

Cindy Meister, Thomas Gahler, Peter Jamieson, Christian Freßdorf

## **Microsoft Word Programmierung – Das Handbuch**



Cindy Meister, Thomas Gahler, Peter Jamieson, Christian Freßdorf

# Microsoft Word- Programmierung – Das Handbuch

**Microsoft**<sup>®</sup>  
*Press*

Cindy Meister, Thomas Gahler, Peter Jamieson, Christian Freßdorf: Microsoft Word-  
Programmierung – Das Handbuch  
Microsoft Press Deutschland, Konrad-Zuse-Str. 1, D-85716 Unterschleißheim  
Copyright © 2006 by Microsoft Press Deutschland

Das in diesem Buch enthaltene Programmmaterial ist mit keiner Verpflichtung oder Garantie irgendeiner Art verbunden. Autor, Übersetzer und der Verlag übernehmen folglich keine Verantwortung und werden keine daraus folgende oder sonstige Haftung übernehmen, die auf irgendeine Art aus der Benutzung dieses Programmmaterials oder Teilen davon entsteht.

Das Werk einschließlich aller Teile ist urheberrechtlich geschützt. Jede Verwertung außerhalb der engen Grenzen des Urheberrechtsgesetzes ist ohne Zustimmung des Verlags unzulässig und strafbar. Das gilt insbesondere für Vervielfältigungen, Übersetzungen, Mikroverfilmungen und die Einspeicherung und Verarbeitung in elektronischen Systemen.

Die in den Beispielen verwendeten Namen von Firmen, Organisationen, Produkten, Domänen, Personen, Orten, Ereignissen sowie E-Mail-Adressen und Logos sind frei erfunden, soweit nichts anderes angegeben ist. Jede Ähnlichkeit mit tatsächlichen Firmen, Organisationen, Produkten, Domänen, Personen, Orten, Ereignissen, E-Mail-Adressen und Logos ist rein zufällig.

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1  
08 07 06

ISBN 3-86063-989-7

© Microsoft Press Deutschland  
(ein Unternehmensbereich der Microsoft Deutschland GmbH)  
Konrad-Zuse-Str. 1, D-85716 Unterschleißheim  
Alle Rechte vorbehalten

Fachlektorat: Georg Weiherer, Münzenberg  
Korrektorat: Karin Baeyens, Dorothee Klein, Siegen  
Layout und Satz: Gerhard Alfes, mediaService, Siegen ([www.media-service.tv](http://www.media-service.tv))  
Umschlaggestaltung: Hommer Design GmbH, Haar ([www.HommerDesign.com](http://www.HommerDesign.com))  
Gesamtherstellung: Kösel, Krugzell ([www.KoeselBuch.de](http://www.KoeselBuch.de))



# Übersicht

<b>Teil A</b>	
<b>Einführung in VBA</b>	27
1 Word-Makros	29
2 Der Visual Basic-Editor	53
3 VBA-Grundlagen	77
4 Windows-APIs in VBA nutzen	137
<b>Teil B</b>	
<b>Das Objektmodell von Word</b>	171
5 Überblick über die Arbeit mit Objekten	173
6 Das Objektmodell von Word	187
7 Ereignisse in Word	431
<b>Teil C</b>	
<b>Steuern und gesteuert werden</b>	463
8 Grundlagen der Fernsteuerung	465
9 Word von anderen Umgebungen aus steuern	481
10 Andere Programme aus Word heraus steuern	509
11 Eingebettete OLE-Objekte	539
<b>Teil D</b>	
<b>Optimierung der Benutzerschnittstelle</b>	573
12 Anwendungsoptionen	575
13 Speicherort von Anpassungen	599
14 Mit Dialogfeldern arbeiten	607
15 Menüs und Symbolleisten	649

16	Tastaturbelegungen .....	665
17	Aufgabenbereiche .....	683
18	Interne Word-Befehle übersteuern .....	707
19	Zugriff auf den Visual Basic-Editor (VBE) .....	713

