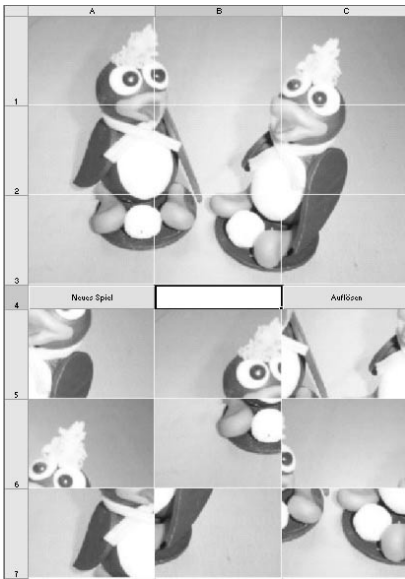


Abbildung 401: Ein Puzzlespiel

```
'=====
' Auf CD      Buchdaten\Beispiele\14_Specials\
' Dateiname   14_01_Puzzle.xls
' Tabelle     14_Specials_Puzzle
' Modul       mdl_01_Random
'=====
```

```
Sub Random()
    Dim strDummy As String
    Dim intZahl As Integer
    Dim i As Integer

    For i = 1 To 9
        Do
            intZahl = Int(Rnd() * 9) + 1
            Loop Until InStr(strDummy, intZahl) = 0

            ' Die Zufallszahlen werden in den Dummy gefüllt
            ' Der Dummy umfasst am Ende 9 Zahlen
            strDummy = strDummy & intZahl
        Loop
    Next i

    MsgBox strDummy
End Sub
```

Die nächste Prozedur, die wir uns ansehen, ist dazu da, die neun Bildteile gemischt in neun Zellen auszugeben. Nachdem Sie nun wissen, wie der Zufallsgenerator funktioniert, integrieren wir ihn in diese Prozedur.

Nach der Variablendeklaration werden alle im Tabellenblatt enthaltenen Bilder gelöscht, damit jedes Mal beim Start der Prozedur ein neues Puzzle aufgebaut wird. Der Variablen `i` wird der Wert 1 zugewiesen. Damit werden später nacheinander die Zahlen aus dem Array `strDummy`, die im Zufallsgenerator aufbereitet wurden, ausgelesen. Die Bildschirmaktualisierung `ScreenUpdating` wird deaktiviert. Damit wird einerseits das Bildflackern unterdrückt und andererseits die Geschwindigkeit der Prozedur erhöht. Am Ende des Codes wird die Bildschirmaktualisierung wieder aktiviert. In der ersten `For`-Schleife befindet sich der Zufallsgenerator. Hier werden die neun Zufallszahlen für die neun Bilder an die Variable `strDummy` übergeben. Nach Ablauf dieser Schleife befinden sich in der Variablen `strDummy` neun Zahlen, für jedes Bild, das eingefügt werden soll, eine.

In der zweiten `For`-Schleife werden die Bilder ab der Zeile 5, Spalte 1 in die Tabelle eingefügt. In jeder Zeile erscheinen drei der Bildteile, dies verteilt über drei Zeilen. Damit werden über neun Zellen Bilder gelegt.

Innerhalb der inneren `For`-Schleife werden die einzelnen Bilder über den neun Zellen eingefügt. Wichtig ist, dass beim Einfügen der Bilder mittels `Pictures.Insert` die Zahlen des Zufallsgenerators `strDummy` ausgelesen werden. Wir verwenden die Funktion `InStr` in Kombination mit `strDummy` und der Variablen `i`. Der Variablen `strDummy` wurde zu Beginn der Prozedur der Wert 1 zugewiesen. Mit der Variablen `i` werden nacheinander die Zahlen aus dem `strDummy`-Array ausgelesen. Vor dem Ablauf der inneren Schleife wird der Wert der Variablen `i` jeweils um den Wert 1 erhöht. In der Funktion `InStr` wird so der Dateiname der Bilder ermittelt, die in die Zellen eingefügt werden. Die neun Bilder sind entsprechend gemischt: z.B. `Puzzle8.jpg`, `Puzzle3.jpg`, `Puzzle5.jpg` usw.

Die genaue Beschreibung, wie die einzelnen Bilder exakt über den Zellen eingepasst werden können, können Sie in der Kategorie »Objekte« nachlesen. Wir werden an dieser Stelle nicht mehr im Detail darauf eingehen. In unserer Prozedur geschieht der Vorgang in der `With`-Anweisung.

Damit nur die Zellen, die die Spielfelder darstellen, sichtbar sind, wurden einige Zeilen und Spalten ausgeblendet. Sichtbar sind nur noch die Spalten A bis C und die Zeilen 1 bis 7. Um das Puzzle zu spielen, ziehen Sie die Bildteile, die sich nun im unteren Teil der Mappe befinden, in der richtigen Reihenfolge über die Zellen im oberen Bereich der Tabelle.

```
'=====
' Auf CD      Buchdaten\Beispiele\14_Specials\
' Dateiname   14_01_Puzzle.xls
' Tabelle     14_Specials_Puzzle
' Modul       mdl_02_MixPuzzle
'=====

Sub MixPuzzle()
    Dim pic As Picture
    Dim strPath As String
    Dim i As Integer
    Dim intRow As Integer
    Dim intCol As Integer

    ' Variablen für den Zufallsgenerator
    Dim strDummy As String
    Dim intZahl As Integer
    Dim inti As Integer

    strPath = ThisWorkbook.Path & "\Puzzle"
    ActiveSheet.Pictures.Delete
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

```

i = 1

Application.ScreenUpdating = False

' Die Zufallszahlen werden in den Dummy gefüllt
' Der Dummy umfasst am Ende 9 Zahlen
For inti = 1 To 9
    Do
        intZahl = Int(Rnd() * 9) + 1
        Loop Until InStr(strDummy, intZahl) = 0

        strDummy = strDummy & intZahl
    Next inti

' Bilder an Zellen übergeben
For intRow = 5 To 7
    For intCol = 1 To 3
        Set pic = ActiveSheet.Pictures.Insert _
            (strPath & InStr(strDummy, i) & ".jpg")

        ' Bild exakt in Zellen ausrichten
        With ActiveSheet.Cells(intRow, intCol)
            pic.Left = .Left
            pic.Top = .Top
            pic.Height = .Height
            pic.Width = .Width
        End With

        i = i + 1
    Next intCol
Next intRow

Application.ScreenUpdating = True

Set pic = Nothing
End Sub

```

Die dritte und letzte Prozedur in unserem Beispiel ist dazu da, das Puzzle aufzulösen. Dabei werden zuerst alle im Tabellenblatt vorhandenen Bilder gelöscht, um die Bildteile neu einzufügen. Die einzelnen Bilder werden der Reihe nach über den Zellen im Bereich A1:C3 eingepasst. Diesmal jedoch in der korrekten Reihenfolge, so dass das ursprüngliche Gesamtbild entsteht.

```

' =====
' Auf CD      Buchdaten\Beispiele\14_Specials\
' Dateiname   14_01_Puzzle.xls
' Tabelle     14_Specials_Puzzle
' Modul       mdl_03_SolvePuzzle
' =====

Sub SolvePuzzle()
    Dim pic As Picture

```

```
Dim strPath As String
Dim i As Integer
Dim intRow As Integer
Dim intCol As Integer

strPath = ThisWorkbook.Path & "\Puzzle"
ActiveSheet.Pictures.Delete

i = 1

Application.ScreenUpdating = False

For intRow = 1 To 3
  For intCol = 1 To 3
    Set pic = ActiveSheet.Pictures.Insert(strPath & i & ".jpg")

    With ActiveSheet.Cells(intRow, intCol)
      pic.Left = .Left
      pic.Top = .Top
      pic.Height = .Height
      pic.Width = .Width
    End With
    i = i + 1
  Next intCol
Next intRow

Application.ScreenUpdating = True

Set pic = Nothing
End Sub
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

478 Die Farbpalette von Excel verändern

Die Farbpalette von Excel stellt lediglich 56 verschiedene Farben zur Verfügung. Mehr können nicht in die Farbpalette aufgenommen werden. Welche Farben das sind, können Sie jedoch selbst bestimmen. Änderungen der Farbpalette wirken sich lediglich auf die aktive Mappe aus.

Um Farben der Farbpalette manuell zu verändern, klicken Sie auf den Menüpunkt EXTRAS | OPTIONEN und aktivieren die Registerkarte FARBE. Hier können Sie jede einzelne bestehende Farbe durch eine andere ersetzen. Aktivieren Sie die Grundfarbe, die Sie verändern möchten, und klicken Sie dann auf die Schaltfläche ÄNDERN. Eine Dialogbox mit zwei verschiedenen Registerkarten wird angezeigt. Wählen Sie dort die gewünschte Farbe aus. In der Kategorie »Einstieg« können Sie nachlesen, was RGB-Farben sind und wie sich diese zusammensetzen. Es stehen insgesamt 16,77 Mio. verschiedene RGB-Farbwerte zur Verfügung.

Wenn Sie die Ursprungsfarben von Excel wiederherstellen möchten, steht dazu in den OPTIONEN im Register FARBE eine Schaltfläche mit dem Namen ZURÜCKSETZEN zur Verfügung. Es werden dabei *sämtliche* Farben auf die Standardeinstellung zurückgestellt.

Wenn Sie beispielsweise einen Farbverlauf von Schwarz bis Gelb erstellen möchten, dann ist es relativ aufwendig, diesen manuell zusammenzustellen. Wir werden deshalb in unserem Beispiel zeigen, wie Sie automatisch einen Farbverlauf per VBA erzeugen können. 56 Farben werden dabei der Farbpalette von Excel neu zugewiesen.

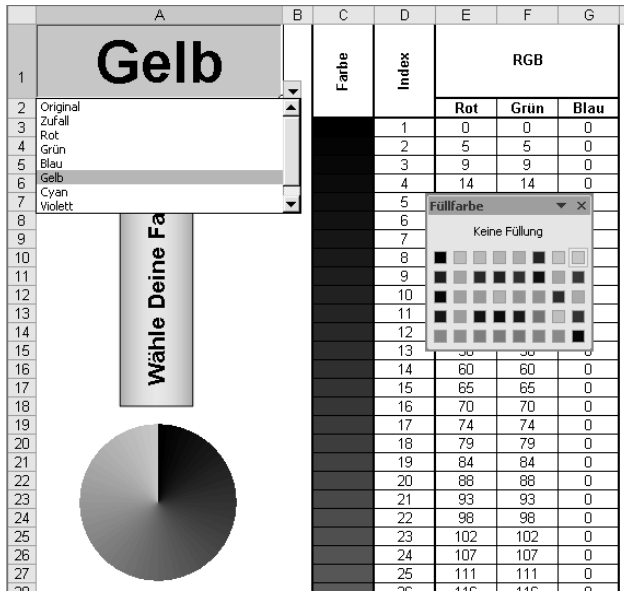


Abbildung 402: Die Farbpalette von Excel verändern

In unserer Beispieldatei haben wir in der Zelle A1 eine Gültigkeit erstellt (DATEN | GÜLTIGKEIT | LISTE). Hier können über ein DropDown-Feld verschiedene Farbpaletten ausgewählt werden. In der Mappe sind zudem Beispieldiagramme vorhanden. Diese zeigen, wie sich eine Veränderung der Farbpalette auf bestehende Objekte auswirkt. Zusätzlich zu den Farbverläufen finden Sie die Auswahl ZUFALL. Wenn Sie diese Farbpalette auswählen, werden nach dem Zufallsprinzip 56 unterschiedliche Farben zusammengestellt. Jedes Mal, wenn diese Prozedur ausgeführt wird, erhalten Sie eine andere Farbzusammenstellung.

Damit auf Veränderung des Inhalts der Zelle A1 die Farben zugewiesen werden, benötigen wir das Ereignis `Worksheet_Change` des Tabellenblattes. Das Ereignis veranlasst, dass je nach Auswahl im DropDown-Feld eine der Prozeduren gestartet wird, die sich im Standardmodul befinden. In dem Modul sind drei verschiedene Prozeduren vorhanden.

- **Original:** Sie setzt die Farben in den Originalzustand zurück.
- **Random:** Die Farben werden nach einem Zufallsgenerator festgelegt.
- **MyColor:** Je nachdem, welche Farbe ausgewählt wurde, wird ein Farbverlauf erstellt.

Die Prozedur `MyColor` erfordert die Rückgabe der RGB-Werte, je nachdem, welche Farbe ausgewählt wurde. Die Variable `intC` oder der Wert 0 werden abhängig von der gewählten Farbe an die Ereignis-Prozedur `Worksheet_Change` übergeben. Es müssen, um die RGB-Farben zu erhalten, immer drei Werte übergeben werden (Rot, Grün und Blau). In der `For`-Schleife der Prozedur `MyColor` werden der Variablen `intC` die RGB-Werte des gewählten Farbverlaufes übergeben.

In der `Select Case`-Anweisung der Ereignis-Prozedur wird geprüft, welche Auswahl im DropDown-Feld des Tabellenblattes getroffen wurde. Je nachdem wird eine der drei Prozeduren aus dem Standardmodul aufgerufen. Falls einer der Farbverläufe gewählt wurde, werden die dazugehörigen RGB-Farbwerte übergeben, zum Beispiel `intC, 0, 0` für den roten Farbverlauf.

In der For...Next-Schleife, die nach der Select Case-Anweisung folgt, werden zudem der Farbin-
dex sowie die RGB-Werte in die dafür vorgesehenen Spalten des Tabellenblattes geschrieben. Inner-
halb der With-Anweisung wird veranlasst, dass die Farbpalette entkoppelt und eingeblendet wird.

```
' =====  
' Auf CD      Buchdaten\Beispiele\14_Specials\  
' Dateiname  14_02_ColorPalette.xls  
' Ereignis    Tabelle1(14_Specials_Color)  
' =====  
  
Private Sub Worksheet_Change(ByVal Target As Range)  
    Dim i As Integer  
  
    With Application  
        .ScreenUpdating = False  
        .EnableEvents = False  
    End With  
  
    If Target.Address <> "$A$1" Then  
        Exit Sub  
    End If  
  
    Select Case (Target.Value)  
        Case "Original"                ' Original  
            Original  
        Case "Zufall"                  ' Random  
            Random  
  
            ' Prozedur MyColor aufrufen und Variablen übergeben  
        Case "Rot"                     ' Red  
            MyColor intC, 0, 0  
        Case "Grün"                    ' Green  
            MyColor 0, intC, 0  
        Case "Blau"                    ' Blue  
            MyColor 0, 0, intC  
        Case "Gelb"                     ' Yellow  
            MyColor intC, intC, 0  
        Case "Cyan"                     ' Cyan  
            MyColor 0, intC, intC  
        Case "Violett"                  ' Violet  
            MyColor intC, 0, intC  
        Case "Grau"                     ' Grey  
            MyColor intC, intC, intC  
    End Select  
  
    For i = 2 To 57  
        Cells(i + 1, 3).Interior.Color = ActiveWorkbook.Colors(i - 1)  
        Cells(i + 1, 4) = i - 1  
        Cells(i + 1, 5) = ActiveWorkbook.Colors(i - 1) \ 256 ^ 0 _  
            And 255  
        Cells(i + 1, 6) = ActiveWorkbook.Colors(i - 1) \ 256 ^ 1 _  
            And 255  
        Cells(i + 1, 7) = ActiveWorkbook.Colors(i - 1) \ 256 ^ 2 _
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

```

        And 255

    Next i

    With Application
        ' Farbpalette entkoppeln und anzeigen
        .CommandBars("Fill Color").Visible = True
        .ScreenUpdating = True
        .EnableEvents = True
    End With
End Sub

```

Wie bereits erwähnt, befinden sich in dem separaten Modul drei Prozeduren, die je nach Auswahl des DropDown-Feldes der Zelle A1 ausgeführt werden. Die erste Prozedur *Original* setzt die Farben in den Originalzustand von Excel mit *Reset.Colors* zurück.

In der zweiten Prozedur *Random* werden nach dem Zufallsprinzip Rnd 56 von 1677216 möglichen Farben ermittelt.

Die Prozedur *MyColor* wird aus dem *Select Case*-Abschnitt der *Worksheet_Change*-Ereignis-Prozedur aufgerufen. Beim Aufruf werden die benötigten Argumente der einzelnen RGB-Werte für die Farbverläufe an *MyColor* übergeben.

```

' =====
' Auf CD      Buchdaten\Beispiele\14_Specials\
' Dateiname   14_02_ColorPalette.xls
' Modul       mdl_01_Colors
' =====

```

Option Explicit

```

Public intC As Integer
Private i As Integer

```

```

Sub Original()
    ActiveWorkbook.ResetColors
End Sub

```

```

Sub Random()
    Dim i As Long
    For i = 1 To 56
        ' Ergibt: 16777216
        ActiveWorkbook.Colors(i) = Rnd() * 256 ^ 3
    Next i
End Sub

```

```

Sub MyColor(intR As Integer, intG As Integer, intB As Integer)
    For i = 1 To 56
        intC = Round((i - 1) * (260 / 56), 0)
        ActiveWorkbook.Colors(i) = RGB(intR, intG, intB)
    Next i
End Sub

```

479 Einen Lottoschein ausfüllen (Array)

Bevor wir mit dem Aufbau des Lottoscheins beginnen, werden wir uns zuerst den Kern der Programmierung, den Zufallsgenerator, ansehen .

In einem Lottoschein können von 45 möglichen Zahlen sechs angekreuzt werden. Das bedeutet, dass wir mittels eines Zufallsgenerators aus den möglichen Zahlen 1 bis 45 sechs Werte auslesen müssen. Jeder der sechs Werte muss eindeutig sein. Um dies zu realisieren, werden wir mit zwei Arrays arbeiten.

Das erste Array `Arr1` enthält 45 Datenfelder. Das zweite Array `Arr2` enthält 6 Datenfelder. Beide Arrays werden zu Beginn der Prozedur mit der `Dim`-Anweisung entsprechend dimensioniert.

```
Dim Arr1(1 To 45) As Byte
Dim Arr2(1 To 6) As Byte
```

Nach der Variablendeklaration wird das erste Datenfeld `Arr1` mit 45 Werten gefüllt. Dazu verwenden wir eine `For...Next`-Schleife. Der Inhalt des Datenfeldes sieht gekürzt wie folgt aus:

```
{1,2,3,...,44,45}
```

In der zweiten `For...Next`-Schleife werden die sechs Zufallszahlen ermittelt und an das zweite Array `Arr2` übergeben. Zu Beginn der Schleife übergeben wir der Variablen `x` einen Zufallswert im Bereich von 1 bis 45. Wir verwenden dazu die Funktion `Rnd` (`Randomize` = Zufallszahl). Da die Funktion `Rnd` einen Wert im Bereich von 1 bis 45 *mit Nachkommastellen* liefert, übergeben wir die Berechnung der Zufallszahl an die Funktion `Int`, diese wiederum gibt eine *Ganzzahl* zurück.

Vor dem Ende der zweiten `For`-Schleife wird dem ersten Array `Arr1` eine 0 übergeben, wenn die entsprechende Zahl in das zweite Array `Arr2` geschrieben wurde. So werden alle Zahlen, die im ersten Array bereits einmal ausgelesen wurden, auf 0 gestellt. Damit ist sichergestellt, dass jede Zahl nur einmal ausgelesen werden kann. Der Inhalt der beiden Arrays könnte nach dem Ausführen des Codes wie folgt aussehen:

```
Arr1: {1,2,3,4,0,6,0,8,0,0,...,35,0,37,38,39,40,41,42,43,44,0}
Arr2: {36, 5, 9, 45, 7, 10}
```

Beachten Sie die Zahlen 0 im ersten Array. Diese entsprechen den Stellen, die dem zweiten Array übergeben wurden.

In der zweiten `For`-Schleife wurde geprüft, ob im ersten Array `Arr1` an der Stelle, die der Zufallszahl entspricht, eine 0 steht. Falls dies zutrifft, würde das bedeuten, dass diese Zahl bereits ausgelesen und an `Arr2` übergeben wurde. In diesem Fall wird zurück auf die Sprungmarke `Playback` verwiesen und eine neue Zufallszahl berechnet. Dieser Vorgang wird so lange durchgeführt, bis sechs ungleiche Zahlen ermittelt wurden.

Am Ende der Prozedur wird der Inhalt der beiden Arrays im Direktfenster Ihres VBA-Editors mit `Debug.Print` ausgegeben.

```
'=====
' Auf CD      Buchdaten\Beispiele\14_Specials\
```

Grundlagen

Allgemein

Datum und Zeit

Tabellen

Steuerungselemente

Befehlsliste

Objekte

Diagramme

Ereignisse

User-Formen

Web/Mail

Extern

Gemischtes

Spezial

```
' Dateiname 14_03_Lotto.xls
' Modul      mdl_01_MyArray
' =====
```

Option Explicit

```
Sub MyArray()
    Dim Arr1(1 To 45) As Byte
    Dim Arr2(1 To 6) As Byte
    Dim i As Byte
    Dim x As Byte

    ' Das Array wird mit 45 Zahlen gefüllt
    For i = 1 To 45
        Arr1(i) = i
    Next

    ' Sechs Zufallszahlen werden aus dem Array ausgelesen
    For i = 1 To 6
Playback:
        x = Int(Rnd * 45) + 1
        If Arr1(x) = 0 Then GoTo Playback
        Arr2(i) = x
        Arr1(x) = 0
    Next

    ' Den Inhalt der Arrays an das Direktfenster übergeben
    For i = 1 To 45: Debug.Print Arr1(i),: Next
    ' Eine leere Zeile im Direktfenster erzeugen
    Debug.Print
    ' Den Inhalt des Arrays Arr2 an das Direktfenster übergeben
    For i = 1 To 6: Debug.Print Arr2(i),: Next
    ' Wiederum eine leere Zeile im Direktfenster erzeugen
    Debug.Print
End Sub
```

Da Sie nun wissen, wie der Zufallsgenerator funktioniert, werden wir diesen in unser Lotto-Programm einbauen.

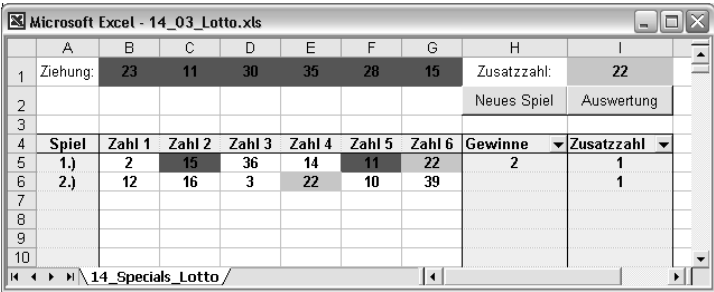


Abbildung 403: Lottospiel

In der ersten Prozedur von zwei Prozeduren wird der Zufallsgenerator verwendet. Dort werden die Glückszahlen ermittelt. Die Zufallszahlen werden im Bereich B5:GN ausgegeben. Zu Beginn der Prozedur können Sie über einen Eingabedialog `InputDialog` wählen, wie viele Zeilen an Zufallszahlen Sie ausgeben möchten. Jede Zeile entspricht einem Spielfeld.

```
'=====
' Auf CD      Buchdaten\Beispiele\14_Specials\
' Dateiname  14_03_Lotto.xls
' Modul      mdl_02_PlayAGame
'=====

Sub PlayAGame()
    Dim Arr1(1 To 45) As Byte
    Dim Arr2(1 To 6) As Byte
    Dim i As Byte
    Dim x As Byte
    Dim lngGames As Long
    Dim lngRow As Long

    Application.ScreenUpdating = False

    ' Felder leeren
    Range("A5:I" & Cells(Rows.Count, 1).End(xlUp).Row).Delete

    ' Anzahl Spielfelder
    lngGames = Application.InputBox _
        ("Geben Sie die Anzahl der Spielfelder ein", _
        Title:="Wieviele Spiele?", _
        Default:=2)

    ' Zur Anzahl Spielfelder eine 4 addieren, damit erst ab
    ' 5. Zeile die Spielzahlen eingetragen werden
    lngGames = lngGames + 4

    ' Ab Zeile 5 die Zufallszahlen eintragen
    For lngRow = 5 To lngGames
        ' Erstes Array füllen mit 45 möglichen Werten
        For i = 1 To 45
            Arr1(i) = i
        Next

        ' Zweites Array mit 6 Zufallszahlen füllen
        ' Benutzte Zahlen aus dem ersten Array löschen
        For i = 1 To 6
            Playback:
                x = Int(Rnd * 45) + 1
                If Arr1(x) = 0 Then GoTo Playback
                Arr2(i) = x
                Arr1(x) = 0
        Next

        ' Nummerierung in der Spalte A
        Cells(lngRow, 1) = lngRow - 4 & "."
    Next
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

```

' Zufallszahlen aus dem Array an das Spielfeld übergeben
Cells(lngRow, 2) = Arr2(1)
Cells(lngRow, 3) = Arr2(2)
Cells(lngRow, 4) = Arr2(3)
Cells(lngRow, 5) = Arr2(4)
Cells(lngRow, 6) = Arr2(5)
Cells(lngRow, 7) = Arr2(6)
Next lngRow

```

```

Application.ScreenUpdating = True
End Sub

```

In der zweiten Prozedur wird ermittelt, welche Spielzahlen mit der Ziehung übereinstimmen. Die Zahlen der offiziellen Ziehung können Sie manuell in die Zeile 1 eingeben. Wenn Sie einen Treffer haben, der den Zahlen im Bereich B1:G1 entspricht, werden die entsprechenden Zufallszahlen im Bereich B5:GN rot eingefärbt. Wenn eine der Zahlen der Zusatzzahl entspricht, wird diese gelb eingefärbt. In Spalte H wird die Anzahl der roten Treffer angezeigt. In Spalte I wird eine 1 eingetragen, wenn eine der Zufallszahlen der Zusatzzahl entspricht.

```

' =====
' Auf CD      Buchdaten\Beispiele\14_Specials\
' Dateiname   14_03_Puzzle.xls
' Modul       mdl_03_DidYouWin
' =====

Sub DidYouWin()
    Dim intGames As Integer
    Dim c As Range
    Dim intRow As Integer
    Dim intRed As Integer
    Dim intYellow As Integer
    Dim strWin As String

    ' Ermitteln, wieviele Spiele gespielt wurden
    ' Letzte intRow in Spalte A an Variable übergeben
    intGames = Cells(Rows.Count, 1).End(xlUp).Row

    ' Farben und Zellinhalt von letzter Auswertung leeren
    Range("B5:G" & intGames).Interior.ColorIndex = xlNone
    Range("H5:I" & intGames).Value = ""

    ' Gewinnzahlen ermitteln
    For Each c In Range("B5:G" & intGames)
        ' Rot, Index 3 = Treffer
        If c = Range("B1") Or c = Range("C1") Or _
           c = Range("D1") Or c = Range("E1") Or _
           c = Range("F1") Or c = Range("G1") Then
            c.Interior.ColorIndex = 3
        End If

        ' Gelb, Index 6 = Zusatzzahl
    
```

<pre> If c = Range("H1") Then c.Interior.ColorIndex = 6 End If Next c For intRow = 5 To intGames ' Prüfen, ob pro Zeile bzw. Bereich Rot oder Gelb ' gefunden wird ' intRed: Die gefundenen intReden Zellen zählen ' intYellow: Es kann nur eine Zelle intYellow sein For Each c In Range(Cells(intRow, 2), _ Cells(intRow, 2).Offset(0, 5)) If c.Interior.ColorIndex = 3 Then intRed = intRed + 1 If c.Interior.ColorIndex = 6 Then intYellow = 1 Next c ' Auswertung in grünen Bereich schreiben If intRed > 0 Then Cells(intRow, 8).Value = intRed If intYellow > 0 Then Cells(intRow, 9).Value = intYellow intRed = 0 intYellow = 0 Next intRow ' Gewinn ermitteln For Each c In Range("H5:H" & intGames) If c.Value > 2 Then strWin = "Ja" Next c ' Meldung ausgeben If strWin = "Ja" Then MsgBox "Du hat etwas gewonnen :-)))", _ vbInformation, "Gewonnen!" Else MsgBox "Diesmal hat es leider nicht geklappt", _ vbCritical, _ "Kein Treffer" End If End Sub</pre>	

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

480 Formeln, Werte und Text eines Tabellenblattes ermitteln

In umfangreichen Tabellenblättern kann der Anwender leicht den Überblick über die vorhandenen Formeln verlieren. Unser Rezept ist so aufgebaut, dass Formeln, Werte und Text des aktiven Tabellenblattes in einem separaten Tabellenblatt ausgegeben werden.