## **Teil E**

### **Praktische Anwendungsbeispiele .....**

20	Kalenderblatt erstellen .....	761
21	Aktive Symbolleisten .....	771
22	Aktualisierung der Felder mit einem Fortschrittsbalken visualisieren .....	779
23	Allerlei in Kopf- und Fußzeilen .....	799
24	Grafiken mit Beschriftungen .....	821
25	Rechtschreibfehler für den Ausdruck formatieren .....	831
26	Rechnung erstellen mit Formularfeldern .....	839
27	Attraktive Tischkarten im Nu .....	849

## **Teil F**

### **XML und Smart-Technologien .....**

28	Word 2003 und XML: Eine Einführung .....	861
29	Smarttags .....	911
30	Smart Documents .....	943

## **Teil G**

### **Anhang .....**

A	Namenskonventionen für Word-VBA .....	999
B	Allgemeines zum Thema Feldfunktionen .....	1003
C	Bezeichnungen von Symbolleisten .....	1011
D	Startoptionen von Word .....	1021
E	Bei Problemen – Word zurücksetzen .....	1025

<b>F</b>	Nützliche Links ins Internet .....	1029
<b>G</b>	Inhalt der CD-ROM .....	1033
<b>H</b>	Über die Autoren .....	1039
	Verzeichnis zum Objektmodell .....	1043
	Stichwortverzeichnis .....	1049



# Inhaltsverzeichnis

Vorwort .....	21
Für wen wurde das Buch geschrieben? .....	22
Word-Versionen .....	23
Wie ist dieses Buch aufgebaut? .....	23
Danksagung .....	24
 <b>Teil A</b>	
<b>Einführung in VBA .....</b>	<b>27</b>
 <b>1 Word-Makros .....</b>	<b>29</b>
Aller Anfang ist schwer .....	30
Den Makrorekorder einsetzen .....	31
Den aufgezeichneten Code bearbeiten .....	33
Wo können Makros gespeichert werden? .....	37
Makros in der Benutzerschnittstelle einbinden und verwalten .....	38
Makro einer Symbolleiste-Schaltfläche zuweisen .....	39
Makros kopieren .....	42
Makrosicherheit .....	43
Sicherheitsstufe anpassen .....	43
Signatur mittels <i>selfcert.exe</i> erstellen .....	47
VBA-Projekte mit einer Signatur versehen .....	48
Zertifikat zur Signatur erstellen .....	48
Zertifikat einlesen .....	49
Zusammenfassung .....	51
 <b>2 Der Visual Basic-Editor .....</b>	<b>53</b>
Der Blick ins Innere: Die Fenster .....	54
Der Projekt-Explorer .....	56
Das Eigenschaftsfenster .....	57
Das Code-Fenster .....	59
Die Symbolleiste .....	60
Hinter den Kulissen: Die Optionen .....	61
Optionen für den Editor .....	61
Optionen für das Editorformat .....	63
Allgemeine Optionen .....	65
Die VBA-Hilfe – eine versteckte Schatzkammer .....	66
Wo alle Fäden zusammenlaufen: Der Objektkatalog .....	69
The Show must go on: Code bearbeiten .....	73
Zusammenfassung .....	75

<b>3</b>	<b>VBA-Grundlagen</b>	<b>77</b>
	Variablen	78
	Standard-Datentypen	79
	Gültigkeit bzw. Sichtbarkeit	81
	Umwandlung von Datentypen	84
	Weiterreichen von Variablen an Prozeduren	85
	Variablen in Datenfeldern ablegen	88
	Konstanten	91
	Benutzerdefinierte Typen	93
	Type-Anweisung	93
	Enum-Anweisung	96
	Nützliche VBA-Funktionen	100
	Bedingungen	105
	Schleifen	108
	Code im VB-Editor debuggen	111
	Der Überwachungsbereich	114
	Fehlerbehandlung	118
	Dateisystem-Operationen	125
	Alle Dateien eines Verzeichnisses auflisten	125
	Verzeichnisname mit Backslash ergänzen	127
	Prüfen, ob eine bestimmte Datei vorhanden ist	128
	Prüfen, ob ein bestimmter Ordner vorhanden ist	129
	Prüfen, ob eine Datei von jemanden im Zugriff ist	131
	Datei löschen	131
	Letztes Speicherdatum einer Datei ermitteln	133
	Größe einer Datei ermitteln	134
	Zusammenfassung	136
<b>4</b>	<b>Windows-APIs in VBA nutzen</b>	<b>137</b>
	Aufbau der API-Funktionen	138
	Kombinieren und Abschließen von Pfaden	139
	Abschließen eines Pfades	140
	Kombinieren von Pfaden	141
	Dateinamen an Pfad anhängen	142
	Dateierweiterung an Datei anhängen	142
	Datei über die Dateiendung ausführen	143
	Zugriff auf Registry-Einträge	147
	Ermitteln von Registry-Einträgen und Werten	147
	Zugriff auf INI-Dateien	152
	Auslesen und Schreiben von Abschnitten	153
	Ermitteln aller Abschnitte einer INI-Datei	155
	Name des angemeldeten Benutzers ermitteln	157
	Verarbeitung für eine bestimmte Zeit unterbrechen	159
	Wave-Datei abspielen	160
	Tastenstatus abfragen	163
	Zusammenfassung	168

**Teil B**

<b>Das Objektmodell von Word</b> .....	<b>171</b>
<b>5 Überblick über die Arbeit mit Objekten</b> .....	<b>173</b>
Arbeiten mit Objekten .....	174
Das gesuchte Objekt aufspüren .....	174
Objekte als Variablen .....	175
Auflistung von Objekten .....	179
Auflistung bearbeiten .....	180
Selection – das verpönte Objekt .....	183
Auto-Makros als Pseudo-Ereignisse .....	184
Zusammenfassung .....	185
<b>6 Das Objektmodell von Word</b> .....	<b>187</b>
Das <i>System</i> -Objekt .....	189
Die Anwendung: Das <i>Application</i> -Objekt .....	191
Die gegenwärtige Markierung: <i>Selection</i> und ähnliche Objekte .....	200
Der Kern der Sache: Das <i>Document</i> -Objekt .....	202
Dokumentvorlagen: Das <i>Template</i> -Objekt .....	214
Mit Bereichen arbeiten: Das <i>Range</i> -Objekt .....	219
Einen Bereich definieren .....	219
Einen Bereich bearbeiten .....	221
Wo befindet sich der Bereich? .....	226
Text aus einem Bereich lesen .....	229
Eine Formel berechnen .....	230
Die Nadel im Heuhaufen: <i>Find/Replace</i> einsetzen .....	233
Vor- und Nachteile des Makrorekorder-Resultats .....	233
Anpassung des aufgezeichneten Codes .....	236
Bekannte Probleme vermeiden oder beheben .....	250
Formatieren mit Stil: Das <i>Style</i> -Objekt .....	254
Formatvorlagen .....	255
Benutzerschnittstellen für Formatvorlagen .....	256
Formatierungseinschränkungen .....	257
Word-interne Formatvorlagen .....	262
Formatvorlagen erstellen und modifizieren .....	265
»Wilde« Formatvorlagen .....	269
Zielscheibe Textmarke: Das <i>Bookmark</i> -Objekt .....	271
Inhalt mit Tabellen strukturiert darstellen .....	280
Tabellen erstellen .....	281
Tabellen formatieren .....	287
Informationen aus Tabellen holen .....	300
Automatische Nummerierung mit Listen .....	309
Listeigenschaften ermitteln .....	310
Listvorlagen ( <i>ListTemplates</i> ) .....	312
Feldfunktionen .....	320
Verknüpfungen, Tabellen und Berechnungen .....	328