Die gewünschten Informationen sollen auf einem separaten Tabellenblatt mit dem Namen 14_SPECIALS_FORMULAS ausgegeben werden (siehe Abbildung 404). Im ersten Codeteil unserer Prozedur wird geprüft, ob bereits ein Tabellenblatt mit dem Namen 14_SPECIALS_FORMULAS vorhanden ist. Wenn nicht, wird dieses erstellt (Worksheets.Add). Falls das Tabellenblatt bereits vorhanden ist, werden die darin enthaltenen Werte gelöscht (Delete), damit das Tabellenblatt neu aufbereitet werden kann.

	A	B	C	D	E	F	G	H
1	14_Specials_Finance							
2	G6	=RMZ(D6/12;D5;-D4)		D4	10000		C3	Monatliche Zahlung
3	G13	=ZINSZ(D13;D12;D12;-D11)		D5	36		C4	Betrag
4	G20	=ZW(D20;D19;-;D18)		D6	0,05		C5	Laufzeit
5	G27	=ZINS(D25;-D26;D27)		D11	10000		C6	Zinssatz
6	G33	=KAPZ(D31/12;1;24;-D33)		D12	1		F6	mtl. Zahlung
7				D13	0,05		C10	Zinsberechnung
8				D18	10000		C11	Betrag
9				D19	10		C12	Laufzeit
10				D20	0,05		C13	Zinssatz

Abbildung 404: Formeln des aktiven Tabellenblattes ermitteln

Direkt nach der Variablen-Deklaration wird der Name des aktiven Tabellenblattes ermittelt und an die Variable `strActWS` übergeben. So können wir dieses Tabellenblatt später wieder ansprechen, denn es sollen nur die Werte dieses Tabellenblattes ermittelt werden.

In der zweiten `For...Each`-Schleife wird mittels einer `If...Then...Else`-Entscheidung geprüft, ob die Zellen des Tabellenblattes Formeln, Werte oder Text enthalten. Die Zellenbezüge `Address` sowie die darin enthaltenen Werte werden in die entsprechenden Spalten ausgegeben. Nach dem Ausführen des Codes enthält die Spalte B sämtliche Formeln in Textform. In der Spalte E werden die Werte angezeigt und in Spalte H der Text. Die Zellen des Tabellenblattes, aus dem wir die Zellen ermitteln, die Formeln enthalten, werden zudem in roter Schriftfarbe hinterlegt.

Wenn in dem Tabellenblatt keine Formeln gefunden wurden, dann wird eine entsprechende Meldung in einem Meldungsfenster `MsgBox` angezeigt.

In der `With`-Anweisung werden die Spalten in der optimalen Breite ausgerichtet. Das Tabellenblatt mit den neuen Informationen wird aktiviert.

Am Ende der Prozedur wird im Formeln-Übersichtsblatt in der Zelle A1 der Name des Tabellenblattes ausgegeben, dessen Zellen auf Formeln überprüft wurden. Die Zelle A1 wird zudem mit einem Hyperlink versehen, der zurück zu dem Tabellenblatt führt, dessen Formeln, Werte und Text wir ermittelt haben.

```
'=====
' Auf CD      Buchdaten\Beispiele\14_Specials\
' Dateiname  14_04_LocateFormulas.xls
' Modul      mdl_01_Formulas
'=====
```

```
Sub LocateFormulas()
    Dim c As Range
    Dim i As Integer
    Dim strActWS As String
    Dim ws As Worksheet
    Dim blnMatch As Boolean
    Dim strHyperLink As String
    Dim wsFormulas As Worksheet

    strActWS = ActiveWorkbook.ActiveSheet.Name
```

```
' Prüfen, ob das Tabellenblatt für die Ausgabe der Formeln
' bereits vorhanden ist. Wenn nicht, dann neu erstellen.
```

<pre>blnMatch = False For Each ws In ActiveWorkbook.Worksheets If ws.Name = "14_Specials_Formulas" Then blnMatch = True Exit For End If Next ws If Not blnMatch Then ActiveWorkbook.Worksheets.Add.Name = "14_Specials_Formulas" End If Set wsFormulas = Worksheets("14_Specials_Formulas") ' Formeln-Ausgabeblatt neu aufbereiten With wsFormulas .Range("A:IV").Delete .Range("A1").Value = ActiveSheet.Name End With ' Jede Zelle des gewünschten Tabellenblatts wird geprüft For Each c In Worksheets(strActWS).UsedRange ' Prüfen, ob es sich um eine FORMEL handelt If c.HasFormula Then ' Zellbezug in Spalte A schreiben wsFormulas.Cells(Rows.Count, 1). _ End(xlUp).Offset(1, 0).Value = c.Address(False, False) ' Formel in Spalte B schreiben wsFormulas.Cells(Rows.Count, 2). _ End(xlUp).Offset(1, 0).Value = _ " " & c.FormulaLocal ' Formeln in roter Schrift hinterlegen c.Font.ColorIndex = 3 i = i + 1 ' Prüfen, ob es sich um einen WERT handelt ElseIf IsNumeric(c) And c <> "" Then ' Zellbezug in Spalte D schreiben wsFormulas.Cells(Rows.Count, 4). _ End(xlUp).Offset(1, 0).Value = c.Address(False, False) ' Wert in Spalte E schreiben wsFormulas.Cells(Rows.Count, 5). _ End(xlUp).Offset(1, 0).Value = " " & c i = i + 1 ' Prüfen, ob es sich um einen TEXT handelt ElseIf IsNumeric(c) = False And c <> "" Then ' Zellbezug in Spalte G schreiben wsFormulas.Cells(Rows.Count, 7). _ End(xlUp).Offset(1, 0).Value = c.Address(False, False) ' Text in Spalte F schreiben wsFormulas.Cells(Rows.Count, 8). _</pre>	Grundlagen Allgemein Datum und Zeit Diagramme Steuerelemente Befehlsliste Objekte Ereignisse User-Formen Web/Mail Extern Gemischtes Spezial

```

        End(xlUp).Offset(1, 0).Value = " " & c
        i = i + 1
    End If
Next c

' Meldung, wenn keine Formeln gefunden wurden
If i = 0 Then
    MsgBox "Dieses Tabellenblatt enthält keine Formeln", _
        vbInformation
    Exit Sub
End If

' Wenn Formeln gefunden wurden,
' dann Spalten ausrichten und Formeln-Blatt aktivieren
With wsFormulas
    .Range("A:IV").Columns.AutoFit
    .Activate
End With

' Einen Hyperlink in Zelle A1 einfügen, der zu dem Blatt
' führt, aus dem die Formeln stammen
strHyperLink = " " & wsFormulas.Range("A1").Value & "!A1"
Worksheets(strActWS).Range("A1").Hyperlinks.Add _
    Anchor:=Selection, Address:="", SubAddress:= _
    strHyperLink, TextToDisplay:=[A1].Value

Set wsFormulas = Nothing
End Sub

```

481 Sprachtrainer

Der Sprachtrainer ist so aufgebaut, dass jede Sprache gelernt werden kann, und zwar in beide Richtungen. Sie können beispielsweise in der Spalte A die Wörter in Deutsch erfassen und in der Spalte B die Wörter in Englisch. Wenn Sie die Wörter in der Gegenrichtung abfragen möchten, können Sie einfach die Spalten vertauschen.

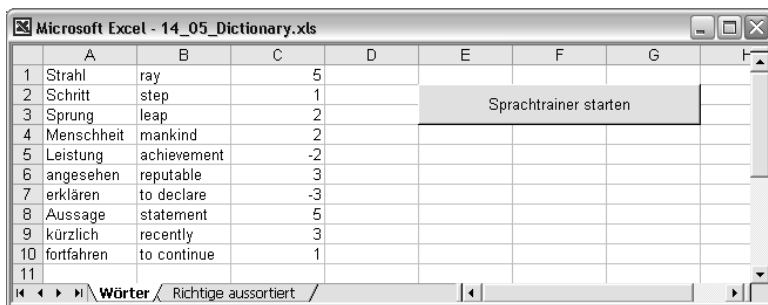


Abbildung 405: Sprachtrainer

Wenn Sie den Sprachtrainer über die Schaltfläche im Tabellenblatt der Beispieldatei starten, wird ein Eingabefenster `InputBox` angezeigt. Das Eingabefenster zeigt das deutsche Wort an und erwartet die Eingabe in Englisch. Bei richtiger Eingabe wird ein entsprechendes Meldungsfenster `MsgBox`

angezeigt. Bei falscher Eingabe wird ebenfalls ein Meldungsfenster angezeigt, diesmal jedoch mit dem Hinweis auf das korrekte englische Wort. Die Prozedur kann jederzeit abgebrochen werden.

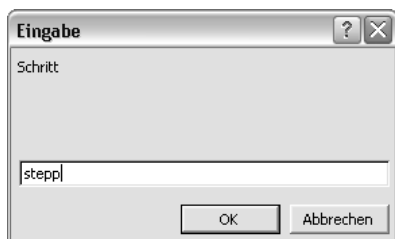


Abbildung 406: Eingabefenster und Fehlermeldung mit richtiger Ausgabe

Damit Wörter, die mehrmals richtig beantwortet wurden, nicht immer wieder abgefragt werden, werden sie aussortiert. Bei richtiger Antwort wird der Wert 1 in die Spalte C geschrieben, bei falscher Antwort der Wert -1. Die Zahl wird bei richtiger oder falscher Antwort jeweils um den Wert 1 erhöht oder reduziert. Wenn der Wert größer als 5 ist, dann wird die Zeile ausgeschnitten und ins zweite Tabellenblatt kopiert. Damit gehen die Wörter nicht verloren, sondern werden lediglich aus der aktuellen Abfrage entfernt.

Wenn das Programm gestartet wird, werden die Spalten A und B ausgeblendet, damit die Wörter nicht sichtbar sind. Beim Verlassen der Prozedur werden die beiden Spalten wieder eingeblendet. Zuvor wird in einer If-Entscheidung geprüft, ob überhaupt Wörter in der Tabelle vorhanden sind.

Die Wörter werden nicht der Reihe nach, sondern nach einem Zufallsprinzip Rnd abgefragt. Zu Beginn der Prozedur wird gezählt, wie viele Zeilen in der Tabelle Wörter enthalten. Es werden entsprechend viele Wörter abgefragt. Es kann dabei vorkommen, dass einzelne Wörter mehrmals abgefragt werden und andere gar nicht. Damit nicht zweimal aufeinander dasselbe Wort abgefragt wird, verwenden wir die öffentliche Variable `intStatic`, die außerhalb der Prozedur deklariert wird. Diese Variable speichert jeweils das letzte Wort und vergleicht dieses bei erneutem Durchgang der Schleife mit dem neuen Zufallswort. Eine `Do...Loop`-Schleife wird so lange durchlaufen, bis ein ungleiches Wort abgefragt werden kann.

Am Ende der Prozedur wird rückgefragt, ob noch eine Runde gelernt werden soll. Wenn die Frage mit Ja beantwortet wird, wird die Prozedur rekursiv aufgerufen. Das heißt, die Prozedur ruft sich selber wieder auf.

Wenn die Prozedur abgeschlossen ist oder vorzeitig verlassen wird, dann wird in einer Dialogbox angezeigt, wie viele Fehler gemacht wurden. Wenn Sie fehlerfrei abschließen konnten, wird ebenfalls eine entsprechende Meldung angezeigt.

Die Beschreibung zu den einzelnen Codeabschnitten finden Sie in kursiver Schreibweise direkt in der Prozedur.

Grundlagen

Allgemein

Daten
Zeit

Funktionen

Steuerelemente

Befehle
leisten

Objekte

Diagramme

Ereignisse

User-Formen

Web/
Mail

Extern

Gemischtes

Spezial

```
'=====
' Auf CD      Buchdaten\Beispiele\14_Specials\
' Dateiname   14_05_Dictionary
' Modul       mdl_01_Dictionary
'=====
```

Option Explicit

```
Public intStatic As Integer
```

```
Sub Dictionary()
    Dim strLeftCol As String
    Dim strRightCol As String
    Dim intError As Integer
    Dim varAnswer As Variant
    Dim intRow As Integer
    Dim intCol As Integer
    Dim i As Integer
    Dim ws2 As Worksheet
    Dim strQuestion As String

    Set ws2 = Worksheets("Richtige aussortiert")

    ' Anzahl benutzte Zeilen ermitteln
    intCol = ActiveSheet.UsedRange.Columns(1).Cells.Count

    ' Abbrechen, wenn keine Wörter vorhanden sind
    If intCol = 1 Then
        MsgBox "Es ist keine Abfrage vorhanden, " & _
            "oder eine Abfrage lohnt sich nicht mehr."
        Exit Sub
    End If

    ' Spalten mit Wörtern ausblenden
    Columns("A:B").EntireColumn.Hidden = True

    ' Sprachtrainer starten
    For i = 1 To intCol

        ' Zufallsgenerator für Wörter
        intRow = Int((intCol * Rnd) + 1)

        ' Schleife, damit nicht zweimal dasselbe Wort hintereinander
        ' abgefragt wird
        Do While intRow = intStatic
            If intCol = 1 Then
                ' Wenn nur ein Wort übrig ist, Prozedur verlassen
                MsgBox "Eine Abfrage lohnt sich nicht mehr. " & _
                    "Es wird abgebrochen"
                Columns("A:B").EntireColumn.Hidden = False
                Exit Sub
            End If
            intRow = Int((intCol * Rnd) + 1)
        Loop
    Next i
End Sub
```

Loop

```

' Werte an Variablen übergeben
intStatic = intRow
strLeftCol = Cells(intRow, 1).Value
strRightCol = Cells(intRow, 2).Value

varAnswer = Application.InputBox(strLeftCol)
' Wenn der "Abbrechen"-Button geklickt wird,
' dann Prozedur vorzeitig verlassen
If varAnswer = 0 Then
    Columns("A:B").EntireColumn.Hidden = False

    ' Falls Fehler gemacht wurden, dann anzeigen
    ' wieviele es waren
    If intError <> 0 Then
        MsgBox "Anzahl falsche Antworten: " & _
            intError & " von " & i
        Exit Sub
    Else
        MsgBox "Bravo! Du hast keinen Fehler gemacht!"
    End If
    Exit Sub

' Bei richtiger Antwort eine Meldung ausgeben und
' den Zähler um 1 erhöhen
ElseIf varAnswer = strRightCol Then
    MsgBox ("Richtig!")
    Cells(intRow, 1).Offset(0, 2).Value = _
        Cells(intRow, 1).Offset(0, 2).Value + 1

    ' Wenn der Zähler 6 ist, dann den Datensatz ausschneiden
    ' und in das zweite Tabellenblatt schreiben
    ' (aussortieren)
    If Cells(intRow, 1).Offset(0, 2).Value = 6 Then
        Cells(intRow, 1).EntireRow.Cut
        ws2.Paste _
            Destination:=ws2. _
                Cells(Rows.Count, 1).End(xlUp).Offset(1, 0)
        If ws2.Range("A1") = "" Then
            ws2.Rows(1).Delete
        End If
        Cells(intRow, 1).EntireRow.Delete
    End If

' Bei falscher Antwort die richtige Antwort
' in einer Dialogbox anzeigen
ElseIf varAnswer <> strRightCol Then
    MsgBox "Falsch: " & Cells(intRow, 1).Offset(0, 1).Value

    ' Den Zähler um 1 reduzieren
    Cells(intRow, 1).Offset(0, 2).Value = _
        Cells(intRow, 1).Offset(0, 2).Value - 1

```

Grund-
lagenAllge-
meinDatu
Zeit

tungen

Steuer
elemenBefehl
leisten

Objekt

Dia-
grammEreig-
nisseUser-
FormWeb/
Mail

Extern

Ge-
misch
tes

Specia

```

        intError = intError + 1
    End If
Next i

' Falls Fehler gemacht wurden, dann anzeigen, wieviele es waren
If intError <> 0 Then
    MsgBox "Anzahl falsche Antworten: " & intError & " von " & i
Else
    MsgBox "Bravo! Du hast keinen Fehler gemacht!"
End If

' Ausgeblendete Spalten wieder einblenden
Columns("A:B").EntireColumn.Hidden = False

' Rückfrage, ob noch eine Runde gelernt werden soll
' Wenn "ja", wird die Prozedur rekursiv wieder aufgerufen
strQuestion = MsgBox("Noch eine Runde?", vbYesNo)
If strQuestion = vbYes Then
    Call Dictionary
End If

Set ws2 = Nothing
End Sub

```

Um die Sprache in die Gegenrichtung zu lernen, können Sie die Spalten vertauschen, indem Sie die Spalte B ausschneiden und vor der Spalte A wieder einfügen. Bei erneutem Ausführen der Prozedur werden die Spalten wieder vertauscht.

```

' =====
' Auf CD      Buchdaten\Beispiele\14_Specials\
' Dateiname   14_05_Dictionary
' Modul       mdl_02_ChangeLanguage
' =====

Sub ChangeLanguage()
    Columns("B").Cut
    Columns("A").Insert Shift:=xlToRight
End Sub

```

482 Tabellenblatt-Übersicht in Menüleiste

Manche Excel-Arbeitsmappen enthalten eine große Anzahl Tabellen- und/oder Diagrammblättern. Es kann vorkommen, dass nicht alle Register in der Leiste der Blattregister Platz finden und angezeigt werden. Über die Pfeiltasten an der linken äußeren Seite der Registerleiste können Sie die Registeransicht verschieben, um nicht sichtbare Blattregister anzuzeigen und auszuwählen (siehe Abbildung 407). Mit einem Rechtsklick auf den Pfeiltastenbereich öffnet sich ein Kontextmenü mit einer Übersicht aller vorhandenen Blätter der Arbeitsmappe. Mit einem Klick auf eines der Blätter in diesem Kontextmenü wird das ausgewählte Blatt selektiert.

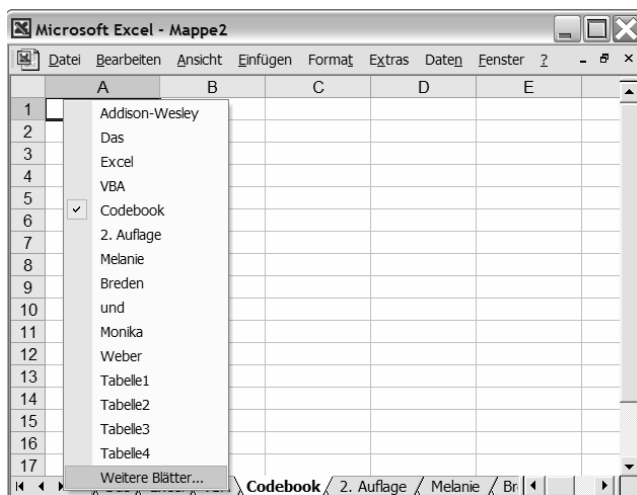


Abbildung 407: Integrierte Blattübersicht

Eine weitere Möglichkeit zur besseren Übersicht und bequemerem Blattausswahl bietet ein selbst erstelltes DropDown-Feld in der Menüleiste von Excel. Selbst wenn Sie über das Menü EXTRAS | OPTIONEN, Registerkarte ANSICHT die Option BLATTREGISTERKARTEN deaktivieren und damit die Blattregister ausgeblendet sind, ist noch eine Auswahl möglich. Wenn die Blattregister eingeblendet sind und Sie über diese einen manuellen Blattwechsel vornehmen, wird der Name des jeweils aktiven Arbeitsblattes im DropDown-Feld angezeigt.

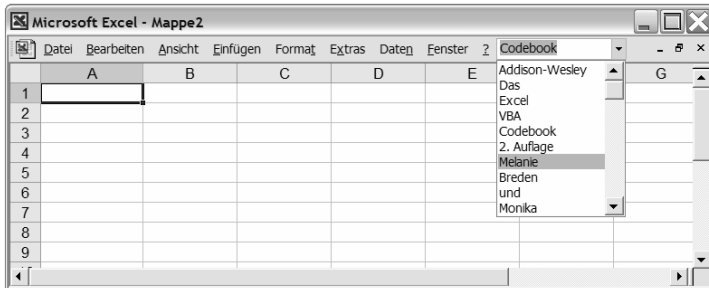


Abbildung 408: Blätter der aktiven Excel-Arbeitsmappe über Dropdownfeld anwählen

Damit Ihnen diese Funktionalität in allen geöffneten Excel-Arbeitsmappen zur Verfügung steht, haben wir die Datei zunächst als normale Excel-Arbeitsmappe erstellt und anschließend als Add-In gespeichert. Um aus einer Excel-Arbeitsmappe mit der Dateieindung *.xls ein Add-In mit der Dateieindung *.xla zu erstellen, speichern Sie die fertige *.xls-Datei im *.xla-Dateiformat ab. Klicken Sie dazu in der Excel-Ansicht im Menü DATEI auf SPEICHERN UNTER und wählen Sie als Dateityp den untersten Eintrag in der Liste *Microsoft Excel-Add-In (*.xla)*.

Tipp

Weitere Informationen zum Thema Add-Ins finden Sie in der Kategorie »Gemischtes«.

Grundlagen

Allgemein

Datenzeit

Funktionen

Steuerelemente

Befehle leisten

Objekt

Diagramm

Ereignisse

User-Form

Web/Mail

Extern

Gemischtes

Spezial

Speichern Sie das Add-In im Ordner *C:\Dokumente und Einstellungen\<Benutzername>\Anwendungsdaten\Microsoft\AddIns* ab, damit es im Menü EXTRAS | ADD-INS angezeigt wird. Durch die Aktivierung des Eintrages in der Liste der vorhandenen Add-Ins wird es installiert und steht bei jeder Excel-Sitzung zur Verfügung. Sie können das Add-In auch in einen anderen Ordner speichern und die *.xla-Datei im ADD-INS-Manager über die Schaltfläche DURCHSUCHEN auf Ihrer Festplatte suchen und auf diese Weise installieren.

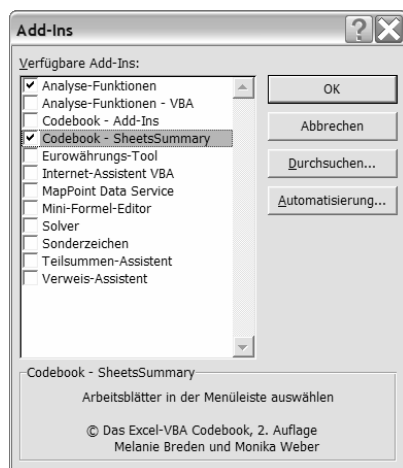


Abbildung 409: Liste der vorhandenen und installierten Add-Ins

Die Tabellenübersicht im DropDown-Feld soll immer alle sichtbaren Blätter der momentan aktiven Excel-Arbeitsmappe aufweisen. Dazu sind einige Application-Ereignisse erforderlich, die in allen geöffneten Arbeitsmappen überwacht werden.

Tip

In der Kategorie »Gemischtes« finden Sie ausführliche Beispiele zu integrierten Application-Ereignissen.

Der Verweis `pEvents` auf die Application-Ereignisse in `clsEvents` sowie der Prozeduraufruf `AddSheetsSummary` zur Erstellung des DropDown-Feldes wird im `Workbook_Open`-Ereignis des Add-Ins `CODEBOOK – SHEETSUMMARY` vorgenommen. Dieses Ereignis wird ausgelöst, wenn das Add-In installiert oder die Excel-Anwendung gestartet wird, während das Add-In bereits installiert ist. Wenn Sie die Excel-Anwendung schließen oder das Add-In deinstallieren, erfolgt im `Workbook_BeforeClose`-Ereignis der Prozeduraufruf `DeleteSheetsSummary`, um das DropDown-Feld zu löschen. Es ist nicht notwendig, die öffentliche Ereignis-Variable `pEvents` zurückzusetzen, da diese bei Beendigung der Datei automatisch aus dem Speicher gelöscht wird.

```
'=====
' Auf CD      Buchdaten\Beispiele\14_Specials\
' Dateiname   14_06_SheetsSummary.xla
' Ereignis    DieseArbeitsmappe
'=====
```

```
Private Sub Workbook_Open()
    ' Application-Ereignisse zur Verfügung stellen
```

```
Set pEvents = New clsEvent
' Dropdownfeld in Menüleiste erstellen
AddSheetsSummary
End Sub

Private Sub Workbook_BeforeClose(Cancel As Boolean)
' Dropdownfeld in Menüleiste löschen
DeleteSheetsSummary
End Sub
```

In einem Standardmodul erfolgt die Deklaration der Variablen `pEvents` der Klasse `clsEvents`, welche die Application-Ereignisse verwaltet. Durch die Deklaration der Variablen `ctlComboBoxHandler` wird eine neue Instanz der Klasse `ComboBoxHandler` erstellt. Diese verwaltet das Change-Ereignis des `CommandBarComboBox`-Objektes, das ein eigenständiges Klassenmodul benötigt.

Bevor das DropDown-Feld in der Excel-Menüleiste erzeugt wird, deklarieren wir die Variable `newCombo` vom Typ `Office.CommandBarComboBox`. An diese wird ein Verweis auf das zu überwachende DropDown-Feld gesetzt. Die Zeile

```
ctlComboBoxHandler.SyncBox newCombo
```

bewirkt den Aufruf der Prozedur `SyncBox` im Klassenmodul `ComboBoxHandler`. Dabei wird die Variable `newCombo` übergeben. Das Change-Ereignis des `CommandBarComboBox`-Objektes wird dadurch auf das übergebene Steuerelement `newCombo` gesetzt. Wird die Auswahl im DropDown-Feld BLATT-AUSWAHL geändert, wird das `ComboBoxEvent_Change`-Ereignis in `ComboBoxHandler` ausgelöst und die dort hinterlegten Befehle und Prozeduren ausgeführt.

In einer `For Each...Next`-Anweisung werden nur die sichtbaren Blätter der aktiven Excel-Arbeitsmappe mit der `AddItem`-Methode dem DropDown-Feld hinzugefügt. Excel würde eine Fehlermeldung ausgeben, wenn Sie versuchen würden, ein ausgeblendetes Arbeitsblatt zu selektieren. Anschließend wird über die `Width`-Eigenschaft des DropDown-Feldes dessen Breite in Pixel angegeben. Über die `Caption`-Eigenschaft `Blatt-Auswahl` kann das Steuerelement vor dem Löschvorgang desselben identifiziert werden.

In dem ungeöffneten DropDown-Feld soll immer der Name des aktiven Arbeitsblattes zu lesen sein. Dies erreichen wir über die `Text`-Eigenschaft des `CommandBarComboBox`-Objektes.

```
newCombo.Text = ActiveSheet.Name
```

Die Prozedur `DeleteSheetsSummary` stellt sicher, dass ein eventuell bereits vorhandenes Steuerelement gleichen Namens gelöscht wird, bevor die neue Arbeitsblattauflistung im DropDown-Feld Blatt-Auswahl erstellt wird.

```
'=====
' Auf CD      Buchdaten\Beispiele\14_Specials\
' Dateiname  14_06_SheetsSummary.xla
' Modul      mdl_01_AddComboBox
'=====

' Ereignis-Variablen deklarieren
Public pEvents As clsEvent
```

Grund-lagen

Allge-mein

Datu Zeit

tungen

Steuer elemen

Befehl leisten

Objekt

Dia-gramm

Ereig-nisse

User-Form

Web/Mail

Extern

Ge-misch tes

Specia

```

' neue Instanz des Change-Ereignisses erstellen
Private ctlComboBoxHandler As New ComboBoxHandler

Sub AddSheetsSummary()
    Dim newCombo As Office.CommandBarComboBox
    Dim objSh As Object

    ' Evtl. vorhandenes Dropdownfeld löschen
    DeleteSheetsSummary

    ' Neues Dropdownfeld in Menüleiste erstellen
    Set newCombo = Application.CommandBars(1).Controls _
        .Add(msoControlComboBox, Temporary:= True)

    With newCombo
        On Error Resume Next

        ' Alle sichtbaren Blätter der aktiven Datei dem
        ' Dropdownfeld hinzufügen
        For Each objSh In ActiveWorkbook.Sheets
            If objSh.Visible = True Then .AddItem objSh.Name
        Next objSh

        .Width = 120
        .DropDownLines = 10
        .Caption = "Blatt-Auswahl"
        .Text = ActiveSheet.Name
    End With

    ' Aufruf der Prozedur SyncBox im Klassenmodul
    ' Steuerelement an Eventhandler übergeben
    ctlComboBoxHandler.SyncBox newCombo
End Sub

Sub DeleteSheetsSummary()
    On Error Resume Next
    Application.CommandBars(1).Controls("Blatt-Auswahl") _
        .Delete
End Sub

```

Im Klassenmodul `clsEvents` werden die benötigten `Application`-Ereignis-Prozeduren hinterlegt. Die Funktionsweise entnehmen Sie den Kommentaren im Code.