Grafiken: Die <i>InlineShape</i> - und <i>Shape</i> -Objekte .....	334
Grafiken einfügen .....	334
Verknüpfungen erstellen und verwalten .....	340
Layout-Optionen .....	343
Zeichnungsobjekte (AutoFormen) .....	354
Abschnitte im Dokument: Das <i>Section</i> -Objekt .....	357
Bereiche im Dokument: Das <i>StoryRanges</i> -Objekt .....	359
Die Seite definieren: Das <i>PageSetup</i> -Objekt .....	363
Die Seite gestalten: Das <i>HeaderFooter</i> -Objekt .....	371
Lange Dokumente .....	377
Zentraldokumente .....	377
Dokumente verknüpfen .....	380
Formulare: Das <i>FormField</i> -Objekt .....	381
Der Seriendruck: Das <i>MailMerge</i> -Objekt .....	388
Wie <i>OpenDataSource</i> funktioniert .....	402
Datenverbindungen unter Word 2000, 2002 sowie 2003 .....	403
Zusammenfassung .....	429
 7 Ereignisse in Word .....	 431
Auto-Makros .....	432
Ereignisse auf Dokumentebene .....	434
Ereignisse auf Applikationsebene .....	436
Klassenmodul einrichten und initialisieren .....	438
Klassenmodul einrichten .....	438
Klassenmodul initialisieren .....	440
Übersicht über die verfügbaren Ereignisse .....	441
WindowActivate .....	441
WindowDeactivate .....	442
WindowSize .....	442
WindowBeforeDoubleClick .....	443
WindowBeforeRightClick .....	444
WindowSelectionChange .....	446
NewDocument-Ereignis .....	448
DocumentOpen .....	449
DocumentChange .....	449
DocumentBeforeClose .....	450
DocumentBeforeSave .....	451
DocumentBeforePrint .....	452
DocumentSync .....	454
Seriendruckereignisse .....	454
MailMergeBeforeMerge .....	455
MailMergeBeforeRecordMerge .....	455
MailMergeAfterRecordMerge .....	456
MailMergeAfterMerge .....	456
Beispiel: 1:n-Liste im Seriendruckresultat .....	456
Zusammenfassung .....	461



<b>Teil C</b>	
<b>Steuern und gesteuert werden</b>	<b>463</b>
<b>8 Grundlagen der Fernsteuerung</b>	<b>465</b>
Verweise auf externe Bibliotheken	466
Early versus Late Binding	470
Vorteile von Early Binding	471
Vorteile von Late Binding	473
Ganz clever, wer beide Arten kombiniert	474
Über das .NET Framework	477
Zusammenfassung	479
<b>9 Word von anderen Umgebungen aus steuern</b>	<b>481</b>
Fernsteuern von Microsoft Word	482
Makros von außen anstoßen	482
Word effektiv fernsteuern	486
Versteckte Word-Instanz steuern	488
Programmzeilen zentral verwalten und gemeinsam nutzen	490
Dokumentvorlagen (.dot) als Add-In	490
Add-In-Prozeduren in anderen Umgebungen nutzen	493
Fernsteuerung aus Office-fremden Umgebungen	495
VSTO	495
Zusammenfassung	508
<b>10 Andere Programme aus Word heraus steuern</b>	<b>509</b>
Fernsteuern von Microsoft Excel	510
Versteckte Excel-Instanz steuern	511
Berechnungen von Excel erledigen lassen	512
Fernsteuern von Microsoft PowerPoint	513
Versteckte PowerPoint-Instanz steuern	513
Text der Präsentation als Inhaltsstruktur in ein Dokument übernehmen	515
Fernsteuern von Microsoft Visio	518
Versteckte Visio-Instanz steuern	518
Fernsteuern von Microsoft Outlook	520
Versteckte Outlook-Instanz steuern	520
Kontakt als Empfängeradresse im Brief nutzen	521
Die aktuelle Datei über Outlook versenden	526
Fernsteuern von Microsoft Access	528
Auf Datenbanken zugreifen	530
Access-Datenbank ansprechen	530
Excel-Tabelle ansprechen	536
Zusammenfassung	538