<pre>' ===== ' Auf CD Buchdaten\Beispiele\14_Specials\ ' Dateiname 14_06_SheetsSummary.xla ' Klasse clsEvent ' ===== ' Objektvariable zur Reaktion auf Ereignisse Public WithEvents xlApp As Application ' neue Klasse erstellen Private Sub Class_Initialize() Set xlApp = Application End Sub ' Bei manuellem Blattwechsel wird der Name des aktiven ' Arbeitsblattes im Dropdownfeld angezeigt Private Sub xlApp_SheetActivate(ByVal Sh As Object) Application.CommandBars(1).Controls("Blatt-Auswahl") _ .Text = ActiveSheet.Name End Sub ' Um Änderungen der Registerbeschriftungen zu übernehmen, wird bei ' jedem Zellenwechsel in jedem Tabellenblatt in jeder Excel-Datei ' die Auflistung im Dropdownfeld aktualisiert. Private Sub xlApp_SheetSelectionChange(ByVal Sh As Object, _ ByVal Target As Range) AddSheetsSummary End Sub ' Beim Wechsel in eine andere Excel-Arbeitsmappe wird ' Dropdownliste neu erstellt Private Sub xlApp_WorkbookActivate(ByVal Wb As Workbook) AddSheetsSummary End Sub ' Wird ein neues Tabellenblatt eingefügt, wird ' Dropdownliste neu erstellt Private Sub xlApp_WorkbookNewSheet(ByVal Wb As Workbook, _ ByVal Sh As Object) AddSheetsSummary End Sub</pre>	Grundlagen Allgemein DatumZeit tungen Steuer elementen Befehl leisten Objekt Diagramm Ereignisse User-Form Web/Mail Extern Gemischtes Spezial

Im Klassenmodul `ComboBoxHandler` wird über das Schlüsselwort `WithEvents` die Variable `ComboBoxEvent` als Ereignis des `CommandBarComboBox`-Objektes deklariert. Wird die Auswahl im DropDown-Feld geändert, wird über die Ereignis-Prozedur `ComboBoxEvent_Change` das ausgewählte Blatt in der aktiven Datei selektiert. Wenn die Blattregisterbeschriftungen geändert wurden und noch keine neue Auflistung im DropDown-Feld erfolgt ist, gibt Excel eine Fehlermeldung aus. Diese verhindern wir durch die `On Error Resume Next`-Anweisung.

```

' =====
' Auf CD      Buchdaten\Beispiele\14_Specials\
' Dateiname   14_06_SheetsSummary.xla
' Klasse      ComboBoxHandler
' =====

' Objektvariable zur Reaktion auf Ereignisse
Private WithEvents ComboBoxEvent As Office.CommandBarComboBox

' Steuerelement an Eventhandler übergeben
Public Sub SyncBox(Box As Office.CommandBarComboBox)
    Set ComboBoxEvent = Box
End Sub

' Auswähländerungen in Dropdownfeld überwachen
Private Sub ComboBoxEvent_Change(ByVal Ctrl As _
                                Office.CommandBarComboBox)

    On Error Resume Next
    ' Blatt aus Dropdownfeld selektieren
    Sheets(Ctrl.Text).Select
End Sub

```

483 Präsentation in Excel

Wenn Sie mehrere Tabellen, Diagramme oder Bilder in Form einer Präsentation nacheinander anzeigen lassen wollen, dann verwenden Sie dazu beispielsweise das Präsentationsprogramm »Power-Point« von Microsoft. Mit einer Prozedur ist es möglich, einzelne Tabellen- sowie Diagrammblätter einer Excel-Datei für eine vorgegebene Dauer, ähnlich einer Präsentation, nacheinander anzuzeigen. Über eine `For...Next`-Anweisung werden die einzelnen Blätter der Reihe nach selektiert und angezeigt. Geben Sie die Dauer der jeweiligen Blattanzeige über die `Wait`-Methode an. Im folgenden Codeabschnitt wird jedes nicht ausgeblendete Blatt der aktiven Datei für jeweils zehn Sekunden angezeigt.

```

For intI = 1 To Sheets.Count
    If Sheets(intI).Visible = xlSheetVisible Then
        Sheets(intI).Select
        Application.Wait (Now + TimeValue("0:00:10"))
    End If
Next

```

In der Beispieldatei finden Sie eine solche Präsentation. Klicken Sie im Tabellenblatt MERLIN auf die angezeigte Grafik, um die zugewiesene Prozedur zu starten.



Abbildung 410: Start-Button der Präsentation

```
'=====
' Auf CD      Buchdaten\Beispiele\14_Specials\
' Dateiname  14_07_ExcelPresentation.xls
' Modul      mdl_01_Merlin
'=====
```

```
Private Sub StartMerlin()
    Dim intI As Integer

    ' Leeres Tabellenblatt als Zwischenbild einblenden
    With Sheets(2)
        .Visible = xlSheetVisible
        .Select
    End With

    ' Excel-Leisten ausblenden
    MyShow False

    ' Tabellenblätter einzeln anzeigen
    For intI = 3 To Worksheets.Count

        With Sheets(intI)
            ' verstecktes Tabellenblatt anzeigen
            .Visible = xlSheetVisible
            .Select

            ' Blattanzeige für 1 Sekunde
            Application.Wait (Now + TimeValue("0:00:01"))

            ' angezeigtes Tabellenblatt wieder verstecken
            .Visible = xlVeryHidden
        End With

        Next

        ' Dummyblatt verstecken
        Sheets(2).Visible = xlVeryHidden

        ' Excel-Leisten einblenden
        MyShow True
    End Sub
```

```
Private Sub MyShow(blnVisible As Boolean)
    With Application
        ' Bildschirmaktualisierung ausschalten
        .ScreenUpdating = False
        ' Ganzen Bildschirm aus-/einblenden
        .DisplayFullScreen = Not blnVisible

        ' Symbolleiste "Ganzer Bildschirm" ausblenden
        .CommandBars("Full Screen").Visible = False
    End With
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

```

' Menüleiste aus-/einblenden
.CommandBars(1).Enabled = blnVisible

' Bildschirmaktualisierung einschalten
.ScreenUpdating = True
End With

' Blattregisterkarten aus-/einblenden
ActiveWindow.DisplayWorkbookTabs = blnVisible
End Sub

```

484 Auto-Berechnung aus Statuszeile in Zwischenablage einfügen

Wenn Sie, ohne eine Formel anzuwenden, die Summe eines markierten Zellbereichs ermitteln wollen, verwenden Sie die Funktion SUMME der AUTO-BERECHNEN-Befehlsleiste in der Statuszeile. Mit einem Rechtsklick auf die Statuszeile wird das Kontextmenü AUTOCALCULATE angezeigt (siehe Abbildung 411). Durch einen Klick auf einen Eintrag im Kontextmenü wird die gewählte Funktion aktiviert. Diese eingestellte Funktion gilt bis auf Widerruf in allen geöffneten Arbeitsmappen der Excel-Anwendung.

Wenn Sie in einem Tabellenblatt mehrere zusammenhängende oder einzelne Zellen markieren, zeigt Excel das Ergebnis dieser Markierung je nach eingestellter Funktion in der Statuszeile an.

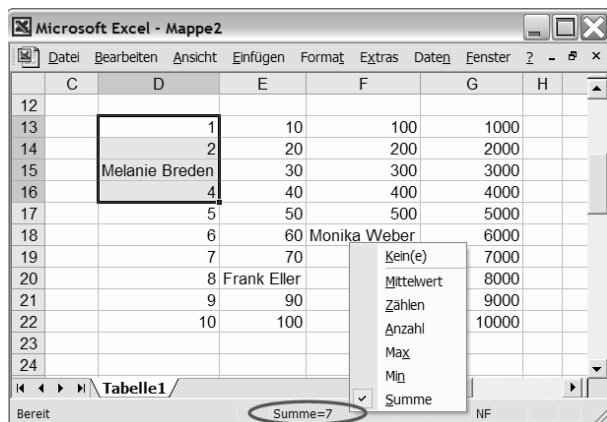


Abbildung 411: Funktionsauswahl mit Ergebnis der aktuellen Markierung

Sobald die Markierung aufgehoben wird, erlischt die Anzeige in der Statuszeile. Oftmals möchte man aber diesen ermittelten Wert weiterverwenden. Dazu müssen Sie standardmäßig eine Formel mit der gewünschten Funktion und Bezugsangabe in einer Zelle hinterlegen. Es wäre sehr hilfreich, den angezeigten Wert der Statuszeile in die Zwischenablage kopieren zu können. Inhalte der Zwischenablage können nicht nur in Excel, sondern in allen Anwendungen eingefügt werden.

Hinweis

Es ist nicht möglich, den angezeigten Wert direkt aus der Statuszeile abzufragen. Der angezeigte Wert muss über eine benutzerdefinierte Funktion neu berechnet werden.

Mit den Prozeduren dieses Rezeptes wird es möglich, über die Tastenkombination `[Strg] + [⇧] + [C]` den angezeigten Wert der Statuszeile in die Zwischenablage zu kopieren.

Im ersten Schritt haben wir die benutzerdefinierte Funktion `GetAutoCalculate` programmiert. Sie liefert als Rückgabewert das Ergebnis der aktivierten Funktion aus der Statuszeile. In der Argumentenliste der Funktion wird der optionale Parameter `rngArea` deklariert. Beim Aufruf der Funktion können Sie auch den Zellenbereich, der berechnet werden soll, an den Parameter `rngArea` übergeben. Wenn Sie keinen Zellbereich angeben, wird der markierte Bereich über die `Selection`-Eigenschaft der Variablen `rngArea` zugewiesen.

Zunächst muss ermittelt werden, welche Funktion der Benutzer im Kontextmenü `AUTO BERECHNEN` eingestellt hat. Dazu werden in einer `For Each...Next`-Anweisung alle Elemente der Befehlsleiste `AUTOCALCUATE` überprüft. Über die `State`-Eigenschaft der Schaltflächen wird ermittelt, welche als gedrückt (`msoButtonDown`) angezeigt und somit aktiviert ist. Wenn die Schaltfläche gefunden wurde, erhält die Variable `strFunction` den entsprechenden Beschriftungstext über die `Caption`-Eigenschaft.

In der `Select Case`-Anweisung wird die nötige Berechnung der eingestellten Funktion nachgestellt. Dies erreichen wir durch den Einsatz der entsprechenden Tabellenblatfunktion `Application.WorksheetFunction`. Das Ergebnis wird als Rückgabewert an die Funktion `GetAutoCalculate` übergeben und die Funktion anschließend beendet.

```
'=====
' Auf CD      Buchdaten\Beispiele\14_Specials\
' Dateiname  14_08_AutoCalculate.xls
' Modul      mdl_01_GetAutoCalculate
'=====

Function GetAutoCalculate(Optional rngArea As Range) As Variant
    Dim Ctrl As CommandBarControl
    Dim strFunction As String
    Dim AWF As WorksheetFunction

    Set AWF = Application.WorksheetFunction

    ' Wenn kein Bereich übergeben wurde,
    ' verwende markierten Zellbereich
    If rngArea Is Nothing Then Set rngArea = Selection

    For Each Ctrl In Application.CommandBars("AutoCalculate") _
        .Controls
        If Ctrl.State = msoButtonDown Then
            strFunction = Ctrl.Caption
            Exit For
        End If
    Next Ctrl

    Select Case strFunction
```

Grundlagen
Allgemein
Datum und Zeit
Formeln
Steuerungselemente
Befehlsleisten
Objekte
Diagramme
Ereignisse
User-Formen
Web/Mail
Extern
Gemischtes
Spezial

```

Case "&Kein(e)"
    GetAutoCalculate = ""
Case "&Mittelwert"
    GetAutoCalculate = AWF.Average(rngArea)
Case "&Anzahl"
    GetAutoCalculate = AWF.Count(rngArea)
Case "&Zählen"
    GetAutoCalculate = AWF.CountA(rngArea)
Case "Ma&x"
    GetAutoCalculate = AWF.Max(rngArea)
Case "Mi&n"
    GetAutoCalculate = AWF.Min(rngArea)
Case "&Summe"
    GetAutoCalculate = AWF.Sum(rngArea)
End Select
End Function

```

Die GetAutoCalculate-Funktion wird in der Prozedur PutAutoCalculateToClipboard aufgerufen. Über das DataObject-Objekt und die PutInClipboard-Methode wird das Ergebnis der Funktion in die Zwischenablage kopiert.

Die Funktionsweise des DataObject-Objekts entspricht der Funktionsweise der Zwischenablage. Wenn Sie eine Textzeichenfolge in ein DataObject-Objekt kopieren, speichert das DataObject-Objekt diese Textzeichenfolge. Wenn Sie eine zweite Zeichenfolge in demselben Format in das DataObject-Objekt kopieren, wird die erste Textzeichenfolge im DataObject-Objekt verworfen und stattdessen eine Kopie der zweiten Zeichenfolge im DataObject-Objekt gespeichert. Das Objekt speichert jeweils ein Textstück eines bestimmten Formats, wobei es sich immer um den Text der letzten Operation handelt.

Hinweis

Zur Verwendung des DataObject-Objektes ist es zwingend erforderlich, im VBA-Editor im Menü EXTRAS | VERWEISE einen Verweis auf die *Microsoft Forms 2.0 Objekt Library*-Bibliothek zu setzen.

```

' =====
' Auf CD      Buchdaten\Beispiele\14_Specials\
' Dateiname   14_08_AutoCalculate.xls
' Modul       mdl_01_AutoCalculate
' =====

```

```

Dim objData As DataObject

Sub PutAutoCalculateToClipboard()
    Set objData = New DataObject

    With objData
        .SetText GetAutoCalculate()
        .PutInClipboard
    End With
End Sub

```

Das Ergebnis der Auto-Berechnung soll mit der Tastenkombination `[Strg] [↕] [C]` in die Zwischenablage kopiert werden. Dazu wird die gewünschte Tastenkombination mit Angabe der Prozedur `PutAutoCalculateToClipboard` im `Workbook_Activate`-Ereignis der Beispieldatei an die `OnKey`-Methode übergeben.

```
' =====
' Auf CD      Buchdaten\Beispiele\14_Specials\
' Dateiname  14_08_AutoCalculate.xls
' Ereignis   DieseArbeitsmappe
' =====

Private Sub Workbook_Activate()
    Application.OnKey "+^{C}", "PutAutoCalculateToClipboard"
End Sub
```

Damit diese Funktionalität nur in der Beispieldatei ausgeführt wird, wird die verwendete Tastenkombination beim Verlassen der Arbeitsmappe im `Workbook_Deactivate`-Ereignis zurückgesetzt.

```
' =====
' Auf CD      Buchdaten\Beispiele\14_Specials\
' Dateiname  14_08_AutoCalculate.xls
' Ereignis   DieseArbeitsmappe
' =====

Private Sub Workbook_Deactivate()
    Application.OnKey "+^{C}"
End Sub
```

Tipp

Wenn Sie die Funktionalität dieses Beispiels in allen geöffneten Excel-Arbeitsmappen anwenden möchten, speichern Sie die Prozeduren aus `mdl_01_AutoCalculate` in ein Standardmodul der Datei `PERSONL.xls`. Schreiben Sie die Zuweisung der Tastenkombination lediglich in das `Workbook_Open`-Ereignis in `DIESEARBEITSMAPPE`. Ein Zurücksetzen der Tastenkombination ist nicht erforderlich. `PERSONL.xls` wird nur beim Beenden der Excel-Anwendung geschlossen und die Tastenkombination dadurch automatisch zurückgesetzt.

Sie können die Funktion `GetAutoCalculate` auch aus anderen Prozeduren aufrufen und das Ergebnis weiterverwenden. In der Prozedur `ShowAutoCalculate` wird der Bereich `D15:G15` entsprechend der eingestellten Funktion der `AUTO BERECHNUNG` ausgewertet und das Ergebnis in einem Meldungsfenster angezeigt. Abbildung 412 zeigt das Ergebnis der Funktion `ANZAHL` in der Statuszeile und im Meldungsfenster. `ANZAHL` zählt alle numerischen Werte im markierten Bereich. Wenn Sie beim Funktionsaufruf die Bezugsangabe `D15:G15` weglassen, wird bei einem Klick auf die Schaltfläche im Tabellenblatt der jeweils markierte Bereich entsprechend der gewählten Funktion berechnet und im Meldungsfenster angezeigt.

Grund-lagen

Allge-mein

Datu-Zeit

tungen

Steuer-elemen

Befehl-leisten

Objekt

Dia-gramm

Ereig-nisse

User-Form

Web/Mail

Extern

Ge-mischtes

Specia

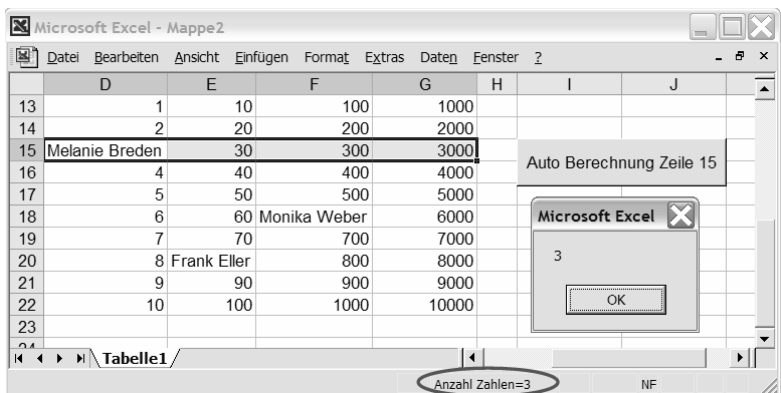


Abbildung 412: Funktionsaufruf über Schaltfläche

```
'=====
' Auf CD      Buchdaten\Beispiele\14_Specials\
' Dateiname  14_08_AutoCalculate.xls
' Modul      mdl_02_ShowAutoCalculate
'=====

Sub ShowAutoCalculate()
    MsgBox GetAutoCalculate(Range("D15:G15"))
End Sub
```

485 Menüeinträge mit Tastenkombinationen aufrufen

Allgemeine Einstellungen für die aktive Arbeitsmappe bzw. das aktive Tabellenblatt werden im Menü EXTRAS | OPTIONEN vorgenommen. Um zum Beispiel die Zeilen- und Spaltenüberschriften ein- bzw. auszublenden, müssen Sie sich durch den langen Weg über das Menü EXTRAS, Menüpunkt OPTIONEN Registerkarte ANSICHT Abschnitt FENSTEROPTIONEN Option ZEILEN- UND SPALTENÜBERSCHRIFTEN durchklicken. Wenn Sie Ihre Einstellungen des Öfteren wechseln müssen, ist dieser Weg sehr umständlich. Dieses Rezept zeigt Ihnen, wie Sie auf einfache Weise häufig benötigte Einstellungen direkt über ein selbst erstelltes Menü in der Arbeitsblatt-Menüleiste oder eine Tastenkombination wechseln können.

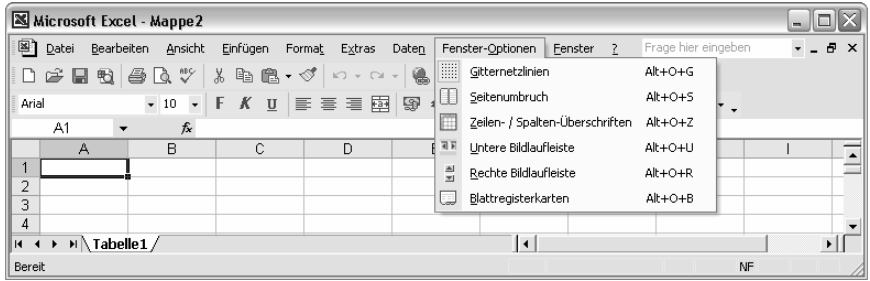


Abbildung 413: Neues Menü mit Anzeige der Tastenkombinationen

Beim Öffnen der Beispieldatei wird die Prozedur `AddWindowOptions` zum Erstellen des neuen Menüs im `Workbook_Activate`-Ereignis aufgerufen. Das neue Menü wird temporär erstellt, damit es nach Beendigung der Excel-Anwendung automatisch aus dem Speicher gelöscht wird.

```
'=====
' Auf CD      Buchdaten\Beispiele\14_Specials \
' Dateiname  14_09_WindowsOptions.xls
' Ereignis   DieseArbeitsmappe
'=====

Private Sub Workbook_Activate()
    AddWindowOptions
End Sub

Private Sub Workbook_Deactivate()
    On Error Resume Next
    Application.CommandBars(1) _
        .Controls("Fenster-Optionen").Delete
End Sub
```

Einige der in diesem Beispiel verwendeten Fenster-Optionen sind nur in der ARBEITSBLATT-MENÜLEISTE funktionsfähig. In einem Diagrammblatt zum Beispiel würde der Versuch, die Gitternetzlinien ein- oder auszublenden, zu einem Fehler führen. Dementsprechend wird das Menü FENSTER-OPTIONEN nicht der aktiven, sondern speziell der ARBEITSBLATT-MENÜLEISTE `CommandBars(1)` hinzugefügt. Um das neue Menü links neben dem FENSTER-Menü zu platzieren, verweisen Sie im `Before`-Parameter auf den Index des FENSTER-Menüs.

Die einzelnen Schaltflächen des Menüs werden über die `Set`-Anweisung erstellt. Die Eigenschaften `Style`, `Caption`, `FaceId` und `OnAction` werden in der Kategorie »Befehlsleisten« ausführlich erläutert. In der `ShortcutText`-Eigenschaft wird der Tastenkombinationstext angegeben. Die Tastenkombinationen zum Aufruf der hinterlegten Makros setzen sich aus der Taste `[Alt]`, dem Buchstaben `[O]` für das Menü FENSTER-OPTIONEN und dem unterstrichenen Buchstaben der Schaltflächenbeschriftung, zum Beispiel `[G]` für GITTERNETZLINIEN, zusammen (siehe Abbildung 413) (`.ShortcutText = "Alt+O+G"`).

Den Schaltflächen werden über die `FaceId`-Eigenschaft Symbole zugewiesen. Für die Option ZEILEN- UND SPALTENÜBERSCHRIFTEN im neuen Menü FENSTER-OPTIONEN haben wir in der Datei *FaceIDs.xls* kein passendes Symbol gefunden. Aus diesem Grund wurden im Tabellenblatt *PICTURES*, der Beispieldatei, einige Zellen beschriftet, entsprechend formatiert und eine neue Grafik daraus erstellt (siehe Abbildung 414).

Für die Option UNTERE BILDLAUFLEISTE des Menüs FENSTER-OPTIONEN haben wir das Schaltflächensymbol der vertikalen Bildlaufleiste in das Tabellenblatt *Pictures* eingefügt und über die Grafik-Symbolleiste mit einer Linksdrehung gedreht. Den neuen Grafiken werden eindeutige Namen zugewiesen. Markieren Sie dazu eine Grafik und vergeben Sie im Namensfeld, links neben der Bearbeitungszeile, einen Namen. Die genaue Vorgehensweise haben wir im Tabellenblatt *Pictures* der Beispieldatei dokumentiert (siehe Abbildung 414).

Um einer Schaltfläche ein eigenes Symbol/Grafik zuzuweisen, wird die im Tabellenblatt *Pictures* gespeicherte Grafik mit der `Copy`-Methode kopiert.

Grund-lagen

Allge-mein

Datu Zeit

tungen

Steuer elemen

Befehl leisten

Objekt

Dia-gramm

Ereig-nisse

User-Form

Web/Mail

Extern

Ge-misch tes

Specia

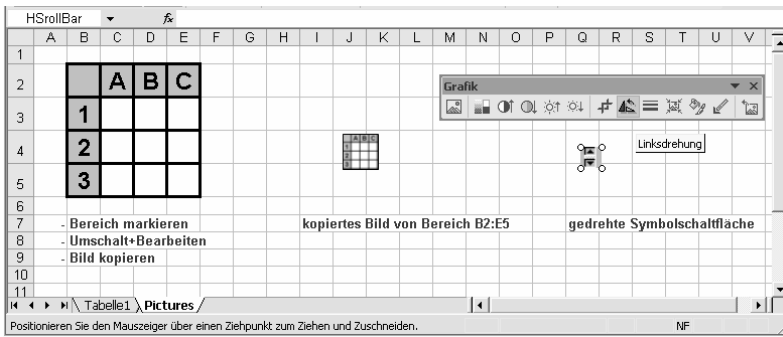


Abbildung 414: Eigene Grafiken für Schaltflächensymbole

```
Worksheets("Picture").Shapes("HScrollBar").Copy
.PasteFace
```

Die PasteFace-Methode fügt anschließend den Inhalt der Zwischenablage in das angegebene Schaltflächensteuerelement der Befehlsleiste ein. Vor Beendigung der Prozedur wird das zuletzt kopierte Bild aus der Zwischenablage gelöscht. Dazu wird der leere Zelleninhalt der Zelle IV65536 kopiert.

```
'=====
' Auf CD      Buchdaten\Beispiele\14_Specials \
' Dateiname  14_09_WindowsOptions.xls
' Modul      mdl_01_AddWindowOptions
'=====
```

```
Sub AddWindowOptions()
    Dim cmdBar As CommandBar
    Dim cmdBarCtrl As CommandBarControl

    Set cmdBar = Application.CommandBars(1)

    On Error Resume Next
    cmdBar.Controls("Fenster-Optionen").Delete
    On Error GoTo 0

    Set cmdBarCtrl = cmdBar.Controls.Add( _
        Type:=msoControlPopup, _
        Before:=cmdBar.Controls("Fenster").Index, _
        Temporary:=True)

    cmdBarCtrl.Caption = "Fenster-&Optionen"

    ' Gitternetzlinien
    With cmdBarCtrl.Controls.Add(ID:=485)
        .Style = msoButtonIconAndCaption
        .Caption = "&Gitternetzlinien"
        .OnAction = "Gridlines"
        .ShortcutText = "Alt+O+G"
    End With
```

```
' Seitenumbruch
With cmdBarCtrl.Controls.Add
    .Style = msoButtonIconAndCaption
    .Caption = "&Seitenumbruch"
    .FaceId = 278
    .OnAction = "PageBreaks"
    .ShortcutText = "Alt+O+S"
End With

' Zeilen- und Spaltenüberschriften
With cmdBarCtrl.Controls.Add
    .Style = msoButtonIconAndCaption
    .Caption = "&Zeilen- / Spalten-Überschriften"
    Worksheets("Pictures").Shapes("ABC").Copy
    .PasteFace
    .OnAction = "Headings "
    .ShortcutText = "Alt+O+Z"
End With

' untere Bildlaufleiste
With cmdBarCtrl.Controls.Add
    .Style = msoButtonIconAndCaption
    .Caption = "&Untere Bildlaufleiste"
    Worksheets("Pictures").Shapes("HScrollBar").Copy
    .PasteFace
    .OnAction = "HScrollBar"
    .ShortcutText = "Alt+O+U"
End With

' rechte Bildlaufleiste
With cmdBarCtrl.Controls.Add
    .Style = msoButtonIconAndCaption
    .FaceId = 1857
    .Caption = "&Rechte Bildlaufleiste"
    .OnAction = "VScrollBar"
    .ShortcutText = "Alt+O+R"
End With

' Blattregisterkarten
With cmdBarCtrl.Controls.Add
    .Style = msoButtonIconAndCaption
    .FaceId = 2587
    .Caption = "&Blattregisterkarten"
    .OnAction = "WorkbookTabs"
    .ShortcutText = "Alt+O+B"
End With

' Bild aus Zwischenablage löschen
Worksheets("Pictures").Range("IV65536").Copy
Application.CutCopyMode = False
End Sub
```

Grund-
lagen

Allge-
mein

Datu
Zeit

tungen

Steuer
elemen

Befehl
leisten

Objekt

Dia-
gramm

Ereig-
nisse

User-
Form

Web/
Mail

Extern

Ge-
misch
tes

Specia

Bei einem Klick auf eine Schaltfläche im Menü FENSTER-OPTIONEN oder Drücken der entsprechenden Tastenkombination wird die hinterlegte Prozedur der Schaltfläche aufgerufen und die angegebene Einstellung mittels des Not-Operators gewechselt.