<b>11</b>	<b>Eingebettete OLE-Objekte</b>	<b>539</b>
	Excel-Tabellenobjekte	543
	Excel-Diagramme	550
	MS Graph-Diagramme	555
	WordArt	559
	ActiveX-Steuerelemente	564
	Das Navigieren	566
	Datengültigkeit prüfen	568
	Steuerelement-Referenzen in VBA-Code	571
	Zusammenfassung	572
	<b>Teil D</b>	
	<b>Optimierung der Benutzerschnittstelle</b>	<b>573</b>
<b>12</b>	<b>Anwendungsoptionen</b>	<b>575</b>
	Dokumentvorlage <i>Normal.dot</i> konfigurieren	576
	Vorlage »UrNormal.dot« erstellen	576
	Benutzereinstellungen abspeichern	586
	<i>SaveSetting</i> -Anweisung	588
	<i>GetSetting</i> -Funktion	588
	<i>PrivateProfileString</i> -Eigenschaft	589
	Dokumenteinstellungen abspeichern	593
	Dokumenteigenschaften bearbeiten	595
	Dokumentvariablen bearbeiten	597
	Zusammenfassung	598
<b>13</b>	<b>Speicherort von Anpassungen</b>	<b>599</b>
	Der Speicherort einer Anpassung	600
	Auswahl des Speicherorts	603
	Kontext für COM-Add-Ins	603
	Hierarchie der Anpassungen	604
	Zusammenfassung	605
<b>14</b>	<b>Mit Dialogfeldern arbeiten</b>	<b>607</b>
	Benutzerdefinierte Dialogfelder	608
	UserForm in verschiedenen Projekten nutzen	609
	Das <i>UserForm</i> -Objekt	610
	Steuerelemente einbinden	618
	Steuerelemente und ihre Besonderheiten	621
	Anforderungen an ein Dialogfeld	624
	Dialogfelder zur Laufzeit beeinflussen	626
	Interne Dialogfelder	631
	Dialogfelder »anzeigen«, »anzeigen und ausführen« oder »ausführen«	632
	Dialogfelder konfigurieren, vorbelegen und auswerten	634
	Übersicht über die in Word enthaltenen Dialogfelder	636

<i>FileDialog</i> -Objekt .....	638
Übersicht über die verschiedenen <i>FileDialog</i> -Typen .....	638
Dialogfelder »anzeigen« oder »anzeigen und ausführen« .....	640
Definieren von Dateiauswahlfilter .....	640
Anwendung des <i>FileDialog</i> -Typs <i>msoFileDialogOpen</i> .....	642
Anwendung des <i>FileDialog</i> -Typs <i>msoFileDialogSaveAs</i> .....	644
Anwendung des <i>FileDialog</i> -Typs <i>msoFileDialogFilePicker</i> .....	645
Anwendung des <i>FileDialog</i> -Typs <i>msoFileDialogFolderPicker</i> .....	646
Zusammenfassung .....	648
<b>15 Menüs und Symbolleisten .....</b>	<b>649</b>
CommandBars .....	650
Symbolschaltflächen .....	655
Symbolschaltflächen erstellen .....	656
Bearbeitungsfelder und Dropdownlisten .....	660
Symbolleisten erstellen .....	661
Zusammenfassung .....	664
<b>16 Tastaturbelegungen .....</b>	<b>665</b>
Tastenbelegungen im Objektmodell .....	667
Bestehende Tastenbelegungen ermitteln .....	669
Tastenbelegungen eines Befehls ermitteln .....	674
Tastenkombinationen verwalten .....	675
Tastenbelegungen entfernen .....	675
Tastenkombinationen definieren .....	676
Tastenbelegung mit COM-Add-In verbinden .....	677
Ein Beispiel .....	677
Das Beispiel installieren .....	679
Zusammenfassung .....	681
<b>17 Aufgabenbereiche .....</b>	<b>683</b>
Allgemeines zum Aufgabenbereich .....	
oder, was steckt dahinter .....	684
Objektmodell-Schnittstellen für Work Panes .....	685
Registry-Einträge .....	686
Fehlermeldungen .....	687
Der programmtechnische Umgang mit Aufgabenbereichen (Task Panes) .....	688
Allgemeine VBA-Schnittstellen für die Arbeit mit Aufgabenbereichen .....	691
Bestimmte Aufgabenbereiche manipulieren .....	698
Zusammenfassung .....	706
<b>18 Interne Word-Befehle übersteuern .....</b>	<b>707</b>
Word-Befehl außer Kraft setzen .....	708
Zusammenfassung .....	711