```
'=====
' Auf CD      Buchdaten\Beispiele\14_Specials \
' Dateiname  14_09_WindowsOptions.xls
' Modul      mdl_02_MenuMakros
'=====

' Gitternetzlinien ein-/ausblenden
Sub Gridlines()
    With ActiveWindow
        .DisplayGridlines = Not .DisplayGridlines
    End With
End Sub

' Seitenumbruch ein-/ausblenden
Sub PageBreaks()
    With ActiveSheet
        .DisplayAutomaticPageBreaks = Not .DisplayAutomaticPageBreaks
    End With
End Sub

' Zeilen- und Spaltenüberschriften ein-/ausblenden
Sub Headings()
    With ActiveWindow
        .DisplayHeadings = Not .DisplayHeadings
    End With
End Sub

' untere Bildlaufleiste ein-/ausblenden
Sub HScrollBar()
    With ActiveWindow
        .DisplayHorizontalScrollBar = Not .DisplayHorizontalScrollBar
    End With
End Sub

' rechte Bildlaufleiste ein-/ausblenden
Sub VScrollBar()
    With ActiveWindow
        .DisplayVerticalScrollBar = Not .DisplayVerticalScrollBar
    End With
End Sub

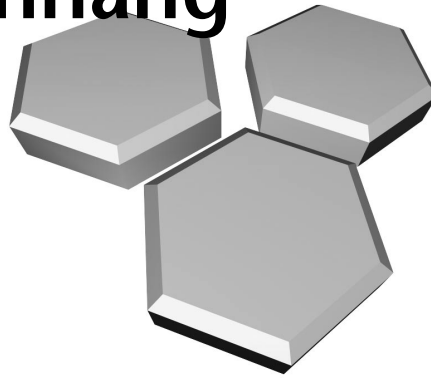
' Blattregisterkarten ein-/ausblenden
Sub WorkbookTabs()
    With ActiveWindow
        .DisplayWorkbookTabs = Not .DisplayWorkbookTabs
    End With
End Sub
```

Tipp	Wenn das neue Menü in allen Arbeitsmappen zur Verfügung stehen soll, hinterlegen Sie die Prozeduren in der Datei <i>PERSONL.xls</i>, die bei jedem Excel-Start im Hintergrund geladen wird. Das Tabellenblatt <i>PICTURES</i>, das die zu kopierenden Grafiken enthält, muss ebenfalls der Datei <i>PERSONL.xls</i> hinzugefügt werden. Gehen Sie dazu wie folgt vor: ▶ Kopieren Sie in der VBA-Umgebung die Module <code>mdl_01_AddWindowOptions</code> und <code>mdl_02_MenüMakros</code>, indem Sie diese nacheinander mit gedrückter linker Maustaste in das <code>VBAPROJECT(PERSONL.XLS)</code> ziehen. ▶ Die Ereignisse <code>Workook_Activate</code> und <code>Workook_Deactivate</code> müssen ebenfalls im Code- modul <i>DIESEARBEITSMAPPE</i> der Datei <i>PERSONL.xls</i> hinterlegt werden. Werden die Ereignisse dort bereits verwendet, fügen Sie lediglich die Befehle der beiden Prozeduren den vorhandenen Ereignis-Prozeduren hinzu, da in einem <code>VBAPROJECT</code> jedes Ereignis nur einmal vorliegen darf. ▶ Blenden Sie in der Excel-Umgebung die Datei <i>PERSONL.xls</i> über das Menü FENSTER EINBLENDEN ein. ▶ Aktivieren Sie das Tabellenblatt <i>PICTURES</i> der Beispieldatei. ▶ Klicken Sie mit der rechten Maustaste auf das Tabellenregister <i>PICTURES</i> und wählen Sie VERSCHIEBEN/KOPIEREN. ▶ Wählen Sie im DropDown-Feld der Zeile <i>ZUR MAPPE</i>: die Datei <i>PERSONL.xls</i> aus.
Tipp	▶ Aktivieren Sie die Option KOPIE ERSTELLEN, damit das Tabellenblatt in der Originaldatei nicht gelöscht wird. Eine Deaktivierung der Option würde das Tabellenblatt <i>PICTURES</i> aus der Originaldatei entfernen und in die ausgewählte Arbeitsmappe verschieben. ▶ Speichern Sie die Datei <i>PERSONL.xls</i>. ▶ Blenden Sie in der Excel-Umgebung die Datei <i>PERSONL.xls</i> über das Menü FENSTER AUSBLENDEN aus.

Grundlagen
Allgemein
Datum und Zeit
Formeln
Diagramme
Objekt
Diagramm
Ereignisse
User-Form
Web/Mail
Extern
Gemischtes
Spezial

TEIL III

Anhang



Anhang

1 Die englischen Funktionsnamen für die deutschen Tabellenblatffunktionen

Deutsche Funktion	Englische Funktion	Deutsche Funktion	Englische Funktion
ABRUNDEN	ROUNDDOWN	MAX	MAX
ABS	ABS	MDET	MDETERM
ACHSENABSCHNITT	INTERCEPT	MEDIAN	MEDIAN
ADRESSE	ADDRESS	MIN	MIN
ANZAHL	COUNT	MINUTE	MINUTE
ANZAHL2	COUNTA	MINV	MINVERSE
ANZAHLLEEREZELLEN	COUNTBLANK	MITTELABW	AVEDEV
ARCCOS	ACOS	MITTELWERT	AVERAGE
ARCCOSHYP	ACOSH	MMULT	MMULT
ARCSIN	ASIN	MODALWERT	MODE
ARCSINHYP	ASINH	MONAT	MONTH
ARCTAN	ATAN	MTRANS	TRANSPOSE
ARCTAN2	ATAN2	N	N
ARCTANHYP	ATANH	NBW	NPV
AUFRUFEN	CALL	NEGBINOMVERT	NEGBINOM-DIST
AUFRUNDEN	ROUNDUP	NICHT	NOT
BEREICH.VERSCHIEBEN	OFFSET	NORMINV	NORMINV
BEREICHE	AREAS	NORMVERT	NORMDIST
BESTIMMTHEITSMASS	RSQ	NV	NA
BETAINV	BETAINV	OBERGRENZE	CEILING
BETAVERT	BETADIST	ODER	OR
BINOMVERT	BINOMDIST	PEARSON	PEARSON
BOGENMASS	RADIANS	PI	PI
BW	PV	POISSON	POISSON
CHIINV	CHIINV	POTENZ	POWER
CHITEST	CHITEST	PRODUKT	PRODUCT

Tabelle 1: Deutsch-Englisch Funktionsnamen

Deutsche Funktion	Englische Funktion	Deutsche Funktion	Englische Funktion
CHIVERT	CHIDIST	QIKV	MIRR
CODE	CODE	QUADRATESUMME	SUMSQ
COS	COS	QUANTIL	PERCENTILE
COSHYP	COSH	QUANTILSRANG	PERCENT-RANK
DATUM	DATE	QUARTILE	QUARTILE
DATEDIF	DATEDIFF	RANG	RANK
DATWERT	DATEVALUE	RECHTS	RIGHT
DBANZAHL	DCOUNT	REGISTER.KENNUM-MER	REGISTER.ID
DBANZAHL2	DCOUNTA	REST	MOD
DBAUSZUG	DGET	RGP	LINEST
DBMAX	DMAX	RKP	LOGEST
DBMIN	DMIN	RMZ	PMT
DBMITTELWERT	DAVERAGE	RÖMISCH	ROMAN
DBPRODUKT	DPRODUCT	RUNDEN	ROUND
DBSTDABW	DSTDEV	SÄUBERN	CLEAN
DBSTDABWN	DSTDEVP	SCHÄTZER	FORECAST
DBSUMME	DSUM	SCHIEFE	SKEW
DBVARIANZ	DVAR	SEKUNDE	SECOND
DBVARIANZEN	DVARP	SIN	SIN
DIA	SYD	SINHYP	SINH
DM	DOLLAR	SPALTE	COLUMN
ERSETZEN	REPLACE	SPALTEN	COLUMNS
EXP	EXP	STABW	STDEV
EXPONVERT	EXPONDIST	STABWN	STDEVP
FAKULTÄT	FACT	STANDARDISIERUNG	STANDARDIZE
FALSCH	FALSE	STANDNORMINV	NORMSINV
FEHLER.TYP	ERROR.TYPE	STANDNORMVERT	NORMSDIST
FEST	FIXED	STEIGUNG	SLOPE
FINDEN	FIND	STFEHLERYX	STEYX
FINV	FINV	STUNDE	HOUR
FISHER	FISHER	SUCHEN	SEARCH
FISHERINV	FISHERINV	SUMME	SUM
FTEST	FTEST	SUMMENPRODUKT	SUMPRODUCT

Tabelle 1: Deutsch-Englisch Funktionsnamen (Forts.)

Deutsche Funktion	Englische Funktion	Deutsche Funktion	Englische Funktion
FVERT	FDIST	SUMMEWENN	SUMIF
GAMMAINV	GAMMAINV	SUMMEX2MY2	SUMX2MY2
GAMMALN	GAMMALN	SUMMEX2PY2	SUMX2PY2
GAMMAVERT	GAMMADIST	SUMMEXMY2	SUMXMY2
GANZZAHL	INT	SUMQUADABW	DEVSQ
GDA	DDB	SVERWEIS	VLOOKUP
GDA2	DB	T	T
GEOMITTEL	GEOMEAN	TAG	DAY
GERADE	EVEN	TAGE360	DAYS360
GESTUTZTMITTEL	TRIMMEAN	TAN	TAN
GLÄTTEN	TRIM	TANHYP	TANH
GRAD	DEGREES	TEIL	MID
GROSS	UPPER	TEILERGEBNIS	SUBTOTAL
GROSS2	PROPER	TEXT	TEXT
GTEST	ZTEST	TINV	TINV
HARMITTEL	HARMEAN	TREND	TREND
HÄUFIGKEIT	FREQUENCY	TTEST	TTEST
HEUTE	TODAY	TVERT	TDIST
HYPGEOMVERT	HYPGEOM-DIST	TYP	TYPE
IDENTISCH	EXACT	UND	AND
IKV	IRR	UNGERADE	ODD
INDEX	INDEX	UNTERGRENZE	FLOOR
INDIREKT	INDIRECT	VARIANZ	VAR
INFO	INFO	VARIANZEN	VARP
ISTBEZUG	ISREF	VARIATION	GROWTH
ISTFEHL	ISERR	VARIATIONEN	PERMUT
ISTFEHLER	ISERROR	VDB	VDB
ISTKTEXT	ISNONTEXT	VERGLEICH	MATCH
ISTLEER	ISBLANK	VERKETTEN	CON-CATENATE
ISTLOG	ISLOGICAL	VERWEIS	LOOKUP
ISTNV	ISNA	VORZEICHEN	SIGN
ISTTEXT	ISTEXT	WAHL	CHOOSE
ISTZAHL	ISNUMBER	WAHR	TRUE

Tabelle 1: Deutsch-Englisch Funktionsnamen (Forts.)

Deutsche Funktion	Englische Funktion	Deutsche Funktion	Englische Funktion
JAHR	YEAR	WAHRSCHEBEREICH	PROB
JETZT	NOW	WECHSELN	SUBSTITUTE
KAPZ	PPMT	WEIBULL	WEIBULL
KGRÖSSTE	LARGE	WENN	IF
KKLEINSTE	SMALL	WERT	VALUE
KLEIN	LOWER	WIEDERHOLEN	REPT
KOMBINATIONEN	COMBIN	WOCHENTAG	WEEKDAY
KONFIDENZ	CONFIDENCE	WURZEL	SQRT
KORREL	CORREL	WVERWEIS	HLOOKUP
KOVAR	COVAR	ZÄHLENWENN	COUNTIF
KRITBINOM	CRITBINOM	ZEICHEN	CHAR
KURT	KURT	ZEILE	ROW
KÜRZEN	TRUNC	ZEILEN	ROWS
LÄNGE	LEN	ZEIT	TIME
LIA	SLN	ZEITWERT	TIMEVALUE
LINKS	LEFT	ZELLE	CELL
LN	LN	ZINS	RATE
LOG	LOG	ZINSZ	IPMT
LOG10	LOG10	ZUFALLSZAHL	RAND
LOGINV	LOGINV	ZW	FV
LOGNORMVERT	LOGNORM-DIST	ZZR	NPER

Tabelle 1: Deutsch-Englisch Funktionsnamen (Forts.)

2 VBA-Schlüsselwörter

Deutsch	Englisch	Deutsch	Englisch
Abs	Abs	ErstelleVerz	MkDir
Zugriff	Access	Mod	Mod
Original	Alias	Modul	Module
Und	And	Monat	Month
Unbestimmt	Any	MeldungsDlG	MsgBox
AktiviereAnw	AppActivate	Benenne	Name
Anhängen	Append	Nächste	Next
Argument	ArgName	Nicht	Not

Tabelle 2: VBA-Schlüsselwörter Englisch – Deutsch

Deutsch	Englisch	Deutsch	Englisch
Datenfeld	Array	Nichts	Nothing
Als	As	Jetzt	Now
Code	Asc	Null	Null
ArcTan	Atn	Zahl	Number
Attribute	Attributes	Objekt	Object
R	B	Okt	Oct
Basis	Base	Von	Of
Signal	Beep	Bei	On
RF	BF	Öffne	Open
Binär	Binary	Option	Option
Blau	Blue	Optional	Optional
Boolesch	Boolean	Oder	Or
Schaltflächen	Buttons	Ausgabe	Output
AlsZeiger	ByRef	ArgumentListe	ParamArray
AlsWert	ByVal	Pfad	Path
Rufe	Call	Pfadname	PathName
Fall	Case	Erhaltend	Preserve
ZuBoolesch	CBool	Drucke	Print
ZuWährung	CCur	Privat	Private
ZuDatumZeit	CDate	Eingabeaufforderung	Prompt
ZuDoppelt	CDbl	Meldung	Prompt
CDekl	CDecl	Eigenschaft	Property
Zeichen	Character	Öffentlich	Public
Zeichencode	CharCode	Lege	Put
WechsleVerz	ChDir	Wahlfrei	Random
WechsleLW	ChDrive	ZufallInit	Randomize
Zn	Chr	Lese	Read
ZuGanz	CInt	Rot	Red
Klasse	Class	Redim	ReDim
ZuLang	CLng	Kmt	Rem
Schliesse	Close	SchliesseAlles	Reset
Farbe	Color	Weiter	Resume
Vergleiche	Compare	Zurück	Return
Konst	Const	RückgabeTyp	ReturnType
Konstante	Constant	RGB	RGB

Tabelle 2: VBA-Schlüsselwörter Englisch – Deutsch (Forts.)

Deutsch	Englisch	Deutsch	Englisch
HilfeKontextID	Context	Rechts	Right
Konvertierung	Conversion	RechtsB	RightB
Cos	Cos	EntferneVerz	RmDir
ErstelleObjekt	CreateObject	Zufallszahl	Rnd
ZuEinfach	CSng	SetzeR	RSet
ZuZnF	CStr	KürzeR	RTrim
AktVerz	CurDir	Skript	Script
Währung	Currency	Sekunde	Second
ZuVariant	CVar	DPos	Seek
ZuVariantDatumZeit	CVDate	Prüfe	Select
ZuFehler	CVErr	SendeTastenF	SendKeys
DatumZeit	Date	Setze	Set
Datum	Date	SetzeAttr	SetAttr
Datumszahl	DateSerial	VZchn	Sgn
Datumswert	DateValue	Gemeinsam	Shared
Tag	Day	Starte	Shell
Test	Debug	Sin	Sin
Deklariere	Declare	Einfach	Single
Standard	Default	Quelle	Source
DefBoolesch	DefBool	LZn	Space
DefWährung	DefCur	PosLZn	Spc
DefDatumZeit	DefDate	Wurzel	Sqr
DefDoppelt	DefDbl	Statisch	Static
DefGanz	DefInt	Schrittweite	Step
DefLang	DefLng	Stop	Stop
DefObjekt	DefObj	ZnF	Str
DefEinfach	DefSng	VergleicheZnF	StrComp
DefZnF	DefStr	KonvertiereZnF	StrConv
DefVariant	DefVar	Strikt	Strict
Ziel	Destination	ZeichenF	String
Dim	Dim	Zeichenfolge	String
Verz	Dir	Sub	Sub
Durchlaufe	Do	PosTab	Tab
Ereignisse	DoEvents	Tan	Tan
Doppelt	Double	TextSpezial	Text

Tabelle 2: VBA-Schlüsselwörter Englisch – Deutsch (Forts.)

Deutsch	Englisch	Deutsch	Englisch
Laufwerk	Drive	Dann	Then
Alle	Each	Zeit	Time
Sonst	Else	Zeitgeber	Timer
SonstWenn	Elseif	Zeitzahl	TimeSerial
Leer	Empty	Zeitwert	TimeValue
Ende	End	Titel	Title
EndeWenn	EndIf	Bis	To
DEnde	EOF	Kürze	Trim
Äqv	Eqv	Wahr	True
Lösche	Erase	Typ	Type
FehlerZeile	Erl	TypName	TypeName
FehlerNr	Err	GrenzeO	UBound
Fehler	Error	Gross	UCase
Verlasse	Exit	Unbekannt	Unknown
Exp	Exp	Entsperre	Unlock
Explizit	Explicit	BisWahr	Until
Ausdruck	Expression	Wert	Val
F	F	Variant	Variant
Falsch	False	Variable	VarName
DAttr	FileAttr	VarTyp	VarType
DKopiere	FileCopy	vbAbbruch	vbAbort
DDatumZeit	FileDateTime	vbAbbruchWieder- holenIgnorieren	vbAbortRetryIgnore
DLänge	FileLen	vbAnwendungModal	vbApplicationModal
Dateinummer	FileNumber	vbArchiv	vbArchive
Schneide	Fix	vbDatenfeld	vbArray
Für	For	vbBoolesch	vbBoolean
Format	Format	vbAbbrechen	vbCancel
DNrNeu	FreeFile	vbKritisch	vbCritical
Funktion	Function	vbWährung	vbCurrency
Hole	Get	vbDatenObjekt	vbDataObject
HoleAttr	GetAttr	vbDatumZeit	vbDate
HoleObjekt	GetObject	vbStdSchaltfläche1	vbDefaultButton1
Global	Global	vbStdSchaltfläche2	vbDefaultButton2
RufeMarke	GoSub	vbStdSchaltfläche3	vbDefaultButton3

Tabelle 2: VBA-Schlüsselwörter Englisch – Deutsch (Forts.)

Deutsch	Englisch	Deutsch	Englisch
GeheZu	GoTo	vbVerzeichnis	vbDirectory
Grün	Green	vbDoppelt	vbDouble
Hilfedatei	HelpFile	vbLeer	vbEmpty
Hex	Hex	vbFehler	vbError
Stunde	Hour	vbWarnung	vbExclamation
Wenn	If	vbVersteckt	vbHidden
IMEStatus	IMEStatus	vbHiragana	vbHiragana
Imp	Imp	vbIgnorieren	vbIgnore
In	In	vbInformation	vbInformation
Eingabe	Input	vbGanz	vbInteger
EingabeB	InputB	vbKatakana	vbKatakana
EingabeDlg	InputDialog	vbLang	vbLong
InZnF	InStr	vbKleinschreibung	vbLowerCase
InZnFB	InStrB	vbSchmal	vbNarrow
RundeAb	Int	vbNein	vbNo
Ganz	Integer	vbNormal	vbNormal
Ist	Is	vbNull	vbNull
IstDatenfeld	IsArray	vbObjekt	vbObject
IstDatumZeit	IsDate	vbOK	vbOK
IstLeer	IsEmpty	vbOKAbbrechen	vbOKCancel
IstFehler	IsError	vbNurOK	vbOKOnly
IstFehlend	IsMissing	vbGroßschreibung- Wort	vbProperCase
IstNull	IsNull	vbFrage	vbQuestion
IstZahl	IsNumeric	vbNurLesen	vbReadOnly
IstObjekt	IsObject	vbWiederholen	vbRetry
DLösche	Kill	vbWiederholen- Abbrechen	vbRetryCancel
GrenzeU	LBound	vbEinfach	vbSingle
Klein	LCase	vbZnFolge	vbString
Links	Left	vbSystem	vbSystem
LinksB	LeftB	vbSystemModal	vbSystemModal
Länge	Len	vbGroßschreibung	vbUpperCase
LängeB	LenB	vbVariant	vbVariant
Länge	Length	vbDatenträger	vbVolume

Tabelle 2: VBA-Schlüsselwörter Englisch – Deutsch (Forts.)

Deutsch	Englisch	Deutsch	Englisch
Bestimme	Let	vbBreit	vbWide
Biblio	Lib	vbJa	vbYes
Wie	Like	vbJaNein	vbYesNo
Zeilen	Line	vbJaNeinAbbrechen	vbYesNoCancel
Pos	Loc	Wartezeit	Wait
Lokal	Local	Wochentag	WeekDay
Sperre	Lock	EndeSolange	Wend
DGrösse	LOF	Wobei	Where
Ln	Log	Solange	While
Lang	Long	Breite	Width
Schleife	Loop	Fensterart	WindowState
SetzeL	LSet	Mit	With
KürzeL	LTrim	Schreibe	Write
MacID	MacID	XOder	Xor
MacScript	MacScript	XPosition	XPos
TeilZnF	Mid	Jahr	Year
TeilZnFB	MidB	YPosition	YPos
Minute	Minute		

Tabelle 2: VBA-Schlüsselwörter Englisch – Deutsch (Forts.)

3 MsgBox-Konstanten

Eine MsgBox-Konstante zeigt eine Nachricht in einem Meldungsfenster an und wartet darauf, dass der Benutzer auf eine der Schaltflächen klickt, die darin enthalten sind. Es wird dann ein Wert vom Typ Integer zurückgegeben, der angibt, auf welche Schaltfläche der Benutzer geklickt hat.

Syntax

MsgBox(prompt[, buttons] [, title] [, helpfile, context])

Konstante	Wert	Beschreibung
vbOKOnly	0	Nur Schaltfläche OK (Voreinstellung)
vbOKCancel	1	Schaltflächen OK und Abbrechen
vbAbortRetryIgnore	2	Schaltflächen Abbruch, Wiederholen und Ignorieren
vbYesNoCancel	3	Schaltflächen Ja, Nein und Abbrechen
vbYesNo	4	Schaltflächen Ja und Nein
vbRetryCancel	5	Schaltflächen Wiederholen und Abbrechen
vbCritical	16	Meldung für kritischen Fehler
vbQuestion	32	Warnung mit Abfrage
vbExclamation	48	Warnmeldung

Tabelle 3: MsgBox-Argumente

Konstante	Wert	Beschreibung
vbInformation	64	Informationsmeldung
vbDefaultButton1	0	Erste Schaltfläche ist Voreinstellung (Voreinstellung)
vbDefaultButton2	256	Zweite Schaltfläche ist Voreinstellung
vbDefaultButton3	512	Dritte Schaltfläche ist Voreinstellung
vbDefaultButton4	768	Vierte Schaltfläche ist Voreinstellung
vbApplicationModal	0	An die Anwendung gebundenes Meldungsfeld (Voreinstellung)
vbSystemModal	4096	An das System gebundenes Meldungsfeld
vbMsgBoxHelpButton	16384	Fügt eine Hilfeschaltfläche zum Meldungsfeld hinzu.
VbMsgBoxSetForeground	65536	Legt das Meldungsfeld als Vordergrundfenster fest.
vbMsgBoxRight	524288	Der Text ist rechtsbündig
vbMsgBoxRtlReading	1048576	Legt fest, dass der Text von rechts nach links dargestellt wird, für hebräische und arabische Systeme.

Tabelle 3: MsgBox-Argumente (Forts.)

Konstante	Wert	Beschreibung
vbOK	1	Schaltfläche OK gedrückt
vbCancel	2	Schaltfläche Abbrechen gedrückt
vbAbort	3	Schaltfläche Abbruch gedrückt
vbRetry	4	Schaltfläche Wiederholen gedrückt
vbIgnore	5	Schaltfläche Ignorieren gedrückt
vbYes	6	Schaltfläche Ja gedrückt
vbNo	7	Schaltfläche Nein gedrückt

Tabelle 4: MsgBox-Rückgabewerte

Beispiel:

```
Sub Message()  
    MsgBox "Hallo Welt!", 524288, "Rechtsbündig"  
End Sub
```

4 Application.InputBox-Methode

Die Methode `Application.InputBox` öffnet ein Eingabefenster und erwartet eine Benutzereingabe. Zurückgegeben werden die Informationen, die in das Eingabefenster eingegeben wurden.

Wert	Bedeutung
0	Formel
1	Zahl

Tabelle 5: Application.InputBox-Methode

Wert	Bedeutung
2	Text (Zeichenfolge)
4	Logischer Wert (True oder False)
8	Zellbezug, z. B. ein Range-Objekt
16	Fehlerwert, z. B. #N/A
64	Wertematrix

Tabelle 5: Application.InputBox-Methode (Forts.)

Sie können die zulässigen Werte für das `Type`-Argument auch als Summe angeben. Damit in einem Eingabefeld sowohl Text als auch Zahlen eingegeben werden können, setzen Sie `Type` auf 3 (1 + 2). Da Zahlen den Datentyp `Integer` verwenden, Text hingegen einen `String` erfordern, muss die verwendete Variable vom Datentyp `Variant` deklariert werden. Dadurch kann die Variable `varCell` alle erforderlichen Datentypen annehmen.

Beispiele:

```
Sub MyInput()  
    Dim varCell As Variant  
  
    varCell = Application.InputBox( _  
        Prompt:="Markieren Sie einen Bereich", _  
        Type:=3)    ' Text und Zahl  
  
    MsgBox varCell  
End Sub  
  
Sub MyRange()  
    Dim rngArea As Range  
  
    Set rngArea = Application.InputBox( _  
        Prompt:="Markieren Sie einen Bereich", _  
        Type:=8)    ' Zellbezug  
  
    MsgBox rngArea.Address(0, 0)  
End Sub
```

5 Vergleichs-Operatoren

Syntax

```
Ergebnis = Ausdruck1 VergleichsOperator Ausdruck2  
Ergebnis = Objekt1 Is Objekt2  
Ergebnis = Zeichenfolge Like Muster
```

Operator	True, wenn	False, wenn	Null, wenn
< (Kleiner als)	Ausdruck1 < Ausdruck2	Ausdruck1 >= Ausdruck2	Ausdruck1 oder Ausdruck2 = Null
<= (Kleiner oder gleich)	Ausdruck1 <= Ausdruck2	Ausdruck1 > Ausdruck2	Ausdruck1 oder Ausdruck2 = Null
> (Größer als)	Ausdruck1 > Ausdruck2	Ausdruck1 <= Ausdruck2	Ausdruck1 oder Ausdruck2 = Null
>= (Größer oder gleich)	Ausdruck1 >= Ausdruck2	Ausdruck1 < Ausdruck2	Ausdruck1 oder Ausdruck2 = Null
= (Gleich)	Ausdruck1 = Ausdruck2	Ausdruck1 <> Ausdruck2	Ausdruck1 oder Ausdruck2 = Null
<> (Ungleich)	Ausdruck1 <> Ausdruck2	Ausdruck1 = Ausdruck2	Ausdruck1 oder Ausdruck2 = Null
< (Kleiner als)	Ausdruck1 < Ausdruck2	Ausdruck1 >= Ausdruck2	Ausdruck1 oder Ausdruck2 = Null

Tabelle 6: Vergleichs-Operatoren

Beispiel:

```
Sub SmallerThen()  
    ' Prüft, ob der erste Wert kleiner ist als der zweite Wert  
    Dim intA As Integer  
    Dim intB As Integer  
  
    intA = 300  
    intB = 500  
    If intA < intB Then  
        MsgBox "Der Wert " & intA & " ist kleiner als " & intB  
    Else  
        MsgBox "Der Wert " & intB & " ist ist kleiner" & intA  
    End If  
End Sub
```

6 Arithmetische Operatoren

Arithmetisch	Vergleich	Logisch
Potenzierung (^)	Gleich (=)	Not
Negation (-)	Ungleich (<>)	And
Multiplikation und Division (*, /)	Kleiner als (<)	Or
Ganzzahldivision (\)	Größer als (>)	Xor
Restwert (Mod)	Kleiner oder gleich (<=)	Eqv

Tabelle 7: Arithmetische Operatoren

Arithmetisch	Vergleich	Logisch
Addition und Subtraktion (+, -)	Größer oder gleich (>=)	Imp
Zeichenverkettung (&)		Like Is

Tabelle 7: Arithmetische Operatoren (Forts.)

Beispiel:

Der Operator / wird verwendet, um eine Fließkommadivision durchzuführen.

```
Sub Div()  
    Dim dblValue As Double  
  
    dblValue = 10 / 4      ' Liefert 2,5  
    dblValue = 10 / 3      ' Liefert 3,333333  
End Sub
```

Der Operator Eqv wird verwendet, um zwei Ausdrücke mit einer logischen Äquivalenzoperation (Gleichwertigkeit) zu verknüpfen.

```
Sub OperatorEqv()  
    Dim intA As Integer  
    Dim intB As Integer  
    Dim intC As Integer  
    Dim varD As Variant  
    Dim varTest As Variant  
  
    intA = 10  
    intB = 8  
    intC = 6  
    varD = Null  
  
    varTest = intA > intB Eqv intB > intC      ' Liefert True  
    varTest = intB > intA Eqv intB > intC      ' Liefert False  
    varTest = intA > intB Eqv intB > varD      ' Liefert Null  
    varTest = intA Eqv intB                    ' Liefert -3  
                                              ' (Bit-weiser Vergleich)  
End Sub
```

Der Operator Imp wird verwendet, um zwei Ausdrücke mit einer logischen Implikation (Beziehung) zu verknüpfen.

```
Sub OperatorImp()  
  
    Dim intA As Integer  
    Dim intB As Integer  
    Dim intC As Integer  
    Dim varD As Variant  
    Dim varTest As Variant  
  
    intA = 10
```

```

intB = 8
intC = 6
varD = Null

varTest = intA > intB Imp intB > intC      ' Liefert True
varTest = intA > intB Imp intC > intB      ' Liefert False
varTest = intB > intA Imp intC > intB      ' Liefert True
varTest = intB > intA Imp intC > varD      ' Liefert True
varTest = intC > varD Imp intB > intA      ' Liefert Null
varTest = intB Imp intA                    ' Liefert -1
                                           ' (Bit-weiser Vergleich)

```

End Sub

Der Operator `Mod` wird verwendet, um zwei Zahlen zu dividieren und den ganzzahligen Rest zurückzugeben. Fließkommazahlen werden vor der Operation auf einen jeweils ganzzahligen Wert gerundet.

```

Sub OperatorMod()
    ' Ganzzahligen Rest ausgeben
    Dim intResult As Integer

    intResult = 10 Mod 5      ' Liefert 0
    intResult = 10 Mod 3      ' Liefert 1
    intResult = 12 Mod 4.3    ' Liefert 0
    intResult = 12.6 Mod 5    ' Liefert 3

```

End Sub

Der Operator `Like` wird verwendet, um eine Zeichenfolge mit einem Muster zu vergleichen.

```

Sub OperatorLike()
    ' Vergleichen
    Dim strTest As String

    strTest = "aBBBa" Like "a*a"      ' Liefert True
    strTest = "F" Like "[A-Z]"        ' Liefert True
    strTest = "F" Like "[G-Z]"        ' Liefert False
    strTest = "a2a" Like "a#a"        ' Liefert True
    strTest = "ABC" Like "A?C"        ' Liefert True
    strTest = "ABC" Like "A?C?"       ' Liefert False

```

End Sub

7 Verkettungs-Operatoren

& (Operator)

+

Beispiel:

```

Sub Concatenate()
    ' Werte verketteten

```

```
Dim varTest As Variant

varTest = "Hallo" & " " & "Welt" ' Liefert: Hallo Welt (String)
varTest = "Hallo" + " " + "Welt" ' Liefert: Hallo Welt (String)
varTest = 123 & 456                ' Liefert: 123456      (String)
varTest = 123 + 456                ' Liefert: 579       (Integer)

End Sub
```

8 SpecialCells-Methode

Die SpecialCells-Methode gibt ein Range-Objekt zurück, das alle Zellen darstellt, die mit dem angegebenen Wert übereinstimmen.

Syntax

Ausdruck.SpecialCells(Type.Value)

Type

Konstante	Wert	Beschreibung
xlCellTypeAllFormatConditions	-4172	Zellen mit beliebigem Format
xlCellTypeAllValidation	-4174	Zellen mit Gültigkeitskriterien
xlCellTypeBlanks	4	Leere Zellen
xlCellTypeComments	-4144	Zellen mit Anmerkungen
xlCellTypeConstants	2	Zellen mit Konstanten
xlCellTypeFormulas	-4123	Zellen mit Formeln
xlCellTypeLastCell	11	Letzte Zelle im benutzten Bereich
xlCellTypeSameFormatConditions	-4173	Zellen mit gleichem Format
xlCellTypeSameValidation	-4175	Zellen mit gleichen Gültigkeitskriterien
xlCellTypeVisible	12	Alle sichtbaren Zellen

Tabelle 8: SpecialCells-Type-Konstanten

Beispiel:

```
Sub MySpecialCells()
    ' Letzte Zelle des benutzten Bereichs markieren
    ActiveSheet.UsedRange.SpecialCells(xlCellTypeLastCell).Activate
End Sub
```

Value

Value ist ein optionaler Variant-Wert. Falls für Type entweder xlCellTypeConstants oder xlCellTypeFormulas angegeben wird, legen Sie mit diesem Argument fest, welche Typen von Zellen im Ergebnis mit eingeschlossen werden. Diese Werte können addiert werden, wenn mehrere Typen zurückgegeben werden sollen. Standardmäßig werden, unabhängig vom Typ, alle Konstanten und Werte ausgewählt.

Konstante	Wert	Beschreibung
xlNumbers	1	Zahlen
xlTextValues	2	Text
xlLogical	4	Wahrheitswerte
xlErrors	16	Fehler

Tabelle 9: SpecialCells-Value-Konstanten

Die zulässigen Werte für das Value-Argument können auch als Summe angegeben werden. Damit beispielsweise in einem Bereich sowohl Zahlen- als auch Textwerte einbezogen werden, wird der Wert 3 angegeben (1 + 2). Ebenso ist eine Kombination der Konstanten möglich (xlNumbers + xlTextValues).

Beispiel:

```
Sub MySpecialCells()  
    ' Alle Zellen des benutzten Bereichs mit festen Zahlenwerten  
    ActiveSheet.UsedRange.SpecialCells _  
        (xlCellTypeConstants, xlNumbers).Activate  
  
    ' Alle Formelzellen des benutzten Bereichs mit Wahrheits-  
    ' oder Fehlerwerten  
    ActiveSheet.UsedRange.SpecialCells _  
        (xlCellTypeFormulas, 20).Activate  
End Sub
```

9 RGB-Funktion

RGB-Funktionen stellen RGB-Farbwerte dar.

Syntax

RGB(red, green, blue)

Farbe	Rot-Komponente	Grün-Komponente	Blau-Komponente
Schwarz	0	0	0
Blau	0	0	255
Grün	0	255	0
Cyan	0	255	255
Rot	255	0	0
Magenta	255	0	255
Gelb	255	255	0
Weiß	255	255	255

Tabelle 10: RGB-Funktion

Beispiel:

```
Sub RGBColor()  
    ' Zelle A1: Rote Hintergrundfarbe  
    Range("A1").Interior.Color = RGB(255, 0, 0)  
End Sub
```

10 ColorIndex-Eigenschaft

Mittels der ColorIndex-Eigenschaft können Sie einem Objekt eine Farbe zuweisen.

Objekt	ColorIndex
Border	Die Farbe des Rahmens
Borders	Die Farbe aller vier Rahmen. Gibt Null zurück, wenn nicht alle vier Farben gleich sind
Font	Die Farbe der Schriftart. Geben Sie xlColorIndexAutomatic an, um die Farbe automatisch zu verwenden
Interior	Die Füllfarbe des Innenraums. Weisen Sie dieser Eigenschaft xlColorIndexNone zu, um den Innenraum nicht zu füllen; weisen Sie xlColorIndexAutomatic zu, um diesen automatisch zu füllen (bei Zeichnungsobjekten)

Tabelle 11: ColorIndex-Eigenschaft

Beispiel:

```
Sub BackgroundColor()  
    ' Zelle A1: Rote Hintergrundfarbe  
    Range("A1").Interior.ColorIndex = 3  
End Sub
```

11 ColorFormat-Objekt (Fülleffekte)

Das ColorFormat-Objekt stellt die Farbe eines einfarbigen Objekts, die Vordergrund- bzw. Hintergrundfarbe eines Objekts mit Farbverlaufs- bzw. Musterfüllung oder die Farbe des Zeigers dar. Die Farben können durch Angabe eines expliziten Rot-Grün-Blau-Wertes (mit Hilfe der RGB-Eigenschaft) oder durch Auswahl einer Farbe der Farbskala (mit Hilfe der SchemeColor-Eigenschaft) festgelegt werden.

Eigenschaft	Objekt	Beschreibung
BackColor	FillFormat	Hintergrundfüllfarbe (bei schattierter oder gemusterter Füllung)
ForeColor	FillFormat	Vordergrundfüllfarbe (oder nur die Farbe einer einfarbigen Füllung)
BackColor	LineFormat	Hintergrundlinienfarbe (bei einer gemusterten Linie)

Tabelle 12: ColorFormat-Object

Eigenschaft	Objekt	Beschreibung
ForeColor	LineFormat	Vordergrundlinienfarbe (oder nur die Farbe einer einfarbigen Linie)
ForeColor	ShadowFormat	Schattenfarbe
ExtrusionColor	ThreeDFormat	Farbe der Seiten eines projizierten Objekts

Tabelle 12: ColorFormat-Object (Forts.)

Beispiele:

```
Sub Filleffects()  
    ' Ein Quadrat mit Fülleffekt erstellen (Fill)  
    With ActiveSheet.Shapes.AddShape( _  
        msoShapeRectangle, 90, 90, 90, 50).Fill  
        .ForeColor.RGB = RGB(128, 0, 0)  
        .BackColor.RGB = RGB(170, 170, 170)  
        .TwoColorGradient msoGradientHorizontal, 1  
    End With  
End Sub  
  
Sub CircleShadow()  
    ' Einen Kreis mit Schatten erstellen (TreeD & Extrusion)  
    Dim shpCircle As Shape  
  
    Set shpCircle = ActiveSheet.Shapes.AddShape( _  
        msoShapeOval, 90, 90, 90, 40)  
  
    With shpCircle.ThreeD  
        .Visible = True  
        .Depth = 50  
        .ExtrusionColor.RGB = RGB(255, 100, 255)  
    End With  
End Sub
```

12 FormatCondition-Objekt (Bedingte Formatierung)

Verwenden Sie `FormatConditions(Index)`, wobei `Index` die Indexnummer (1 bis 3) des bedingten Formats ist, um ein `FormatCondition`-Objekt zurückzugeben. Ein Bereich, kann bis zu *drei* bedingte Formate enthalten.

Objekt	Eigenschaften
Font	Bold Color ColorIndex FontStyle Italic Strikethrough Underline Die Unterstreichungsarten BUCHHALTUNG können nicht verwendet werden.

Tabelle 13: FormatCondition-Objekt

Objekt	Eigenschaften
Border	Bottom Color Left Right Style Top Weight Folgende Rahmenarten können verwendet werden (alle anderen werden nicht unterstützt): xlNone, xlSolid, xlDash, xlDot, xlDashDot, xlDashDotDot, xlGray50, xlGray75 und xlGray25. Folgende Rahmenstärken können verwendet werden (alle anderen werden nicht unterstützt): xlWeightHairline und xlWeightThin
Interior	Color ColorIndex Pattern PatternColorIndex

Tabelle 13: FormatCondition-Objekt (Forts.)

Beispiel:

```
Sub MyFormatConditions()  
    ' Dem Bereich A1:A10 wird eine Bedingung hinzugefügt  
    Dim rngArea As Range  
  
    Set rngArea = ActiveSheet.Range("A1:A10")  
  
    With rngArea  
        .FormatConditions.Delete  
        .FormatConditions.Add _  
            Type:=xlCellValue, _  
            Operator:=xlBetween, _  
            Formula1:="1", _  
            Formula2:="100"  
  
        With .FormatConditions(1)  
            .Font.ColorIndex = 3  
            .Interior.ColorIndex = 27  
        End With  
    End With  
End Sub
```

13 Formatcodes für Kopf- und Fußzeilen

Die speziellen Formatcodes aus der Tabelle 14 können in Kopf- und Fußzeilen verwendet werden.

Formatcode	Beschreibung
&L	Richtet nachfolgende Zeichen links aus
&C	Zentriert das nachfolgende Zeichen
&R	Richtet nachfolgende Zeichen rechts aus
&E	Schaltet Doppelt Unterstreichen ein oder aus
&X	Schaltet Hochstellen ein oder aus
&Y	Schaltet Tiefstellen ein oder aus
&B	Schaltet Fettdruck ein oder aus
&I	Schaltet Kursivdruck ein oder aus
&U	Schaltet Unterstreichen ein oder aus
&S	Schaltet Durchstreichen ein oder aus
&D	Druckt das aktuelle Datum
&T	Druckt die aktuelle Zeit
&F	Druckt den Namen des Dokuments
&A	Druckt den Namen des Registers einer Arbeitsmappe
&P	Druckt die Seitenzahl
&P+Zahl	Druckt die Seitenzahl zuzüglich der angegebenen Zahl
&&	Druckt ein einzelnes kaufmännisches Und-Zeichen (&)
&"Schriftart"	Druckt die nachfolgenden Zeichen in der angegebenen Schriftart. Schriftart muss von Anführungszeichen eingeschlossen sein
&nn	Druckt die nachfolgenden Zeichen im angegebenen Schriftgrad. Geben Sie eine zweistellige Zahl an, um den Schriftgrad anzugeben
&N	Druckt die Gesamtanzahl der Seiten eines Dokumentes

Tabelle 14: Formatcodes für Kopf- und Fußzeilen

Englisch	Deutsch
LeftHeader	Linke Kopfzeile
CenterHeader	Mittlere Kopfzeile
RightHeader	Rechte Kopfzeile
LeftFooter	Linke Fußzeile
CenterFooter	Mittlere Fußzeile
RightFooter	Rechte Fußzeile

Tabelle 15: Kopf- und Fußzeilen ansprechen

Beispiel:

Sub MyLeftFooter ()

' Linke Fußzeile Schriftgröße 8


```
ActiveSheet.PageSetup.LeftFooter = _  
    "&8Das Excel-VBA-Codebook Band 2"  
End Sub
```

14 Datentypen

Die folgende Übersicht enthält die von Visual Basic unterstützten Datentypen sowie deren Speicherbedarf und Wertebereiche.

Datentyp	Speicherbedarf	Wertebereich
Byte	1 Byte	0 bis 255
Boolean	Boolean	True oder False
Integer	2 Bytes	-32.768 bis 32.767
Long (lange Ganzzahl)	4 Bytes	-2.147.483.648 bis 2.147.483.647
Single (Gleitkommazahl mit einfacher Genauigkeit)	4 Bytes	-3,402823E38 bis -1,401298E-45 für negative Werte; 1,401298E-45 bis 3,402823E38 für positive Werte.
Double (Gleitkommazahl mit doppelter Genauigkeit)	8 Bytes	-1,79769313486231E308 bis -4,94065645841247E-324 für negative Werte; 4,94065645841247E-324 bis 1,79769313486232E308 für positive Werte
Currency (skalierte Ganzzahl)	8 Bytes	-922.337.203.685.477,5808 bis 922.337.203.685.477,5807
Decimal	14 Bytes	+/-79.228.162.514.264.337.593.543.950.335 ohne Dezimalzeichen; +/-7,9228162514264337593543950335 mit 28 Nachkommastellen; die kleinste Zahl ungleich Null ist +/-0,00000000000000000000000000000001
Date	8 Bytes	1. Januar 100 bis 31. Dezember 9999
Object	4 Bytes	Beliebiger Verweis auf ein Objekt vom Typ Object
String (variable Länge)	10 Bytes plus Zeichenfolgenlänge	0 bis ca. 2 Milliarden
String (feste Länge)	Zeichenfolgenlänge	1 bis ca. 65.400
1 bis ca. 65.400	16 Bytes	Numerische Werte im Bereich des Datentyps Double
Variant (mit Zeichen)	22 Bytes plus Zeichenfolgenlänge	Wie bei String mit variabler Länge

Tabelle 16: Datentypen

Datentyp	Speicherbedarf	Wertebereich
Benutzerdefiniert (mit Type)	Zahl ist von Elemen- ten abhängig	Der Bereich für jedes Element entspricht dem Bereich des zugehörigen Datentyps

Tabelle 16: Datentypen (Forts.)

15 Typ-Umwandlungsfunktionen

Jede Funktion legt für einen bestimmten Datentyp zwingend einen Ausdruck fest.

Syntax

CBool (Ausdruck)

Funktion	Rückgabety- p	Bereich des Arguments Ausdruck
CBool	Boolean	Eine gültige Zeichenfolge oder ein gültiger numerischer Ausdruck
CByte	Byte	0 bis 255
CCur	Currency	-922.337.203.685.477,5808 bis 922.337.203.685.477,5807
CDate	Date	Ein beliebiger gültiger Datumsausdruck
CDbl	Double	-1,79769313486231E308 bis -4,94065645841247E-324 für negative Werte; 4,94065645841247E-324 bis 1,79769313486232E308 für positive Werte
CDec	Decimal	+/-79.228.162.514.264.337.593.543.950.335 für skalierte Ganz- zahlen, d.h. Zahlen ohne Dezimalstellen. Für Zahlen mit 28 Dezi- malstellen gilt der Bereich +/-7,9228162514264337593543950335. Die kleinste mögliche Zahl ungleich Null ist 0,0000000000000000000000000000001
CInt	Integer	-32.768 bis 32.767; Nachkommastellen werden gerundet
CLng	Long	-2.147.483.648 bis 2.147.483.647; Nachkommastellen werden gerundet
CSng	Single	-3,402823E38 bis -1,401298E-45 für negative Werte; 1,401298E- 45 bis 3,402823E38 für positive Werte
CVar	Variant	Numerische Werte im Bereich des Typs Double. Nichtnumerische Werte im Bereich des Typs String
CStr	String	Rückgabe für CStr hängt vom Argument <i>Ausdruck</i> ab

Tabelle 17: Typ-Umwandlungsfunktionen

Beispiel:

```
Sub Convert()  
    ' Wandelt den Typ Currency in Double um  
    Dim curValue As Currency  
    Dim dblValue As Double
```

```
' curValue hat Typ Currency
curValue = CCur(234.456784)
' Ergebnis in Double umwandeln
dblValue = CDb1(curValue * 8.2 * 0.01)
End Sub
```

16 Benutzerdefinierte Datums- und Zeitformate (Format-Funktion)

Die folgende Übersicht enthält die Zeichen zum Erstellen von benutzerdefinierten Datums- und Zeitformaten.

Zeichen	Beschreibung
(:)	Zeit-Trennzeichen. Das gebräuchlichste Zeit-Trennzeichen ist der Doppelpunkt, aber in manchen Gebietsschemata werden andere Zeichen zum Trennen von Zeitangaben verwendet. Das Zeit-Trennzeichen trennt Stunden, Minuten und Sekunden beim Formatieren von Zeitangaben voneinander. Das tatsächliche Zeit-Trennzeichen in der formatierten Ausgabe hängt von den Systemeinstellungen ab
(/)	Datumstrennzeichen. In manchen Gebietsschemata können andere Zeichen zum Darstellen des Datums-Trennzeichens verwendet werden. Das Datums-Trennzeichen trennt Tag, Monat und Jahr beim Formatieren von Datumsangaben voneinander. Das tatsächliche Datums-Trennzeichen in der formatierten Ausgabe hängt von den Systemeinstellungen ab
C	Zeigt das Datum in der Form ddddd und die Zeit in der Form ttttt an, und zwar in dieser Reihenfolge. Hat die fortlaufende Zahl für das Datum keine Nachkommastellen, so wird nur das Datum angezeigt. Hat sie ausschließlich Nachkommastellen, so wird lediglich die Uhrzeit angezeigt
d	Zeigt den Tag als Zahl ohne führende Null (1 – 31) an
dd	Zeigt den Tag als Zahl mit führender Null (01 – 31)an
ddd	Zeigt den Tag als Abkürzung (So – Sa) an
dddd	Zeigt den Tag mit vollständigem Namen (Sonntag – Samstag) an
ddddd	Zeigt ein vollständiges Datum (einschließlich Tag, Monat und Jahr) an, das entsprechend der Systemeinstellung für das kurze Datumsformat formatiert ist. Die Voreinstellung für das kurze Datumsformat ist dd.mm.yy
dddddd	Zeigt ein vollständiges Datum (einschließlich Tag, Monat und Jahr) an, das entsprechend der Systemeinstellung für das lange Datumsformat formatiert ist. Die Voreinstellung für das lange Datumsformat ist dd.mmmm.yyyy
aaaa	Das gleiche wie dddd, allerdings die lokalisierte Version der Zeichenfolge
w	Zeigt den Wochentag als Zahl (1 für Sonntag bis 7 für Samstag) an
ww	Zeigt die Kalenderwoche als Zahl (1 – 54) an

Tabelle 18: Datums- und Zeitformate

Zeichen	Beschreibung
m	Zeigt den Monat als Zahl ohne führende Null (1 – 12) an. Wenn m unmittelbar auf h oder hh folgt, wird nicht der Monat, sondern die Minute angezeigt
mm	Zeigt den Monat als Zahl mit führender Null (01 – 12) an. Wenn m unmittelbar auf h oder hh folgt, wird nicht der Monat, sondern die Minute angezeigt
mmm	Zeigt den Monat als Abkürzung (Jan – Dez) an
mmmm	Zeigt den Monat mit vollständigem Namen (Januar – Dezember) an
oooo	Das gleiche wie mmmm, allerdings die lokalisierte Version der Zeichenfolge
q	Zeigt das Jahresquartal als Zahl (1 – 4) an
y	Zeigt den Kalendertag als Zahl (1 – 366) an
yy	Zeigt das Jahr als zweistellige Zahl (00 – 99) an
yyyy	Zeigt das Jahr als vierstellige Zahl (100 – 9999) an
h	Zeigt die Stunde als Zahl ohne führende Nullen (0 – 23) an
hh	Zeigt die Stunde als Zahl mit führender Nullen (00 – 23) an
n	Zeigt die Minute als Zahl ohne führende Nullen (0 – 59) an
nn	Zeigt die Minute als Zahl mit führender Nullen (00 – 59) an
s	Zeigt die Sekunde als Zahl ohne führende Nullen (0 – 59) an
ss	Zeigt die Sekunde als Zahl mit führender Nullen (00 – 59) an
t t t t t	Zeigt eine vollständige Zeitangabe (einschließlich Stunden, Minuten und Sekunden) an, die mit dem Zeit-Trennzeichen formatiert ist, das in der Systemeinstellung für das Zeitformat festgelegt ist. Eine führende Null wird angezeigt, wenn die entsprechende Option ausgewählt wurde und die Zeit vor 10:00 liegt. Die Voreinstellung für das Zeitformat ist h:mm:ss
AM/PM	12-Stunden-Format mit Anzeige von AM (groß geschrieben) vor 12 Uhr mittags und PM (großgeschrieben) ab 12 Uhr mittags und vor 24:00 Uhr
am/pm	12-Stunden-Format mit Anzeige von am (klein geschrieben) vor 12 Uhr mittags und pm (klein geschrieben) ab 12 Uhr mittags und vor 24:00 Uhr
A/P	12-Stunden-Format mit Anzeige von A (groß geschrieben) vor 12 Uhr mittags und P (großgeschrieben) ab 12 Uhr mittags und vor 24:00 Uhr
A/p	12-Stunden-Format mit Anzeige von a (klein geschrieben) vor 12 Uhr mittags und p (klein geschrieben) ab 12 Uhr mittags und vor 24:00 Uhr
AMPM	12-Stunden-Format mit Anzeige des AM-Zeichenfolgenliterals, das von Ihrem System festgelegt ist. Vor 12 Uhr mittags wird die AM-Zeichenfolge, und ab 12 Uhr mittags bis 24:00 Uhr die PM-Zeichenfolge angezeigt. AMPM kann klein oder groß geschrieben werden, die tatsächlich verwendete Schreibweise hängt von den Systemeinstellungen ab. Die Voreinstellung ist AM/PM

Tabelle 18: Datums- und Zeitformate (Forts.)

Beispiele dazu finden Sie in der Kategorie »Datum/Zeit«.

17 DateDiff-Funktion

Die DateDiff-Funktion gibt einen Wert vom Typ Long zurück, der die Anzahl der Zeitintervalle zwischen zwei bestimmten Terminen angibt.

Syntax

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

Die Syntax für die DateDiff-Funktion besteht aus folgenden benannten Bereichen:

Teil	Beschreibung
interval	Erforderlich. Zeichenfolgenausdruck, der das Zeitintervall ergibt, das Sie zur Berechnung der Differenz zwischen date1 und date2 verwenden
date1, date2	Erforderlich. Wert vom Typ Variant (Date). Zwei Termine, die Sie in Ihrer Berechnung verwenden möchten
Firstdayofweek	Optional. Eine Konstante, die den ersten Tag der Woche angibt. Ist dieser Wert nicht angegeben, so wird Sonntag angenommen
firstweekofyear	Optional. Eine Konstante, die die erste Woche des Jahres angibt. Ist dieser Wert nicht angegeben, wird die Woche mit dem 1. Januar als die erste Woche angenommen

Tabelle 19: DateDiff-Funktion

Das Argument interval hat die folgenden Einstellungen:

Einstellung	Beschreibung
yyyy	Jahr
q	Quartal
m	Monat
y	Tag des Jahres
d	Tag
w	Wochentag
ww	Woche
h	Stunde
n	Minute
s	Sekunde

Tabelle 20: Interval

Das Argument firstdayofweek hat folgende Einstellungen:

Konstante	Wert	Beschreibung
vbUseSystem	0	Die NLS API-Einstellung wird verwendet
vbSunday	1	Sonntag (Voreinstellung)

Tabelle 21: firstdayofweek

Konstante	Wert	Beschreibung
vbMonday	2	Montag
vbTuesday	3	Dienstag
vbWednesday	4	Mittwoch
vbThursday	5	Donnerstag
vbFriday	6	Freitag
vbSaturday	7	Samstag

Tabelle 21: firstdayofweek (Forts.)

Das Argument firstweekofyear hat folgende Einstellungen:

Konstante	Wert	Beschreibung
vbUseSystem	0	Die NLS API-Einstellung wird verwendet
vbFirstJan1	1	Anfang in der Woche mit dem 1. Januar (Voreinstellung)
vbFirstFourDays	2	Anfang in der ersten Woche, die mindestens vier Tage im neuen Jahr enthält
vbFirstFullWeek	3	Anfang in der ersten vollständigen Woche des Jahres

Tabelle 22: firstweekofyear

Beispiel:

```
Sub MyDate()  
    ' Datumsdifferenz berechnen  
    Dim datDate As Date  
    Dim intDay As Integer  
  
    datDate = InputBox("Geben Sie ein Datum ein")  
    intDay = DateDiff("d", Now, datDate)  
  
    MsgBox "Tagesdifferenz von heute an: " & intDay  
End Sub
```

18 OnKey-Methode

Führt die angegebene Prozedur aus, wenn eine bestimmte Taste oder Tastenkombination gedrückt wird.

Syntax

Ausdruck.OnKey(Key, Procedure)

TASTE	Code
RÜCKTASTE	{BACKSPACE} oder {BS}
PAUSE	{BREAK}

Tabelle 23: OnKey-Methode

TASTE	Code
FESTSTELLTASTE	{CAPSLOCK}
ENTF	{CLEAR}, {DELETE} oder {DEL}
NACH-UNTEN-TASTE	{DOWN}
ENDE	{END}
EINGABETASTE (Zehnertastatur)	{ENTER}
EINGABETASTE	~ (Tilde)
ESC	{ESCAPE} oder {ESC}
HILFE	{HELP}
POS1	{HOME}
EINFG	{INSERT}
NACH-LINKS-TASTE	{LEFT}
NUM	{NUMLOCK}
BILD-AB	{PGDN}
BILD-AUF	{PGUP}
EINGABETASTE	{RETURN}
NACH-RECHTS-TASTE	{RIGHT}
ROLLEN	{SCROLLOCK}
TAB	{TAB}
NACH-OBEN-TASTE	{UP}
F1 bis F15	{F1} bis {F15}

Tabelle 23: OnKey-Methode (Forts.)

Es kann jede beliebige Tastenkombination mit ,  und  angegeben werden. Eine Taste kann mit einer oder mehreren anderen Tasten entsprechend der Tabelle 23 kombiniert werden.

Kombinieren mit	Vorangestelltes Zeichen
UMSCHALT	+ (Pluszeichen)
STRG	^ (Caret-Zeichen)
ALT	% (Prozentzeichen)

Tabelle 24: Kombinationsmöglichkeiten

Beispiel:

```
Sub OnKeyZ()  
    ' Tastenkombination: Strg + z  
    Application.OnKey "^z", "MyMacro"  
End Sub  
  
Sub ResetOnKeyZ()
```

```
' Tastenkombination zurücksetzen
Application.OnKey "^z"
End Sub
```

19 Attributes-Eigenschaft

Die Attributes-Eigenschaft legt die Attribute von Dateien oder Ordnern fest, oder gibt diese zurück. Je nach Attribut kann aus den Dateien/Ordnern gelesen und in sie geschrieben werden bzw. sind sie schreibgeschützt.

Syntax

```
Objekt.Attributes [= neueAttribute]
```

Konstante	Wert	Beschreibung
Normal	0	Normale Datei. Es wurden keine Attribute festgelegt
ReadOnly	1	Schreibgeschützte Datei. Attribut mit Lese-/Schreibzugriff
Hidden	2	Verborgene Datei. Attribut mit Lese-/Schreibzugriff
System	4	Systemdatei. Attribut mit Lese-/Schreibzugriff
Volume	8	Datenträgerbezeichnung des Laufwerks. Attribut mit Schreibschutz
Directory	16	Ordner oder Verzeichnis. Das Attribut ist schreibgeschützt
Archive	32	Die Datei hat sich seit der letzten Sicherung geändert. Attribut mit Lese-/Schreibzugriff
Alias	64	Verknüpfung. Attribut mit Schreibschutz
Compressed	128	Komprimierte Datei. Attribut mit Schreibschutz

Tabelle 25: Attributes-Eigenschaft

20 Verschiedene Konstanten

Konstante	Äquivalent	Beschreibung
vbCrLf	Chr(13) + Chr(10)	Kombination aus Wagenrücklauf und Zeilenvorschub
vbCr	Chr(13)	Wagenrücklaufzeichen
vbLf	Chr(10)	Zeilenvorschub
vbNewLine	Chr(13) + Chr(10) oder, auf dem Macintosh, Chr(13)	Plattformspezifisches Zeilenumbruchzeichen; je nachdem, welches für die aktuelle Plattform geeignet ist
vbNullChar	Chr(0)	Zeichen mit dem Wert 0
vbNullString	Zeichenfolge mit dem Wert 0	Nicht identisch mit der Null-Zeichenfolge (»«); wird verwendet, um externe Prozeduren aufzurufen

Tabelle 26: Verschiedene Konstanten

Konstante	Äquivalent	Beschreibung
vbObjectError	- 2147221504	Benutzerdefinierte Fehlernummern sollten größer als dieser Wert sein. Zum Beispiel: Err.Raise Number = vbObjectError + 1000
vbTab	Chr(9)	Tabulatorzeichen
vbBack	Chr(8)	Rückschrittzeichen
vbFormFeed	Chr(12)	Nicht sinnvoll unter Microsoft Windows oder auf dem Macintosh
vbVerticalTab	Chr(11)	Nicht sinnvoll unter Microsoft Windows oder auf dem Macintosh

Tabelle 26: Verschiedene Konstanten (Forts.)

Beispiel:

```
Sub MyConst()  
    MsgBox "Hallo" & vbLf & "Welt"  
End Sub
```

21 ClipboardFormats-Eigenschaft

Die ClipboardFormats-Eigenschaft gibt die Formate, die zurzeit in der Zwischenablage enthalten sind, als eine Matrix numerischer Werte zurück.

Syntax

Ausdruck.ClipboardFormats(Index)

Diese Eigenschaft gibt eine Matrix numerischer Werte zurück. Um festzustellen, ob ein bestimmtes Format vorhanden ist, vergleichen Sie jedes Element der Matrix mit einer der folgenden xlClipboardFormat-Konstanten.

ClipboardFormats-Eigenschaften	
xlClipboardFormatBIFF	xlClipboardFormatObjectDesc
xlClipboardFormatBIFF2	xlClipboardFormatObjectLink
xlClipboardFormatBIFF3	xlClipboardFormatOwnerLink
xlClipboardFormatBIFF4	xlClipboardFormatPICT
xlClipboardFormatBinary	xlClipboardFormatPrintPICT
xlClipboardFormatBitmap	xlClipboardFormatRTF
xlClipboardFormatCGM	xlClipboardFormatScreenPICT
xlClipboardFormatCSV	xlClipboardFormatStandardFont
xlClipboardFormatDIF	xlClipboardFormatStandardScale
xlClipboardFormatDspText	xlClipboardFormatSYLK
xlClipboardFormatEmbeddedObject	xlClipboardFormatTable
xlClipboardFormatEmbedSource	xlClipboardFormatText

Tabelle 27: ClipboardFormats-Eigenschaften

ClipboardFormats-Eigenschaften	
xlClipboardFormatLink	xlClipboardFormatToolFace
xlClipboardFormatLinkSource	xlClipboardFormatToolFacePICT
xlClipboardFormatLinkSourceDesc	xlClipboardFormatVALU
xlClipboardFormatMovie	xlClipboardFormatWK1
xlClipboardFormatNative	

Tabelle 27: ClipboardFormats-Eigenschaften (Forts.)

Beispiel:

```
Sub MyClipboard()  
    ' Prüfung, ob sich in der Zwischenablage  
    ' ein Objekt im RTF-Format befindet  
    Dim varCBF As Variant  
    Dim varFmt As Variant  
  
    varCBF = Application.ClipboardFormats  
  
    For Each varFmt In varCBF  
        If varFmt = xlClipboardFormatRTF Then  
            MsgBox "Die Zwischenablage enthält Rich-Text-Format"  
        End If  
    Next  
End Sub
```

22 Spezifikationen und Einschränkungen von Excel

Die nachfolgenden Spezifikationen und Einschränkungen können je nach Version leicht abweichen. Mehr dazu finden Sie in der Online-Hilfe, unter dem Stichwort »Spezifikationen«.

Spezifikationen für Arbeitsblätter und Arbeitsmappen

Feature	Maximaler Wert
Geöffnete Arbeitsmappen	Durch den verfügbaren Speicher und die Systemressourcen begrenzt
Arbeitsblattgröße	65.536 Zeilen mal 256 Spalten
Spaltenbreite	255 Zeichen
Zeilenhöhe	409 Punkte
Seitewechsel	1000 horizontal und vertikal
Länge des Inhalts der Zelle (Text)	32.767 Zeichen. Nur 1.024 Zeichen werden in einer Zelle angezeigt; alle 32.767 werden in der Formelleiste angezeigt.
Blätter in einer Arbeitsmappe	Durch den verfügbaren Speicher begrenzt (Standardwert 3 Blätter)
Farben pro Arbeitsmappe	56

Tabelle 28: Spezifikationen für Arbeitsblätter und Arbeitsmappen

Feature	Maximaler Wert
Zellformate pro Arbeitsmappe	4.000
Benannte Ansichten in einer Arbeitsmappe	Durch den verfügbaren Speicher begrenzt
Benutzerdefinierte Zahlenformate	Durch den verfügbaren Speicher begrenzt
Namen pro Arbeitsmappe	Durch den verfügbaren Speicher begrenzt
Fenster pro Arbeitsmappe	Durch die Systemressourcen begrenzt
Ausschnitte pro Fenster	4
Verknüpfte Blätter	Durch den verfügbaren Speicher begrenzt
Szenarios	Durch den verfügbaren Speicher begrenzt; in einem Übersichtsbericht werden nur die ersten 251 Szenarios angezeigt
Sich ändernde Zellen pro Szenario	32
Anpassbare Zellen in Solver	200
Benutzerdefinierte Funktionen	Durch den verfügbaren Speicher begrenzt
Zoom-Bereich	10 bis 400 Prozent
Berichte	Durch den verfügbaren Speicher begrenzt
Sortierbezüge	3 in einem einfachen Sortiervorgang; bei sequenziellen Sortiervorgängen unbegrenzt
Rückgängig-Stufen	16
Felder pro Datenformular	32
Benutzerdefinierte Symbolleisten pro Arbeitsmappe	Durch den verfügbaren Speicher begrenzt
Benutzerdefinierte Schaltflächen der Symbolleiste	Durch den verfügbaren Speicher begrenzt

Tabelle 28: Spezifikationen für Arbeitsblätter und Arbeitsmappen (Forts.)

Spezifikationen für Arbeitsgruppen

Feature	Maximaler Wert
Benutzer, die gleichzeitig eine freigegebene Arbeitsmappe öffnen und gemeinsam verwenden können	256
Persönliche Ansichten in einer freigegebenen Arbeitsmappe	Durch den verfügbaren Speicher begrenzt
Tage, für die ein Änderungsprotokoll geführt wird	32.767 (Standard: 30 Tage)
Arbeitsmappen, die gleichzeitig zusammengeführt werden können	Durch den verfügbaren Speicher begrenzt

Tabelle 29: Spezifikationen für Arbeitsgruppen

Feature	Maximaler Wert
Zellen, die in einer freigegebenen Arbeitsmappe markiert werden können	32.767
Farben, die zur Kennzeichnung der von verschiedenen Benutzern durchgeführten Änderungen verwendet werden, wenn ÄNDERUNGEN HERVORHEBEN aktiviert ist	32 (jeder Benutzer wird durch eine individuelle Farbe identifiziert; Änderungen des aktuellen Anwenders werden in Marineblau hervorgehoben)

Tabelle 29: Spezifikationen für Arbeitsgruppen (Forts.)

Berechnungs-Spezifikationen

Feature	Maximaler Wert
Genauigkeit bei Zahlen	15 Stellen
Größte Zahl, die in eine Zelle eingegeben werden kann	9,999999999999999E307
Größte zulässige positive Zahl	1,79769313486231E308
Kleinste zulässige negative Zahl	2,2250738585072E-308
Kleinste zulässige positive Zahl	2,229E-308
Größte zulässige negative Zahl	-1E-307
Länge des Formelinhalts	1.024 Zeichen
Iterationen	32.767
Arbeitsmappenmatrizen	Durch den verfügbaren Speicher begrenzt. Außerdem können sich Matrizen nicht auf vollständige Spalten beziehen. Beispielsweise kann sich eine Matrix nicht auf die vollständige Spalte C:C oder auf den Bereich C1:C65536 beziehen. Eine Matrix kann sich hingegen auf den Bereich C1:D65535 beziehen, da der Bereich eine Zeile kleiner als die maximale Größe des Arbeitsblattes ist und nicht die vollständige Spalte C oder D einschließt.
Ausgewählte Bereiche	2.048
Argumente in einer Funktion	30
Verschachtelte Funktionsebenen	7
Anzahl von verfügbaren Arbeitsblattfunktionen	329
Frühestes zulässiges Datum bei Berechnungen	1. Januar 1900 (1. Januar 1904, falls das 1904-Datumssystem verwendet wird)
Spätestes zulässiges Datum bei Berechnungen	31. Dezember 9999

Tabelle 30: Berechnungs-Spezifikationen

Feature	Maximaler Wert
Größte Uhrzeit, die eingegeben werden kann	9999:99:99

Tabelle 30: Berechnungs-Spezifikationen (Forts.)

Pivottabellen-Berichtsspezifikationen

Feature	Maximaler Wert
PivotTable-Berichte pro Blatt	Durch den verfügbaren Speicher begrenzt
Objekte pro PivotTable-Bericht	8.000
Zeilen- oder Spaltenfelder in einem PivotTable-Bericht	Durch den verfügbaren Speicher begrenzt
Seitenfelder pro PivotTable-Bericht	256 (u.U. durch den verfügbaren Speicher begrenzt)
Datenfelder pro PivotTable-Bericht	256
Berechnete Elementformeln pro PivotTable-Bericht	Durch den verfügbaren Speicher begrenzt

Tabelle 31: Pivottabellen-Berichtsspezifikationen

Spezifikationen für Diagramme

Feature	Maximaler Wert
Diagramme, die mit einer Tabelle verknüpft sind	Durch den verfügbaren Speicher begrenzt
Arbeitsblätter, auf die sich ein Diagramm bezieht	255
Datenreihen pro Diagramm	255
Datenpunkte in einer Datenreihe für 2-D-Diagramme	32.000
Datenpunkte in einer Datenreihe für 3-D-Diagramme	4.000
Datenpunkte für alle Datenreihen in einem Diagramm	256.000
Linienarten	2,40
Linienstärken	4
Flächenmuster (Bildschirm-anzeige)	18
Gesamtanzahl von Muster- und Farbkombinationen (Farbanzeige)	56.448

Tabelle 32: Spezifikationen für Diagramme

Feature	Maximaler Wert
Flächenmuster- und Farbkombinationen (Farbanzeige)	56.448 (die tatsächliche Anzahl hängt vom Drucker und der zugehörigen Software ab)
Seitenfelder in einem PivotChart-Bericht	256 (u.U. durch den verfügbaren Speicher begrenzt)
Datenfelder in einem PivotChart-Bericht	256
Berechnete Formelobjekte in einem PivotChart-Bericht	Durch den verfügbaren Speicher begrenzt

Tabelle 32: Spezifikationen für Diagramme (Forts.)

23 Liste der integrierten Dialogfeldargumente

Per VBA können Excel-eigene Dialogfenster aufgerufen werden. Die nachfolgende Übersicht beschreibt die gängigsten dieser integrierten Dialogfenster. Eine komplette Liste befindet sich in der Online-Hilfe unter dem Stichwort »Liste der integrierten Dialogfeldargumente«.

Hinweis

Bei einigen dieser Konstanten wird kein Dialogfenster angezeigt, sondern die Aktion wird direkt ausgeführt, wie z.B. beim Autofilter `xlDialogFilter`. Erste Voraussetzung ist, dass ein zu filternder Datenbereich vorhanden ist. Zweitens muss sich der Cursor in diesem Bereich befinden, wenn die Prozedur ausgeführt wird. In diesem bestimmten Fall wird wider erwarten nicht ein Dialogfenster angezeigt, sondern die Filter-Pfeile werden direkt dem Datenbereich zugewiesen. Bei erneutem Ausführen dieser Prozedur werden die Pfeile wieder entfernt. Das ganze mag zwar erst verwirrend sein, da hier ja von Dialogfenstern die Rede ist. Im Grunde genommen ist es jedoch logisch, denn beim aktivieren des Autofilters über den Menüpunkt DATEN | FILTER | AUTOFILTER wird ebenfalls kein Dialogfenster angezeigt.

Insgesamt stehen Excel-intern weit über 700 Dialogfenster zur Verfügung. Der Objektkatalog umfasst jedoch lediglich etwas über 250 Stück. Die genaue Zahl ist von der eingesetzten Excel-Version abhängig. Excel unterstützt nicht alle dieser Dialogfeldargumente, wie zum Beispiel die Konstante `xlDialogSaveCopyAs`. Das bedeutet, dass diese Argumente nicht verwendet werden können.

Viele der Dialogfelder beziehen sich auf ein bestimmtes Objekt. Dieses Objekt muss in der Mappe vorhanden und teilweise selektiert sein, da VBA sonst eine Fehlermeldung anzeigt (Debugger). Eine typische Fehlermeldung für so einen Fall ist der »Laufzeitfehler '1004'«.

Im folgenden Beispielcode wird das Dialogfenster `SPEICHERN UNTER` geöffnet. Als Dateiname (Argument) wird der Inhalt der Zelle A1 vorgeschlagen.

Beispiel:

```
Sub MySaveAs()  
    Application.Dialogs(xlDialogSaveAs).Show Range("A1").Value  
End Sub
```

Dialogfeldkonstante	Index	Argumentenliste(n)	Deutscher Dialogname
xlDialogActivate	103	window_text, pane_num	Aktivieren
xlDialogActiveCellFont	476	font, font_style, size, strike-through, superscript, subscript, outline, shadow, underline, color, normal, background, start_char, char_count	Zellen formatieren (Schrift)
xlDialogAddinManager	321	operation_num, addinname_text, copy_logical	Add-Ins
xlDialogAlignment	43	horiz_align, wrap, vert_align, orientation, add_indent	Zellen formatieren (Ausrichtung)
xlDialogApplyNames	133	name_array, ignore, use_rowcol, omit_col, omit_row, order_num, append_last	Namen übernehmen (benannte Bereiche müssen vorhanden sein)
xlDialogApplyStyle	212	style_text	Formatvorlage
xlDialogAppMove	170	x_num, y_num	Fenster verschieben (im Vollfenstermodus nicht möglich. Es wird keine Dialogbox angezeigt)
xlDialogAppSize	171	x_num, y_num	Fenstergröße (im Vollfenstermodus nicht möglich. Es wird keine Dialogbox angezeigt)
xlDialogArrangeAll	12	arrange_num, active_doc, sync_horiz, sync_vert	Fenster anordnen
xlDialogAssignToObject	213	macro_ref	Makro zuweisen (ein Objekt muss aktiv bzw. selektiert sein)
xlDialogAttachToolbars	323		Symbolleiste anfügen
xlDialogAutoCorrect	485	correct_initial_caps, capitalize_days	Autokorrektur
xlDialogBorder	45	outline, left, right, top, bottom, shade, outline_color, left_color, right_color, top_color, bottom_color	Zellen formatieren (Rahmen)

Tabelle 33: Integrierte Dialogfeldargumente

Dialogfeldkonstante	Index	Argumentenliste(n)	Deutscher Dialogname
xlDialogCalculation	32	type_num, iter, max_num, max_change, update, precision, date_1904, calc_save, save_values, alt_exp, alt_form	Berechnungsoptionen
xlDialogCellProtection	46	locked, hidden	Zellen formatieren (Schutz)
xlDialogClear	52	type_num	Inhalte löschen
xlDialogColorPalette	161	file_text	Farbpalette
xlDialogColumnWidth	47	width_num, reference, standard, type_num, standard_num	Spaltenbreite
xlDialogConditionalFormatting	583		Bedingte Formatierung
xlDialogConsolidate	191	source_refs, function_num, top_row, left_col, create_links	Konsolidieren
xlDialogCopyPicture	108	appearance_num, size_num, type_num	Bild kopieren
xlDialogCreateNames	62	top, left, bottom, right	Namen erstellen
xlDialogCustomizeToolbar	276	category	Anpassen
xlDialogCustomViews	493		Benutzerdefinierte Ansichten
xlDialogDataValidation	525		Gültigkeitsprüfung
xlDialogDefineName	61	name_text, refers_to, macro_type, shortcut_text, hidden, category, local	Namen definieren
xlDialogDefineStyle	229	style_text, number, font, alignment, border, pattern, protection	Formatvorlage
xlDialogDeleteFormat	111	format_text	Zellen formatieren
xlDialogDeleteName	110	name_text	Name definieren
xlDialogDemote	203	row_col	Gruppierung
xlDialogDisplay	27	formulas, gridlines, headings, zeros, color_num, reserved, outline, page_breaks, object_num	Bildschirmanzeigeoptionen
xlDialogEditDelete	54	shift_num	Löschen (Zeilen, Spalten, Zellen)
xlDialogFileDelete	6	file_text	Löschen (Dateien)

Tabelle 33: Integrierte Dialogfeldargumente (Forts.)

Dialogfeldkonstante	Index	Argumentenliste(n)	Deutscher Dialogname
xlDialogFileSharing	481		Arbeitsmappe freigeben
xlDialogFilter	447		Datenfilter ein/aus (es wird keine Dialogbox angezeigt)
xlDialogFilterAdvanced	370	operation, list_ref, criteria_ref, copy_ref, unique	Spezialfilter
xlDialogFindFile	475		Öffnen (Datei)
xlDialogFont	26	name_text, size_num	Schriftarten
xlDialogFontProperties	381	font, font_style, size, strike-through, superscript, subscript, outline, shadow, underline, color, normal, background, start_char, char_count	Zellen formatieren
xlDialogFormatAuto	269	format_num, number, font, alignment, border, pattern, width	AutoFormat
xlDialogFormatNumber	42	format_text	Zellen formatieren
xlDialogFormatText	89	x_align, y_align, orient_num, auto_text, auto_size, show_key, show_value, add_indent	Textfeld formatieren
xlDialogFormulaFind	64	text, in_num, at_num, by_num, dir_num, match_case, match_byte	Suchen
xlDialogFormulaGoto	63	reference, corner	Gehe zu
xlDialogFormulaReplace	130	find_text, replace_text, look_at, look_by, active_cell, match_case, match_byte	Ersetzen
xlDialogFunctionWizard	450		Funktion einfügen (Funktionsassistent)
xlDialogGoalSeek	198	target_cell, target_value, variable_cell	Zielwertsuche
xlDialogImportTextFile	666		Textdatei importieren
xlDialogInsert	55	shift_num	Zellen einfügen
xlDialogInsertHyperlink	596		Hyperlink einfügen
xlDialogInsertNameLabel	496		Beschriftungsbereiche

Tabelle 33: Integrierte Dialogfeldargumente (Forts.)

Dialogfeldkonstante	Index	Argumentenliste(n)	Deutscher Dialogname
xlDialogInsertObject	259	object_class, file_name, link_logical, display_icon_logical, icon_file, icon_number, icon_label	Objekt einfügen
xlDialogInsertPicture	342	file_name, filter_number	Grafik einfügen
xlDialogListboxProperties	437	range, link, drop_size, multi_select, 3d_shading	Steuerelement formatieren (Formular Listenfeld)
xlDialogNew	119	type_num, xy_series, add_logical	Aufgabenbereich einblenden (keine Dialogbox)
xlDialogNewWebQuery	667		Neue Webabfrage
xlDialogNote	154	add_text, cell_ref, start_char, num_chars	Notiz einfügen (es wird kein Dialogfenster angezeigt)
xlDialogOpen	1	file_text, update_links, read_only, format, prot_pwd, write_res_pwd, ignore_rorec, file_origin, custom_delimit, add_logical, editable, file_access, notify_logical, converter	Öffnen
xlDialogOptionsCalculation	318	type_num, iter, max_num, max_change, update, precision, date_1904, calc_save, save_values	Optionen (Berechnung)
xlDialogOptionsEdit	319	incell_edit, drag_drop, alert, entermove, fixed, decimals, copy_objects, update_links, move_direction, autocomplete, animations	Optionen (Bearbeiten)
xlDialogOptionsGeneral	356	R1C1_mode, dde_on, sum_info, tips, recent_files, old_menus, user_info, font_name, font_size, default_location, alternate_location, sheet_num, enable_under	Optionen (Allgemein)
xlDialogOptionsListsAdd	458	string_array, import_ref, by_row	Optionen (Benutzerdefinierte Listen)

Tabelle 33: Integrierte Dialogfeldargumente (Forts.)

Dialogfeldkonstante	Index	Argumentenliste(n)	Deutscher Dialogname
xlDialogOptionsME	647	def_rtl_sheet, crsr_mvmt, show_ctrl_char, gui_lang	Optionen (International)
xlDialogOptionsTransition	355	menu_key, menu_key_action, nav_keys, trans_eval, trans_entry	Optionen (Umsteigen)
xlDialogOptionsView	320	formula, status, notes, show_info, object_num, page_breaks, formulas, gridlines, color_num, headers, outline, zeros, hor_scroll, vert_scroll, sheet_tabs	Optionen (Ansicht)
xlDialogPageSetup	7	head, foot, left, right, top, bot, hdng, grid, h_cntr, v_cntr, orient, paper_size, scale, pg_num, pg_order, bw_cells, quality, head_margin, foot_margin, notes, draft	Seite einrichten
xlDialogParse	91	parse_text, destination_ref	Analyse
xlDialogPasteSpecial	53	paste_num, operation_num, skip_blanks, transpose	Inhalte einfügen (funktioniert nur im Kopiermodus)
xlDialogPatterns	84	apattern, afore, aback, newui	Zellen formatieren (Muster)
xlDialogPrint	8	range_num, from, to, copies, draft, preview, print_what, color, feed, quality, y_resolution, selection, printer_text, print_to_file, collate	Drucken
xlDialogPrinterSetup	9	printer_text	Druckereinrichtung
xlDialogPrintPreview	222		Seitenansicht
xlDialogProperties	474	title, subject, author, keywords, comments	Eigenschaften-Fenster
xlDialogProtectDocument	28	contents, windows, password, objects, scenarios	Blatt schützen
xlDialogProtectSharing	620		Freigegebenen Arbeitsmappe schützen

Tabelle 33: Integrierte Dialogfeldargumente (Forts.)

Dialogfeldkonstante	Index	Argumentenliste(n)	Deutscher Dialogname
xlDialogPublishAsWebPage	653		Als Webseite ver- öffentlichen
xlDialogReplaceFont	134	font_num, name_text, size_num, bold, italic, under- line, strike, color, outline, shadow	Schriftarten
xlDialogRoutingSlip	336	recipients, subject, message, route_num, return_logical, status_logical	Microsoft Outlook
xlDialogRowHeight	127	height_num, reference, standard_height, type_num	Zeilenhöhe
xlDialogRun	17	reference, step	Makro ausführen
xlDialogSaveAs	5	document_text, type_num, prot_pwd, backup, write_res_pwd, read_only_rec	Speichern unter
xlDialogSaveWorkbook	145	document_text, type_num, prot_pwd, backup, write_res_pwd, read_only_rec	Speichern unter
xlDialogSaveWorkspace	285	name_text	Arbeitsbereich speichern
xlDialogSelectSpecial	132	type_num, value_type, levels	Inhalte auswählen
xlDialogSendMail	189	recipients, subject, return_receipt	Mappe als E-Mail versenden
xlDialogSetBackground- Picture	509		Hintergrundbild
xlDialogSetPrintTitles	23	titles_for_cols_ref, titles_for_rows_ref	Drucktitel festlegen
xlDialogShowToolbar	220	bar_id, visible, dock, x_pos, y_pos, width, protect, tool_tips, large_buttons, color_buttons	Anpassen
xlDialogSort	39	orientation, key1, order1, key2, order2, key3, order3, header, custom, case	Sortieren
xlDialogSortSpecial	192	sort_by, method, key1, order1, key2, order2, key3, order3, header, order, case	Sortieren
xlDialogSplit	137	col_split, row_split	Fenster teilen (kein Dialogfenster)

Tabelle 33: Integrierte Dialogfeldargumente (Forts.)

Dialogfeldkonstante	Index	Argumentenliste(n)	Deutscher Dialogname
xlDialogStandardFont	190	name_text, size_num, bold, italic, underline, strike, color, outline, shadow	Schriftarten
xlDialogStandardWidth	472	standard_num	Standardbreite
xlDialogStyle	44	bold, italic	Schriftarten
xlDialogSubtotalCreate	398	at_change_in, function_num, total, replace, pagebreaks, summary_below	Teilergebnisse
xlDialogSummaryInfo	474	title, subject, author, keywords, comments	Zusammenfassung
xlDialogTable	41	row_ref, column_ref	Tabelle
xlDialogTextToColumns	422	destination_ref, data_type, text_delim, consecutive_delim, tab, semicolon, comma, space, other, other_char, field_info	Textkonvertierungs-Assistent
xlDialogUnhide	94	window_text	Einblenden (Mappe)
xlDialogWebOptionsEncoding	686		Weboptionen (Codierung)
xlDialogWebOptionsFiles	684		Weboptionen (Dateien)
xlDialogWebOptionsFonts	687		Weboptionen (Schriftarten)
xlDialogWebOptionsGeneral	683		Weboptionen (Allgemein)
xlDialogWebOptionsPictures	685		Weboptionen (Bilder)
xlDialogWindowMove	14	x_pos, y_pos, window_text	Fenster verschieben
xlDialogWindowSize	13	width, height, window_text	Fenstergröße
xlDialogWorkbookAdd	281	name_array, dest_book, position_num	Verschieben oder kopieren (Tabellenblatt)
xlDialogWorkbookCopy	283	name_array, dest_book, position_num	Verschieben oder kopieren (Tabellenblatt)
xlDialogWorkbookInsert	354	type_num	Einfügen (Tabellenblatt etc.)

Tabelle 33: Integrierte Dialogfeldargumente (Forts.)

Dialogfeldkonstante	Index	Argumentenliste(n)	Deutscher Dialogname
xlDialogWorkbookMove	282	name_array, dest_book, position_num	Verschieben oder kopieren (Tabellenblatt)
xlDialogWorkbookName	386	oldname_text, newname_text	Blatt umbenennen
xlDialogWorkbookNew	302		Einfügen (Tabellenblatt etc.)
xlDialogWorkbookOptions	284	sheet_name, bound_logical, new_name	Blatt umbenennen
xlDialogWorkbookProtect	417	structure, windows, password	Arbeitsmappe schützen
xlDialogWorkbookTabSplit	415	ratio_num	Arbeitsmappe teilen (keine Dialogbox)
xlDialogWorkbookUnhide	384	sheet_text	Arbeitsmappe einblenden
xlDialogWorkgroup	199	name_array	Gruppe bearbeiten
xlDialogWorkspace	95	fixed, decimals, r1c1, scroll, status, formula, menu_key, remote, entermove, underlines, tools, notes, nav_keys, menu_key_action, drag_drop, show_info	Arbeitsbereichoptionen
xlDialogZoom	256	magnification	Zoom

Tabelle 33: Integrierte Dialogfeldargumente (Forts.)

24 Liste der integrierten Dialogfeldargumente für Diagramme

Die nachfolgenden Argumente beziehen sich ausschließlich auf Diagramme. Das bedeutet, dass ein Diagramm in Ihrem Tabellenblatt vorhanden sein muss, um die Dialogfelder anzeigen zu können. Je nachdem ist es zudem erforderlich, dass in dem Diagramm gewisse Elemente eingeblendet sind, wie beispielsweise die Datentabelle.

Dialogfeldkonstante	Index	Argumentenliste(n)	Deutscher Dialogname
xlDialogAddChartAutoformat	390	name_text, desc_text	Benutzerdefinierter Diagrammtyp hinzufügen
xlDialogAttachText	80	attach_to_num, series_num, point_num	Titel einfügen

Tabelle 34: Dialogfeldargumente für Diagramme

Dialogfeldkonstante	Index	Argumentenliste(n)	Deutscher Dialogname
xlDialogAxes	78	x_primary, y_primary, x_secondary, y_secondary	Diagrammoptionen (Achsen)
xlDialogChartAddData	392	ref, rowcol, titles, categories, replace, series	Daten hinzufügen
xlDialogChartLocation	527		Diagrammspeicherort
xlDialogChartOptions-DataLabels	505		Diagrammoptionen (Datenbeschriftung)
xlDialogChartOptionsDataTable	506		Diagrammoptionen (Datentabelle)
xlDialogChartSourceData	540		Datenquelle
xlDialogChartTrend	350	type, ord_per, forecast, backcast, intercept, equation, r_squared, name	Trendlinie hinzufügen
xlDialogChartType	526		Diagrammtyp
xlDialogChartWizard	288	long, ref, gallery_num, type_num, plot_by, categories, ser_titles, legend, title, x_title, y_title, z_title, number_cats, number_titles	Diagramm-Assistent
xlDialogCombination	73	type_num	AutoFormat
xlDialogCopyChart	147	size_num	Bild kopieren
xlDialogDataLabel	379	show_option, auto_text, show_key	Datenbeschriftung
xlDialogDataSeries	40	rowcol, type_num, date_num, step_value, stop_value, trend	Reihe
xlDialogEditSeries	228	series_num, name_ref, x_ref, y_ref, z_ref, plot_order	Datenreihenbearbeiten
xlDialogErrorbarX	463	include, type, amount, minus	Fehlerlinie X
xlDialogErrorbarY	464	include, type, amount, minus	Fehlerlinie Y
xlDialogFormatCharttype	423	apply_to, group_num, dimension, type_num	Diagrammtyp
xlDialogFormatFont	150	color, backgd, apply, name_text, size_num, bold, italic, underline, strike, outline, shadow, object_id, start_num, char_num	Diagrammschrift

Tabelle 34: Dialogfeldargumente für Diagramme (Forts.)

Dialogfeldkonstante	Index	Argumentenliste(n)	Deutscher Dialogname
xlDialogFormatLegend	88	position_num	Legende formatieren
xlDialogFormatMain	225	type_num, view, overlap, gap_width, vary, drop, hilo, angle, gap_depth, chart_depth, up_down, series_line, labels, doughnut_size	Diagramm formatieren
xlDialogGallery3dArea	193	type_num	Autoformat (3-D-Fläche)
xlDialogGallery3dBar	272	type_num	Autoformat (3-D-Balken)
xlDialogGallery3dColumn	194	type_num	Autoformat (3-D-Säulen)
xlDialogGallery3dLine	195	type_num	Autoformat (3-D-Linien)
xlDialogGallery3dPie	196	type_num	Autoformat (3-D-Kreis)
xlDialogGallery3dSurface	273	type_num	Autoformat (3-D-Oberfläche)
xlDialogGalleryArea	67	type_num, delete_overlay	Autoformat (Flächen)
xlDialogGalleryBar	68	type_num, delete_overlay	Autoformat (Balken)
xlDialogGalleryColumn	69	type_num, delete_overlay	Autoformat (Säulen)
xlDialogGalleryCustom	388	name_text	Benutzerdefiniert
xlDialogGalleryDoughnut	344	type_num, delete_overlay	Autoformat (Ring)
xlDialogGalleryLine	70	type_num, delete_overlay	Autoformat (Linie)
xlDialogGalleryPie	71	type_num, delete_overlay	Autoformat (Kreis)
xlDialogGalleryRadar	249	type_num, delete_overlay	Autoformat (Netz)
xlDialogGalleryScatter	72	type_num, delete_overlay	Autoformat (Punkt XY)
xlDialogGridlines	76	x_major, x_minor, y_major, y_minor, z_major, z_minor, 2D_effect	Diagrammoptionen (Gitternetzlinien)
xlDialogInsertTitle	380	chart, y_primary, x_primary, y_secondary, x_secondary	Titel einfügen

Tabelle 34: Dialogfeldargumente für Diagramme (Forts.)

Dialogfeldkonstante	Index	Argumentenliste(n)	Deutscher Dialogname
xlDialogMainChart	85	type_num, stack, 100, vary, overlap, drop, hilo, overlap%, cluster, angle	Diagramm formatieren
xlDialogMainChartType	185	type_num	Diagramm formatieren
xlDialogOptionsChart	325	display_blanks, plot_visible, size_with_window	Optionen
xlDialogSeriesOptions	557		Verbunddiagramm
xlDialogSeriesX	461	x_ref	Datenreihe formatieren X-Werte
xlDialogSeriesY	462	name_ref, y_ref	Datenreihe formatieren Y-Werte
xlDialogView3d	197	elevation, perspective, rotation, axes, height%, autoscale	3-D-Ansicht (nur bei 3-D-Diagrammen möglich)

Tabelle 34: Dialogfeldargumente für Diagramme (Forts.)

25 Liste der integrierten Dialogfeldargumente für Pivottabellen

Die nachfolgenden Argumente beziehen sich ausschließlich auf Pivottabellen. Das bedeutet, dass eine Pivottabelle in dem Tabellenblatt vorhanden sein muss, um die Dialogfelder anzeigen zu können.

Dialogfeldkonstante	Index	Argumentenliste(n)	Deutscher Dialogname
xlDialogPivotFieldGroup	433	start, end, by, periods	Felder gruppieren (keine Dialogbox)
xlDialogPivotFieldProperties	313	name, pivot_field_name, new_name, orientation, function, formats	PivotTable-Feld
xlDialogPivotFieldUngroup	434		Feldgruppierung aufheben (keine Dialogbox)
xlDialogPivotSolveOrder	568		Lösungsreihenfolge für berechnete Felder
xlDialogPivotTableOptions	567		Optionen

Tabelle 35: Dialogfeldargumente für Pivottabellen

Dialogfeldkonstante	Index	Argumentenliste(n)	Deutscher Dialogname
xlDialogPivotTableWizard	312	type, source, destination, name, row_grand, col_grand, save_data, apply_auto_format, auto_page, reserved	Assistent

Tabelle 35: Dialogfeldargumente für Pivottabellen (Forts.)

26 Liste der integrierten Dialogfeldargumente für Solver

Die nachfolgenden Argumente beziehen sich ausschließlich auf den Solver.

Dialogfeldkonstante	Index	Argumentenliste(n)	Deutscher Dialogname
xlDialogScenarioAdd	307	scen_name, value_array, changing_ref, scen_comment, lokked, hidden	Solver: Szenario hinzufügen
xlDialogScenarioCells	305	changing_ref	Solver: Szenario-Manager
xlDialogScenarioEdit	308	scen_name, new_scenname, value_array, changing_ref, scen_comment, lokked, hidden	Solver: Szenario verändern
xlDialogScenarioMerge	473	source_file	Solver: Szenarien zusammenführen
xlDialogScenarioSummary	311	result_ref, report_type	Solver: Szenarien summieren

Tabelle 36: Dialogfeldargumente für Solver

27 Liste der integrierten Dialogfeldargumente für Steuerelemente

Die nachfolgenden Argumente beziehen sich ausschließlich auf Steuerelemente aus der FORMULAR-Symboleiste:

Dialogfeldkonstante	Index	Argumentenliste(n)	Deutscher Dialogname
xlDialogCheckboxProperties	435	value, link, accel_text, accel2_text, 3d_shading	Steuerelemente formatieren (Ein Kontrollkästchen muss vorhanden sein)

Tabelle 37: Dialogfeldargumente für Steuerelemente

Dialogfeldkonstante	Index	Argumentenliste(n)	Deutscher Dialogname
xlDialogPushbuttonProperties	445	default_logical, cancel_logical, dismiss_logical, help_logical, accel_text, accel_text2	Steuerelement formatieren (Schaltfläche muss vorhanden sein)
xlDialogScrollbarProperties	420	value, min, max, inc, page, link, 3d_shading	Steuerelement formatieren (Bildlaufleiste muss vorhanden sein)

Tabelle 37: Dialogfeldargumente für Steuerelemente (Forts.)

Stichwortverzeichnis

!

#Bezug! 191

#WERT! 70

* 711

*.acs 427

*.bmp 438

*.gif 385

*.jpg 385

*.png 385

*.wmf 413, 438

*.xla 829, 869

*.xlb 302

Backup 304

Speicherpfad 302

*.xls 830, 869

? 711

Numerics

100-jähriger Kalender 160

1904-Datumswerte 166

1904-Datumswerte mit VBA kopieren 170

8.3-Konvention 719

A

Accelerator 640

Access 771, 776

ActionControl 380

ActivePresentation 758

ActiveSheet.Copy 657

ActiveWorkbook 758

ActiveWorkbook.Path 496

ActiveX 245, 714

ActiveX Data Objects 776

ActiveX-Control 404

AdaptiveMenus 348

Add 39, 752

Worksheet 399

Worksheets 861

AddChartAutoFormat 489

AddComment 46, 48

AddDiagram 417

AddFromFile 789

AddFromGuid 788

Add-In 541, 869

Add-Ins 829

Änderungen speichern 840

auf Festplatte kopieren 833

auflisten 834

aus Liste löschen 837

Beschreibung hinzufügen 839

deinstallieren 837

Funktionen einbinden 841

hinzufügen 831

installieren 831

Makros aufrufen 843

Makros über Schaltfläche aufrufen 843

Speicherpfad 834

Titel hinzufügen 839

über Explorer installieren 835

über UserForm löschen 845

Add-Ins-Manager 831

AddItem 271, 605

Addition mit eigenem Zellenwert 83

AddNew 780

AddNode 418

Address 671

Address(False, False) 413

AddShape 410, 508

AddTextEffekt 406

ADO 776

Adresse zu Hyperlink ermitteln 567

AdvancedFilter 212

Änderungsverfolgung 48

aktive Menüleiste 320

Aktivierreihenfolge 290

Aktuelles Datum finden 144

Aktuelles Monatsblatt anzeigen 525

Alle Befehlsleisten Ein- und Ausblenden 325

Alle Dateien schließen 539

Allgemeine Prozeduren 519

AllowEditRange

Add 104

ChangePassword 108

ChangeRange 111

ChangeTitle 109

Delete 105

Protect 107

Unprotect 106

Alphabetisch sortieren 725

Analyse-Funktionen 149

Anker löschen 423

Antwortassistent 580

Anzahl Kalenderwochen 151

Anzeige 622

Anzeige Datumssystem 168

- Anzeigen von Menüs 346
- Application 489
- Application.Run 528
- Application.ScreenUpdating 415
- Application.Volatile 69
- Application.Wait 186
- Application.WorksheetFunction 179
- Application-Ereignis 870
- ApplyCustomType 491
- ARABISCH() 841
- Arbeitsblatt-Menüleiste deaktivieren 320
- Arbeitsmappenschutz 117
- Arbeitszeiten summieren 146f.
- Arbeitszeittabelle 160
- Archivieren von Bezügen 191
- Argumente 586
- Arithmetische Operatoren 900
- Array 378, 424, 605, 645, 857
- Assistant
 - Animation 441
 - BalloonType 435
 - Button 431
 - Checkboxes 435
 - Close 434
 - Filename 428
 - Header 428
 - Icon 429
 - Label 433
 - Mode 433
 - Navigate 440
 - NewBalloon 428
 - Show 428
 - Visible 428
- Assistent
 - Animationen 441
 - Aufzählungspunkte 432
 - Grafiken 438
 - Hyperlinks 439
 - Kontrollkästchen 435
 - Nummerierung 434
 - Pfadangabe 429
 - Schaltflächen 431
 - Text formatieren 437
- AtEndOfLine 741
- AtEndOfStream 741
- Attribute 721
- Attribute-Eigenschaft 916
- Attributes 719
- Aufgabe 769
- Aufgaben auslesen 768
- Aufgabenbereich ein-/ausblenden 583
- Aufgaben-Fenster anzeigen 769
- Auflistungsobjekt 460
- Ausgewählte Bereiche freigeben 113
- Ausschneiden 38
- Ausschneiden und Löschen verhindern 527
- Auto_Close 568f.
- Auto_Open 568f.
- Auto-Berechnung 876
- AutoCalculate 876
- Autofilter
 - Datum 209
 - deaktivieren 211
 - setzen 206
- Autofilter trotz Blattschutz 91
- AutoFit 50, 555, 763
- AutoForm 410
- AutoFormen löschen 422
- Autokorrektur 140
- AutoSize 192
- Axes 473
- AxisGroup 482
- B**
- BackColor 611
- BackStyle 632
- Backward 471
- Bedingte Formatierung 503, 561
- Beep-Anweisung 186
- Befehlsleiste, schützen 328
- Befehlsleisten 301
 - Ein- und Ausblenden 325
 - ermitteln 309
 - integrierte 309
 - löschen 335
 - Type 309
 - zurücksetzen 318
- Befehlsleisten vor Änderungen schützen
 - ab XL2002 332
 - XL97 und XL2000 329
- Befehlsleistendatei
 - finden 302
 - laden 307
 - sichern 303
- Befehlsleisten-Schutz
 - aufheben 331
 - wechseln 332
- Before 665
- Begrüßung nach der Tageszeit 570
- benannte Bereiche 114, 137
- Benannte Datums- und Zeitformate 139
- Benannter Bereich 660
- Benutzerberechtigte Bereiche
 - Bereich ändern 111
 - hinzufügen 104
 - löschen 105
 - Passwort ändern 108

- schützen 107
- Titel ändern 108
- Benutzerberechtigungen 104
- Benutzerdefinierte, Tabellenfunktionen 61
- Benutzerdefinierte Ansichten 121, 310
- Benutzername
 - unter Excel 127
 - unter Windows 114
- Berechnung von Datumsdifferenzen 175
- Berechnungsart 195
- Berechnungs-Spezifikationen 920
- Bereich als Grafik exportieren 385
- Bereich in Grafik umwandeln 383
- Bereichsauswahl verhindern 554
- Bereichsname 114
- Beschriften von Registerreibern 544
- Bestandveränderungen dokumentieren 137
- Betriebssystem-Informationen 125
- Bezeichnungsfeld 183, 188, 288
- Bezeichnungsfelder 594
- Bezüge archivieren 191
- Bezug-Fehler reparieren 194
- Bibliothek 653, 714
- Bibliothek aktivieren 734
- Bild erzeugen 761
- Bild siehe Grafik
- Bildläufe durchführen 558
- Bildlaufleiste 287
- Bildlaufleisten 536, 618
- Bildschirmaktualisierung 336, 536
- Bildschirmeinstellung 571
- Bitmap 438
- Blasendiagramm 513
- Blattname
 - Excel 803
 - VBA 803
- Blattname ermitteln 67
- Blattnamen ermitteln 70
- Blattregister 536
- Blattregisterkarten 869
- Blattschutz ohne Blattschutz 559
- Börsenkurse abfragen 682
- Brightness 402
- BuildPath 736
- BuiltIn 309, 319, 786
- BuiltinDocumentProperties 55, 840
- ByVal 537
- C**
- Calculate 90
- Calculation 195
- Call 237, 813
- Caller 67
- Callout löschen 422
- Cancel 377, 551
- CancelDefault 819
- Cancel-Parameter 538
- Canvas löschen 422
- Caption 343, 592
- Cascading Style Sheets 681, 696
- casesensitiv 634
- CDbl 284
- Change-Ereignis,
 - CommandBarComboBox 871
- ChangeFileOpenDirectory 752
- Characters 506
- ChartArea 452
- ChartObjects 450
- ChartTitle 458
- ChartType 450
- ChDir 714
- ChDrive 714
- CheckBox 258, 610
 - Wert auslesen 260
- Children 418
- Chr 379
- Clear 207, 607
- ClearComments 52
- ClearContents 38, 220
- Clipart einfügen 583
- Clipboard 878
- ClipboardFormats 917
- CLng 288
- Codegerüst 522
- Codemodul, Fenster
 - anzeigen 796
 - auflisten 795
- Codename 239, 803
 - ändern 805
 - auslesen 803
 - nach Tabellenname benennen 808
- Codezeilen umbrechen 35
- Collection 820
- ColorIndex 452
- ColorIndex-Eigenschaft 905
- Column 741
- ColumnHead 603
- ColumnsCount 603
- ColumnWidths 604
- COM 706
- ComboBox 279, 604
- CommandBar 309, 584
- CommandBarComboBox 871
- CommandButton 184, 597
- Component Object Model 706
- Computername 125

- Connection 777
- Contrast 402
- Controls 311, 635
- ControlTipText 632
- Copy 38f., 351
 - Picture 398
- CopyFace 311
- CopyFile 717, 736, 833
- CopyFolder 736
- CopyFromRecordset 778
- CopyPicture 384, 509, 761
- CopyToRange 212
- Count 96, 425
- CreateDate 719
- CreateNewDocument 670
- CreateObject 715
- CreatePivotTable 224
- CreateTextFile 737
- Criterial 207
- Criteria2 207
- CriteriaRange 212, 215
- Crosses 474
- CrossesAt 474
- CSS 681, 696
- CurDir 714
- Cursor mit Textbox 389
- Cursorbewegungen protokollieren 554
- Cursorbild ändern 556
- Cursor-Eigenschaft 556
- CustomDocumentProperties 55
- Cut 38
- CutCopyMode 757
- CVErr 73
- D**
- DAO 776
- Data Access Objects 776
- DataBodyRange 227
- DataLabel 506
- DataObject 781, 878
- DataRange 227
- DataTable 483
- DATEDIF 175
- DateDiff 177
- Date-Funktion 133
- Dateiattribute 719, 721
- Datei-Eigenschaften 718
- Dateien
 - Attribute 721
 - kopieren 717
 - löschen 710
 - sortieren 725
 - suchen 724
 - umbenennen 711
- Unterverzeichnisse auslesen 726
- verschieben 715
- Dateigröße 719
- Dateiname ermitteln 67
- Dateinamen 719
- Dateinamen auslesen 723
- Dateipfad ermitteln 67
- Dateityp 719
- Dateizugriffe dokumentieren 135
- DateLastAccessed 719
- Datenbeschriftung 506
- Datenfeld 275, 857
- Datentabelle 482
- Datentyp 596f.
- Datentypen 909
- DatePart 151
- DateSerial 145, 182
- Datum finden 144
- Datum/Zeit 129
- Datums- und Zeitformate 911
- Datumsdifferenzen berechnen 175
- Datumsformat 129
- Datums-Funktionen bei 1904-
 - Datumswerten 174
- Datumssystem
 - anzeigen 169
 - wechseln 167
- Datumswerte mit anderem Datumssystem
 - importieren 172
- Day-Funktion 145
- Debug.Print 797, 857
- Default 668
- DefaultFilePath 307
- Delete 38, 509, 676
 - DrawingObjects 421
 - Pictures 399
 - Shapes 421
- DeleteChartAutoFormat 490
- DeleteFolder 736
- DeleteLines 816
- Description 489, 786
- Destination 38
- Deutsch – Englisch 889
- Diagramm 417
 - Achsen formatieren 473
 - Achseneinstellungen rückgängig machen 479
 - Achseneinstellungen speichern 477
- Axes 473
- Bedingte Formatierung
 - Säulen 503
 - Werte 505

- benutzerdefiniert anwenden 491
- benutzerdefiniert erstellen 488
- benutzerdefiniert löschen 490
- Bestimmtheitsmaß 469
- Böden 455
- Datenpunkte 462
- Datenpunkte formatieren 462
- Datenpunkt-Farben aus Zelle 501
- Datenreihe formatieren 460
- Datentabelle 482
- Dialogfeldargumente 930
- drucken 495
- Exportieren 497
- Gleichungen 469
- Halbtransparent 507
- Hintergrundfarbe 452
- Hintergrundmuster 452
- Legende 484
- Linienart 456
- Liniendicke 456
- löschen 493
- Markierer und Linien formatieren 462
- Minuswerte invertieren 502
- Muster 456
- Musterfarbe 452
- Oberflächen 486
- Objekt-Hierarchie 445
- Objektnamen ermitteln 499
- Pivot 491
- Primärachse 481
- rotieren lassen 486
- Rückwärtstrend 470
- Säulen mit Grafiken 496
- Sekundärachse 481
- Spezifikationen 921
- Titel 458
- Trendlinien entfernen 466
- Trendlinien hinzufügen 463
- Trendlinien-Typen 467
- umbenennen 450
- Verbund 510
- Vorwärtstrend 470
- Wände 455
- Zeichnungsfläche 452
- Zellfarben übergeben 499
- Diagramm in UserForm 643
- Diagrammblatt löschen 548
- Diagramme löschen 422
- Diagrammecken abrunden 515
- Diagrammhintergrund 452
- Diagrammindex 447
- Diagrammtitel 457
- Diagrammtypen 447
- Dialogfeldargumente 922
- DieseArbeitsmappe 521
- DIN 1355 150
- Dir 304
- Direktbereich löschen 797
- Direktfenster 96
- Diskussion 769
- DislayEquation 469
- Display 769
- DisplayAlerts 38, 192, 668
- DisplayRSquared 469
- dll 734
- DLLs 706
- Do...Loop 144
- Document Object Model 693
- DoEvents 188
- Dokument erstellen (Hyperlink) 670
- Dokumenteigenschaften 54
- Dokumenteigenschaften 839
- DOM 693
- Doppelklick deaktivieren 330
- Doppelte Datensätze ausblenden 213
- DOS 710
- Drag and Drop 527
- DrawingObjects 414, 421
- Drehfeld 156, 285
- Drehfelder 620
- Drive 719
- DriveExists 736
- DriveLetter 732
- Drives 731
- DriveType 731
- DropDown 216, 605
- DropDown erzeugen 342
- DropDown-Feld 869
- Dynamic Link Libraries 706
- Dynamic Link Library 734
- E**
- E/A-Modus 739
- Early Binding 653, 693, 709
- eigene Grafik als Schaltflächensymbol 881
- Eigenschaften 54, 718
- Eigenschaftenfenster, Steuerelement 157
- Eigenschaftsfenster, Projekt Explorer 136
- Ein- und Ausblenden
 - Kontextmenüs 323
 - Menüleisten 320
 - Symboleisten 321
- ein-/ausblenden von Spalten 558
- Einfügen 38f.
- Eingabe von Minuszeiten 170
- Eingebettete OLE-Objekte löschen 422
- Einschränkungen 918

- E-Mail 769
 - BCC 654
 - Body 654
 - CC 654
 - Display 654
 - HTML formatiert 656
 - HTMLBody 656
 - ReadReceiptRequested 654
 - Send 654
 - Serienmail 660
 - Subject 654
 - Tabellenblatt 657
 - To 654
 - versenden 653
 - Zellbereich 659
- E-Mail-Adressen 674
- E-Mail-Links löschen 678
- Enabled 600, 615
- EnableEvents 538
- EnableSelection 95
- Englisch – Deutsch 889
- englische Menübeschriftungen 315
- Entwicklungsumgebung 785, 797, 818
 - ein- und ausblenden 794
- Entwurfsmodus 157
- Environ 49, 53, 125, 302, 714, 733
- Ereignisbedingte Formatierung 561
- Ereignis-Prozedur erstellen 814
- Ereignis-Prozeduren 520
- Ereignisse 519
 - deaktivieren 538
 - ignorieren 530
- Ereignisse aus- und einschalten 531
- Ergänzen einer Textdatei 739
- Err 191
 - Clear 192, 546
 - Number 115, 192, 546
- Ersatzfunktion für DateDif 179
- Erstellen
 - Kontextmenü 341
 - Menüleiste 335
 - Symbolleiste 338
- Erstellen eines Verzeichnisses 727
- Erzeugen einer Textdatei 737
- Excel kann warten 186
- Excel10.xlb 308
- Excel4-Makro 87, 153
- Excel-Benutzername 127
- Excel-Blattname 803
- Excel-Versionsnummer 584
- Execute 346, 348
- Existiert Tabellenblatt 71
- Exit Sub 672
- Expanded 648
- Export 497
- Export nach
 - Access-Tabelle 779
 - PowerPoint 760
 - Textdatei 748
 - Word 754
 - Zwischenablage 781
 - Zwischenablage nach Word 757
- exportieren 709
- eXtensible Markup Language 689
- Extern 709
- F**
- FaceId 311, 341
- FaceID-Übersicht 381
- Fadenkreuz 585
- Farben sortieren 231
- Farbindex 561
- Farbkonstanten 34
- Farbpalette 561, 853
- Farbpalette zurücksetzen 856
- Farbtabelle 43
- Fehlernummer 73
- Fehlerrountinen, vor Erstellung von
 - Befehlsleisten 334
- Feiertage
 - mit Formeln 152
 - mit VBA 155
- Fensteroptionen 880
- File System Object 734
- FileCopy 304, 714
- FileExists 736
- FileName 724
- Files 723
- FileSearch 724
- FileSystemObject 715
- Filter 206
- FilterMode 214
- Find 558
- FindControl 314, 530
- Find-Methode 145
- Floor 455
- Flüchtige Tabellenblatffunktionen 69
- Fokus verwalten 257
- FolderExists 736
- FollowHyperlink 651, 675
- For...Each 413
- ForAppending 739
- FormatCondition 906
- Format-Funktion 132
- Formatierung bedingt 563
- Formatierung nicht gesperrter Zellen 99
- Formeln anzeigen 152

Formeln in Kommentaren 557
Formelzellen schützen 90
Formula1 216
Formula2 216
FormulaLocal 192
Formular-Elemente löschen 422
Formular-Schaltfläche
 erstellen 248
 löschen 249
 Makro zuweisen 250
ForReading 739
Forward 471
Frame 599
Freihandformen löschen 422
FSO 734
FullName 832
FullPath 787
Funktion
 DATEDIF 175
 DateDiff 177
 Eastern 155
 Holiday 157
 KALENDERWOCHE 149
 NegativeTime 165
 WeekNumberDIN 150
Funktion Formelanzeige 154
Funktionen 57
 Fehlerwerte 73
 Hilfetext einfügen 843
 in Add-In 841
Funktionsassistent 62, 842
Funktions-Kategorien 62
Funktionsname in Großbuchstaben
 ändern 58
Fußzeilen 40
fx 58, 62

G

Gemischte Typen löschen 423
Genaues Datum finden 144
Gesperrte Zellen einfärben 88
GetAbsolutePathName 736
GetBaseName 736
GetDefaultFolder 763
GetDrive 736
GetDriveName 736
GetFile 718, 736
GetFileName 736
GetFolder 736
GetFromClipboard 783
GetParentFolderName 737
GetSpecialFolder 737
GetTempName 737

GetText 783
GIF 387
Gitterzäune 164
GotFocus 271
GoTo 115
Goto 288, 311, 587
GoTo-Methode 558
Grafik
 an Bereich anpassen 394
 an Zellengröße anpassen 393
 animiert 404
 aus Bereich erzeugen 383, 385
 bedingt einfügen 403
 bei Klick einfügen 392
 benennen 397
 formatieren 400
 in Zelle einfügen 390
 kopieren 398
 löschen 399, 423
 Verknüpfung 388
 zentrieren 396
Grafikobjekte 245
Größe von Verzeichnis ermitteln 728
Großbuchstaben 719
Group 424
GroupName 264, 608
Gruppen löschen 422
Gruppenfeld 264
Gruppieren, Objekte (Array) 424
Gruppierung aufheben 414
Gültigkeit 216
GUID 787

H

Halbtransparent 507
Handled 819
HasDataTable 483
HasDiagram 420
HasFormula 91
HasLegend 485
HasTitle 458
Height 393
Hetterich-Formel 152
Hilfedatei VBE 785
HTML 656, 680
HTMLBody 656
HTML-Code ansehen 679
Hundertstelsekunden 130
Hyperlink 302, 440, 650
 Add 664
 Address 664, 671
 Anchor 664
 Anzeigetext 566

- CreateNewDocument 670
- Delete 676
- E-Mail-Links löschen 678
- extern 671
- FollowHyperlink 675
- in Nebenzelle ausgeben 676
- Inhaltsverzeichnis 665
- Intern 663
- löschen 676
- lokal 672
- Mappe 673
- mit Makro verknüpfen 565
- Quickinfo 664
- Schaltfläche 675
- ScreenTip 664
- SubAddress 664
- Textmarke 672
- Weblinks löschen 678
- Zelladresse 667
- Hyperlinks über Dropdownfeld 358
- HyperText Markup Language 680
- I**
- ID 352
- If ... Then ... Else 414, 431
- IgnoreBlank 216
- IIf 136, 332, 367
- Image 622
- I-Mauszeiger 556
- Import aus
 - Access-Tabelle 776
 - Outlook Aufgaben auslesen 768
 - Outlook Kalender auslesen 765
 - Outlook Posteingang auslesen 767
 - Outlook-Kontakte auslesen 762
 - PowerPoint 758
 - Textdatei 743
 - Word 751
 - Zwischenablage 783
- importieren 709
- Importieren von Datumswerten 172
- InCellDropdown 216
- IncrementLeft 515, 761
- IncrementTop 515, 761
- Informationen des Betriebssystems 125
- Informationsfunktion 67
- Infotyp 67
- Inhaltsverzeichnis erstellen 665
- Initialize 596
- InputBox 191, 346, 414
 - Default 112
 - Type Range 112
- InputBox-Methode 144, 898
 - Type 144
- Insert 391, 665
- Installed 832
- InStr 192, 566
- Instr 635
- Int 857
- integrierte Befehlsleisten 309
- Integrierte Dialogfeldargumente 922
- Integrierte Steuerelemente in Symbolleiste
 - einfügen 351
- IntelliSense 59, 96, 180, 261, 553
- Interior.ColorIndex 452
- Interior.PatternColorIndex 452
- Internet 440
- Intersect 138
- Invertieren 502
- InvertIfNegative 502
- Is Nothing 46
- IsAddin 540, 839
- IsDate 183
- IsEmpty 183
- IsError 74
- IsMissing 77, 182
- IsNull 260
- IsNumeric 596
- ISTFEHLER 449
- IsWorkbookOpen 72
- Item 418
- Iteration 82
- J**
- Journaleintrag 769
- JPEG 387
- K**
- Kalender 769
- Kalender auslesen 765
- Kalenderwoche 149
- Kalenderwochen eines Jahres 151
- Kaskaden-Menü erstellen 373
- kill 710
- kirchliche Feiertage 152
- Klammerpaar 68
- Klassen-ID 787
- Klassenmodul 818
- Kleinbuchstaben 719
- Kombinationsfeld 131
 - füllen 279
- Kombinationsfelder 604
- Kommentare
 - auslesen 49
 - einfügen 46
 - ergänzen 48
 - formatieren 51

- hinzufügen 192
- löschen 52, 192, 422, 557
 - mit Formeltext 557
- Kommentieren 34
- Konstanten 265
- Kontakt 769
- Kontakte auslesen 762
- Kontextmenüs 301
 - deaktivieren 324
 - des VBA-Projektes 809
 - Einblenden 323
 - ersetzen 373
 - mit Datum und Zeit 341
 - Positionieren 323, 341
 - Registerreiter 118
 - um Schaltflächen erweitern 370
- Kontrollkästchen 258, 610
 - Wert auslesen 260
 - zurücksetzen 267
- Konvertieren, von Datumswerten 172
- Kopf- und Fußzeilen 907
- Kopf- und Fußzeilen mit Bildern 41
- Kopfzeilen 40
- Kopieren 38
- Kopieren von Dateien 717
- kürzeste Makro 531

L

- Label 183, 288, 594
- LabelRange 227
- Late Binding 653, 693
- Late binding 709
- Laufwerk 719, 730
- Laufwerkbuchstaben 731
- Laufwerk-Eigenschaften 733
- Laufzeitfehler 258
- LBound 605
- LCase 719
- Lebensalter berechnen 182
- Leere Spalten löschen 685
- Leerzeilen löschen 685
- Left 391
- Legende verwalten 484
- Legendensymbol 483
- Length 506
- Letzte belegte Zelle 35
- Line 741
- LineStyle 456
- Linien und Pfeile löschen 423
- Link 388
- LinkedCell 156
- List 275
- ListBox 600

- ListCount 273
- Liste 692
- Listenfeld 271
 - mehrspaltiges 277
 - Mit Werten füllen 272
- Listenfeldeinträge ändern 274
- Listenfelder 600
- ListFillRange 277
- ListIndex 604
- ListStyle 272, 601
- Locked 91, 100
- Löschen 37f.
 - Anker 423
 - AutoFormen 422
 - Bestimmte Objekte 422
 - Callout 422
 - Diagramme 422
 - Eingebettete OLE-Objekte 422
 - Formular-Elemente 422
 - Freihandformen 422
 - Gemischte Typen 423
 - Grafiken 423
 - Gruppen 422
 - Kommentare 422
 - Linien und Pfeile 423
 - Media-Daten 423
 - Objekte 421
 - Platzhalter 423
 - Steuerelemente 423
 - Tabellen 423
 - Textboxen 423
 - Texteffekte (WordArt) 423
 - Verlinkte Bilder 423
 - Verlinkte OLE-Objekte 423
- Löschen eines Verzeichnisses 728
- Löschen von Befehlsleisten 335
- Löschen von Blättern verhindern 117
- Löschen von Dateien 710
- Löschen von Diagrammen 493
- Löschen von Menüleisten 337
- logische Negation 321
- Lohn berechnen 163
- LookIn 724
- Lottoschein 857

M

- Major 787
- Makro rückgängig machen 588
- Makro-Rekorder 30, 519
- Makros aus Add-In aufrufen 843
- Makro-Viren 710
- Mappe löschen (Kill) 658
- Mappe.xlt 572

MarkerStyle 463
Mausbewegung, Ereignis auslösen 272
Mauszeiger einstellen 556
Me 100, 266, 558, 567
Media-Daten löschen 423
Meldungsfenster als Grafik 532
MenuBar 336
Menü
 aus- und einblenden 314
 deaktivieren und aktivieren 313
 über ID ansprechen 314
Menü suchen und aufklappen 346
Menü-Auflistung im Direktfenster 312
Menüleisten 301
 Ein- und Ausblenden 320
 erstellen 335
 löschen 337
Menüpunkte mit Hyperlinks 358
Menüs ermitteln 311
Menüs vollständig anzeigen 347
Merge 197
Metadateien 438
Microsoft-Scripting-Runtime 734
Mid 566
Minor 787
Minuswerte invertieren 502
Minuszeiten 164
 eingeben 170
MkDir 714, 727
Modul
 benennen 809
 erstellen 808
 löschen 810
Modulcode löschen 817
Module, überprüfen 811
Module auslesen 800
MonthName 274
MouseMove 272
MoveFile 715, 737
MoveFolder 716, 737
MoveRight 757
MS Agent Character File 427
MsgBox 413, 696
 Fehlermeldung 144
MsgBox-Konstanten 897
MultiLine 596
Multipage 614
Multiseiten 614
MultiSelect 601
Muttertag 152, 158

N

Name 489, 719
name...as... 711

NameLocal 309
Namen
 definieren 137, 153
 Liste einfügen 162
Namensfeld 114
Names 660
Navigate 405, 440
Negation 321
Negative Zeitdifferenz 164
Neuberechnung 195
Neue Arbeitsmappe 583
New 777
NewSearch 726
Nordwestpfeil-Zeiger 556
Normal.dot 572
Notiz 769
Notiz erstellen 770
Not-Operator 321
Now-Funktion 133
Null 259

O

Oberflächen-Diagramm 486
Objekte, löschen 421
Objekte lokalisieren 413
Objekt-Katalog 644
Objekt-Typ 423
Office-Assistent siehe Assistent
Office-Web-Komponente 637
Offline 440
Offset 587
OnAction, Formular-Steuerelemente 249
OnDoubleClick 330
OnKey 580, 586, 879, 914
Online 440
OnRepeat 588
OnTime 183, 574, 686, 814
OnUndo 85, 588
OpenText 746
OpenTextFile 739
Option Private 528
Option Private Module 835
Optional Parameter 182, 877
OptionButton 608
Optionsfelder 608
 auswerten 266
 zurücksetzen 267
Ordner 730
Organigramm 416
Originaldatei *.xlb laden 308
Osterformel 152
Oster-Funktion 155
Ostersonntag 152
Outlook 653, 762

Outlook Aufgaben-Fenster anzeigen 769
Outlook Notiz erstellen 770
Outlook-Standardordner 763
OWC 637

P

Pages 615
Parameter 537, 587
Parameterinfo 110
Parent 567
Parentfolder 719
PasswordChar 281, 633
Passwort 633
Passworтеingabe mit Sternchen 281
Paste 38, 384, 509, 757
 Pictures 398
PasteFace 882
Path 719
Pattern 452, 456
PatternColorIndex 452
Personalisierte Menüs 347
PERSONL.xls 519, 584
Pfad 719
 ActiveWorkbook.Path 496
Pfad ermitteln 67
Pfeiltasten 585
PICT-Dateien 438
PictureFile 497
Pictures 383, 391
 Copy 398
 Delete 399
 Name 397
 Paste 398
PivotCaches 223
Pivot-Diagramm 491
PivotFields 224
PivotItem 228
PivotItems 224
Pivot-Tabelle
 Code bereinigen 224
 Datenpunkte ausblenden 228
 erzeugen 221
 Zwei auf einem Blatt 225
Pivot-Tabellen, löschen 225
Pivottabellen
 Dialogfeldargumente 933
 Spezifikationen 921
PivotTables 223
PivotTableWizard 224
Platzhalter 711
Platzhalter löschen 423
Platzhalterzeichen 281
PlotArea 452

PNG 387
Point(s) 462
Popupmenüs 315
Posteingang auslesen 767
PowerPoint 758, 874
pptLayoutBlank 761
Präfix
 g 359, 588
 m 359, 588
Präsentation in Excel 874
Preserve 425
PresetShape 407
Primärachse 481
Private 835
ProgId 254
Projektname auslesen und ändern 791
Projektschutz aufheben 792
Protect 90
ProtectContents 97
Protection 328
 AllowFormattingCell 99
 AllowInsertingColumns 101
 Übersicht-Tabelle 95
Protection-Objekt 95
Prozedur auslesen 802
Prozedur erstellen und aufrufen 813
Prozedur mit Parametern 268
Prozeduren
 löschen 816
 überprüfen 812
Prozeduren auslesen 800
PutInClipboard 782, 878

Q

Query 771
QueryTable 684
Quickinfo 60, 631, 664

R

Rahmen 599
Randomize 857
Range zusammensetzen 147
Read 741
ReadAll 741
ReadLine 741
ReadOnly 120
Recordset 777
ReDim 425
ReDim Preserve 425
RefEdit 624
Reference 786
Refresh 684
RefreshAll 685

- Register 613
- Registerfarbe 356
- Registerleiste 868
- Registerreiter
 - beschriften 544
 - nummerieren 542
- rekursiv 865
- Remove 810, 820
- RemoveItem 275, 606
- Replace 240
- Reset 318
- Reset.Colors 856
- RGB 45, 414, 618
- RGB-Farben 44
- RGB-Funktion 562, 904
- Rmdir 714, 728
- Rnd 856f., 865
- RÖMISCH() 841
- Rosenmontag 152
- RoundedCorner 515
- RowSource 601
- Rückwärtstrend 470
- Run 566, 813
- RunAutoMacros 568

- S**
- Sanduhrzeiger 556
- SaveAsUI 537
- Saved 120, 185
- ScaleHeight 759
- ScaleHight 515
- ScaleType 474
- ScaleWidth 515, 759
- Schaltfläche, einem Menü hinzufügen 361
- Schaltfläche mit Hyperlink 675
- Schaltflächen 597
 - Darstellungsart 356
 - mit Quickinfo 353
 - mit Tastenkombinationen 880
 - über ID löschen 357
- Schaltflächen-IDs, ermitteln 352
- Schaltflächensymbol, kopieren 311
- Schaltflächensymbole 381
- Schaltjahre 134
- Schema 696
- Schematische Darstellung 416
- Schleife, For ... Each 562
- Schreibgeschützte Datei schließen 119
- Schreibschutz 120
- Schutz der benutzerberechtigten Bereiche
 - aufheben 106
- Schutzoptionen
 - ändern 101
 - auslesen 94
 - einstellen 98
- ScreenUpdating 30, 336, 513, 569
- Scripting 715
- Scroll 558
- ScrollBar 287, 618
- Scrollbar 487
- SearchSubFolders 726
- Seitenumbruchvorschau 77
- Seitenzahlen 77
- Sekundärachse 481
- Sekundenbruchteile 130
- Select 424
- Select ... Case 138, 158, 423, 562
- Select Case True 238
- Selected 602
- SelectedItem 613
- selektieren 30
- SendKeys 797
- Serien-Mail
 - mit einheitlichem Inhalt 660
 - mit unterschiedlichem Inhalt 661
- SeriesCollection 460
- Server 715
- Set 55, 197, 716
- SetAttr 721
- SetFocus 598, 796, 798
- SetSourceData 514
- SetText 782
- Shape 383, 758
- ShareName 731
- SheetExists 71
- Sheets-Auflistung 237
- Short Date 138
- ShortcutText 881
- ShortName 719
- ShortPath 719
- Show 658
- ShowAllData 214
- ShowLegendKey 483
- ShowPopup 323
- Sicherheitskopie, eines Tabellenblattes 192
- Sicherheitsstufen 29
- Sichtbereich 558
- Signaltöne erzeugen 186
- Signalton 186
- Size 719, 728
- Skip 741
- SkipLine 741
- Slide 758
- Smart Tags 25, 702
 - erstellen 703
 - löschen 706
- Solver, Dialogfeldargumente 934

Sommerzeit 158
Sonderzeichen 377
SortBy 725
Sortieren
 nach eigener Farbfolge 234
 nach Farben 231
Sortieren von Dateien 725
SortOrder, =... 725
SPALTE 76
Spalten ein-/ausblenden 558
Spaltenbreite automatisch anpassen 555
Spaltenbuchstabe 76
SpecialCells 53, 172, 686
 xlCellTypeComments 195
 xlCellTypeFormulas 90, 192
SpecialCells-Methode 903
SpecialEffect 273
Spesen Berechnung 161
Spezialfilter
 Datensätze einblenden 214
 Doppelte Datensätze ausblenden 213
 Kriterienbereich 212
 mehr als zwei Kriterien 211
Spezifikationen 918
 Arbeitsblätter 918
 Arbeitsgruppen 919
 Arbeitsmappen 918
 Berechnung 920
 Diagramme 921
 Pivottabellenbericht 921
SpinButton 156, 285, 620
SpinDown 621
SpinUp 621
Split 77
Sprachtrainer 864
Spreadsheet 637
Sprechblase, Icon 428
Sprechblase siehe Assistent
SQL 775
Standardkomponente 808
Standardkonfiguration von
 Symbolleisten 318
Standardmauszeiger 556
Standardordner 763
Standardspeicherpfad 307
Start 506
StartUpPosition 626
Static 560
StatusBar 654
Statuszeile 168
 Auto-Berechnung 876
Steuerelement 157
 temporär 343
 Zugriffstaste 640

Steuerelemente 245
 alle löschen 256
 anzeigen 356
 Dialogfeldargumente 934
 Dropdown 344
 Eigenschaften 261
 eines Typs löschen 257
 ermitteln 311
 erstellen 254
 Formular-Symbolleiste 246
 Gruppen 264
 kopieren 349
 löschen 256
 mit ID erstellen 356
 mit Tastatur aktivieren 290
 suchen 346
Steuerelemente löschen 423
Steuerelement-Toolbox 250
Stoppuhr 183
Strukturschutz 118
Stylesheets 698
SubAddress 566, 671
Sub-Anweisung mit Parametern 326
SubFolders 723
Suchen 583, 667
Suchen nach Dateien 724
Symbole einfügen 377
Symbolleiste
 Formular 246
 mit zwei Ebenen 354
 Steuerelement-Toolbox 156
 vor Änderungen schützen 328
Symbolleiste neben Standard-Symbolleiste
 andocken 340
Symbolleisten 301
 Anpassen 329
 Deaktivieren 321
 Ein- und Ausblenden 321
 erstellen 338
 schwebend positionieren 339
 verankern 338
 zurücksetzen 318
Symbolleistendatei 303
 Speicherpfad 302
Symbolleistenkonfigurationen, speichern 305
Symbolleistentypen Ein- und Ausblenden 326
Systemdatum 133
Systeminformationen 53, 733

T

Tabellen löschen 423
Tabellenblätter sortieren
 Alphabetisch 236
 Codennamen 239

- Farben 242
- Numerisch 241
- Tabellenblatt, überprüfen 71
- Tabellenblatt ausblenden 136
- Tabellenblatt in UserForm 636
- Tabellenblatt-Übersicht 868
- Tabellen-Funktionen, benutzerdefiniert 841
- Tabellenfunktionen
 - Benutzerdefiniert 61
 - in VBA 58
- Tabellenregister
 - beschriften 526
 - nummerieren 542
- Tabellenreiter, färben 363
- TableDestination 226
- TableRange2 225
- TabStrip 613
- Tag 364, 378
- Tageskalender 156
- TakeFocusOnClick 258
- Target 48, 552
 - auswerten 552
- Target.Address 452
- Task Pane 583
- Tasten-Code-Konstanten 291
 - auf der Zehnertastatur 291
- Tastenkombination
 - Aufgabenbereich 583
 - Bildschirmfoto 533
 - Datei schließen 539
 - Datum 139
 - Diagrammblatt einfügen 543
 - Excel schließen 539
 - Formeln anzeigen 152
 - Menü anzeigen 346
 - neues Blatt einfügen 542
 - Speichern 537
 - Speichern unter 537
 - Uhrzeit 139
 - Zeilenumbruch 585
 - Zellen formatieren 86
- Tastenkombinationen erstellen 579
- Tastenkombinationstext anzeigen 881
- Teilergebnis 209
- Temporäre Steuerelemente 343
- TextBox 284, 389, 594
- Textboxen löschen 423
- Textdatei
 - auslesen 740
 - ergänzen 739
 - erzeugen 737
 - exportieren 748
 - importieren 743
 - Leerzeilen einfügen 739
 - Texteffekte (WordArt) löschen 423
 - Textfeld, Werte umwandeln 283
 - Textfelder 594
 - Textmarke 673
 - TextToDisplay 566
 - ThisWorkbook.FullName 654
 - Time-Funktion 133
 - TimeValue 686
 - ToggleButton 136, 262, 612
 - Toolbar List 330
 - TooltipText 353
 - Top 391
 - TopLeftCell 413
 - Transformation 698
 - Transparency 509
 - Transparent 507
 - TreeView 647
 - Trendlinien 463, 466
 - Trendlinien-Typen 467
 - Trennlinie in Befehlsleisten 371, 378
 - TripleState 259
 - TristateFalse 739
 - TristateTrue 739
 - TristateUseDefault 739
 - Type 192, 195, 719, 730
 - InputBox-Methode 144
 - TypeName 544
 - Typ-Umwandlungsfunktionen 283, 597, 910

U

 - UBound 275, 605
 - UCase 238, 719
 - UDF 65
 - Uhrzeit
 - mit Autokorrektur 140
 - ohne Doppelpunkt 141
 - Uhrzeit in Statuszeile 575
 - Umbenennen von Dateien 711
 - umbrechen 35
 - Umschaltfelder 612
 - Umschaltfläche 136, 262, 366
 - Undo 85
 - Ungroup 414
 - Unicode-Zeichensatz 377
 - Unified Resource Locator 671
 - Uniform Resource Identifier 702
 - Union 587
 - Unique 213
 - Unload Me 599
 - Untermenü des Untermenüs deaktivieren und aktivieren 317
 - Untermenü-Auflistung im Direktfenster 315
 - Untermenüs deaktivieren und aktivieren 317

Unterstrich 35
Unterverzeichnisse auslesen 726
Update 780
URI 702
URL 671
UsedRange 50
UserForm 188
 Accelerator 640
 AddItem 605
 Anzeige 622
 Array 605
 BackStyle 632
 Bezeichnungsfeld 594
 Bildlaufleisten 618
 CheckBox 610
 Clear 607
 ColumnHead 603
 ColumnsCount 603
 ColumnWidths 604
 ComboBox 604
 CommandButton 597
 Control 600
 Controls 635
 ControlTipText 632
 Diagramm 643
 Drehfelder 620
 DropDown 605
 Enabled 600, 615
 Expanded 648
 FollowHyperlink 651
 Frame 599
 Größe 626
 GroupName 608
 Height 626, 635
 Hyperlink 650
 Image 622
 Kombinationsfelder 604
 Kontrollkästchen 610
 Label 594
 Left 635
 ListBox 600
 Listenfelder 600
 Listenfelder mehrspaltig 603
 ListIndex 604
 ListStyle 601
 MultiLine 596
 Multipage 614
 Multiseiten 614
 MultiSelect 601
 OptionButton 608
 Optionsfelder 608
 OWC 637
 Pages 615
 PasswordChar 633

 Passwort 633
 Position 626
 Quickinfo 631
 Rahmen 599
 RefEdit 624
 Register 613
 RemoveItem 606
 RowSource 601
 Schaltflächen 597
 ScrollBar 618
 Selected 602
 SelectedItem 613
 SetFocus 598
 SpinButton 620
 SpinDown 621
 SpinUp 621
 Spreadsheet 637
 StartPosition 626
 Tabellenblatt 636
 TabStrip 613
 TextBox 594
 Textfelder 594
 ToggleButton 612
 Top 635
 TreeView 647
 Umschaltfelder 612
 Unload Me 599
 vbModal 640
 Werkzeugsammlung 651
 Width 626, 635
 Zugriffstaste 640
UserName 49, 63
Usernamen 114
UserPicture 497

V

Val 240, 303, 379
Value2 210
Variablenliste 326
VBA DateDiff 177
VBA-Codename 803
VBA-Editor 818
VBA-Funktionen 60
VBA-Projekt 519, 818
VBA-Projekte auslesen 798
VBA-Projekteigenschaften, auslesen 790
VBA-Projektschutz
 aufheben 792
 ermitteln 792
VBA-Schlüsselwörter 892
VBComponents 790
vbCrLf 295
VBE 818

VBE-Entwicklungsumgebung 785
 VBE-Menüleiste, bearbeiten 818
 VBIDE 786
 VBIDE-Bibliothek 818
 vbModal 640
 vbModeless 597
 vbTab 295
 Verbund-Diagramm 510
 Vereinfachte Datumseingabe 139
 Vergleichs-Operatoren 899
 Verkettungs-Operatoren 902
 Verlinkte Bilder löschen 423
 Verlinkte OLE-Objekte löschen 423
 verschachtelte Menüs 315
 Verschieben von Dateien 715
 Verschieben von Verzeichnissen 716
 Verschiedene Konstanten 916
 Version 302
 Versionsnummer 584
 Verteilerliste 769
 Verweis 653, 878
 Verweis setzen 781
 Verweise 734, 818
 aktivieren 788
 deaktivieren 789
 Verzeichnis
 erstellen 727
 Größe ermitteln 728
 löschen 728
 verschieben 716
 wechseln 728
 Verzeichnisnamen auslesen 723
 Visible 584
 VisibleDropDown 206
 Volatile 69
 VolumeName 731
 Vorwärtstrend 470

W

Wahrheitswert 261
 Wait 536
 Wait-Methode 186
 Walls 455
 Warnmeldung 38
 Webabfrage
 aktualisieren 684
 bereinigen 685
 erstellen 681
 löschen 687
 Webbrowser 404
 Webbrowser einbinden 688
 Web-Komponente 637
 Webseite 440

Wechseln eines Verzeichnisses 728
 Wechseln von Menüleisten 337
 Wechseln-Funktion 115
 WeekdayName 272
 Weight 456
 Weitersuchen 667
 Werkzeugsammlung 188, 651
 WholeStory 752
 Width 393
 Windows Meta File 413
 Windows-Auflistung 795
 Windows-Benutzername 114
 WithEvents 818, 873
 Wochentag 133
 Word 751
 WordArt-Formen 407
 WordArt-Katalog 407
 WordArt-Objekt
 erzeugen 406
 formatieren 407
 Workbook 521
 Aktivate 527
 BeforeSave 537
 Close 539
 Deaktiviere 527
 Kill 658
 NewSheet 543, 573
 Open 135, 183, 525, 536
 SheetActivate 549
 SheetSelectionChange 169
 Workbook-Ereignisse 521
 AddinInstall 843
 AddinUninstall 844
 Übersicht 523
 Worksheet 521
 Aktivate 558
 BeforeRightClick 551
 Change 137, 552, 561
 Clear 207
 FollowHyperlink 565
 WorksheetFunction 59
 WorksheetFunction-Objekt 147
 Write 739
 WriteBlankLines 739
 WriteLine 738f.

X

xlAutomatic 474
 xlCalculationAutomatic 195
 xlCustom 474
 xlDefault 556
 xlErrValue 77
 xlManual 195

xlMaximum 474
xlMinimum 474
xlPicture 384
xlScreen 384
xlSheetVeryHidden 136, 532
XLStart-Ordner 519, 572
xlWait 556
XML 689
XML-Liste 692
XML-SS-Schema 696
XSL(T) 698

Z

Zeichencode 378
Zeichentabelle 377
Zeichnungsobjekte entfernen 414
Zeilenumbruch 585
Zeilenumbruch in TextBox 284
Zeit messen 183
Zeitbegrenzte Meldung 188
Zeitgesteuerte Makros 574
Zeitumstellung 158
Zeitwert formatieren 131
Zelle 67
ZELLE.ZUORDNEN 153
Zelle-Kontextmenü deaktivieren 324
Zellen, verbinden 197
Zellen formatieren 98
Zellen sperren 86
Zellenbereich vergrößern 555
Zellen-DropDown 216
Zellenformat 129
Zellenkontextmenü verhindern 551
Zellenwert kumulieren 84
Zellfehlerwerte 73
Zellpositions-Änderungen anzeigen 827
Zirkelbezüge berechnen 81
Zoomfaktor 572
 über Bildschirmeinstellung 571
Zufallsgenerator 857
Zufallszahl 857
Zufallszahlen 849
Zugriffsinformationen 718
Zugriffstaste 640
Zwischenablage 583, 757, 781, 783, 876, 917



Copyright

Daten, Texte, Design und Grafiken dieses eBooks, sowie die eventuell angebotenen eBook-Zusatzdaten sind urheberrechtlich geschützt.

Dieses eBook stellen wir lediglich als **Einzelplatz-Lizenz** zur Verfügung!

Jede andere Verwendung dieses eBooks oder zugehöriger Materialien und Informationen, einschliesslich der Reproduktion, der Weitergabe, des Weitervertriebs, der Platzierung im Internet, in Intranets, in Extranets anderen Websites, der Veränderung, des Weiterverkaufs und der Veröffentlichung bedarf der schriftlichen Genehmigung des Verlags.

Bei Fragen zu diesem Thema wenden Sie sich bitte an:

<mailto:info@pearson.de>

Zusatzdaten

Möglicherweise liegt dem gedruckten Buch eine CD-ROM mit Zusatzdaten bei. Die Zurverfügungstellung dieser Daten auf der Website ist eine freiwillige Leistung des Verlags. Der Rechtsweg ist ausgeschlossen.

Hinweis

Dieses und andere eBooks können Sie rund um die Uhr und legal auf unserer Website



(<http://www.informit.de>)

herunterladen