<b>19</b>	<b>Zugriff auf den Visual Basic-Editor (VBE)</b>	<b>713</b>
	Notwendige Verweise und Sicherheitseinstellungen	714
	Der Visual Basic-Editor	715
	Die VBA-Projekte	717
	Übersicht über alle VBA-Projekte	718
	Das aktive VBA-Projekt	719
	Zugriff auf die in einem Projekt enthaltenen Komponenten	721
	Auslesen des VBA-Codes von Komponenten	724
	Allgemeine Zugriffe auf die Code-Zeilen	724
	Zugriff auf den Deklarationsbereich	725
	Zugriff auf die Code-Zeilen einzelner Prozeduren	726
	Auflisten und Durchsuchen aller Projekte	729
	Ersetzen und Entfernen von VBA-Code-Zeilen	731
	Hinzufügen von Komponenten zu einem Projekt	734
	Eine vorhanden Komponente in ein Projekt importieren	735
	Eine neue Komponente einem Projekt hinzufügen	737
	Entfernen von Komponenten aus einem Projekt	745
	Anzeigen von dynamisch erzeugten UserForms	746
	Verweise auf Bibliotheken und Dateien ermitteln und zur Laufzeit setzen	749
	Übersicht über die gesetzten Verweise	749
	Neuen Verweis setzen	751
	Ungültige Verweise korrigieren bzw. entfernen	753
	Zusammenfassung	757
	<b>Teil E</b>	
	<b>Praktische Anwendungsbeispiele</b>	<b>759</b>
<b>20</b>	<b>Kalenderblatt erstellen</b>	<b>761</b>
	Die Aufgabenstellung	762
	Diskussion zum Code	764
	Zusammenfassung	769
<b>21</b>	<b>Aktive Symbolleisten</b>	<b>771</b>
	Symbolleisten mit dynamischen Schaltflächen	772
	Zusammenspiel der einzelnen Module	773
	Diskussion zum Code	774
	Zusammenfassung	777
<b>22</b>	<b>Aktualisierung der Felder mit einem Fortschrittsbalken visualisieren</b>	<b>779</b>
	Aktualisieren aller Felder eines Dokumentes	780
	Das StoryRange-Objekt	781
	Felder in den StoryRange-Objekten aktualisieren	787
	Textfelder in Kopf- und Fußzeilen ansprechen	788
	Durchlaufen aller Dokumentkomponenten	789

Erstellen und Einbinden eines Fortschrittsbalkens .....	791
Einbinden des Microsoft ProgressBar-Controls .....	795
Zusammenfassung .....	796
<b>23 Allerlei in Kopf- und Fußzeilen .....</b>	<b>799</b>
Seitennummer eintragen .....	801
Firmenlogo einfügen .....	803
Wasserzeichen einfügen .....	806
Falz- und Lochmarke setzen .....	810
Absenderadresse eintragen .....	813
Dateiname und Dokumentdatum setzen .....	815
Zusammenfassung .....	819
<b>24 Grafiken mit Beschriftungen .....</b>	<b>821</b>
Wissenschaftliche Figuren beschriften .....	822
Beschriftung und Grafik im Positionsrahmen .....	826
Zusammenfassung .....	829
<b>25 Rechtschreibfehler für den Ausdruck formatieren .....</b>	<b>831</b>
Wie die Lösung funktioniert .....	833
Zusammenfassung .....	838
<b>26 Rechnung erstellen mit Formularfeldern .....</b>	<b>839</b>
Formular zur Bestätigung von Bestellungen .....	840
Zusammenfassung .....	848
<b>27 Attraktive Tischkarten im Nu .....</b>	<b>849</b>
Das Seriendruck-Hauptdokument .....	850
Die WordArt-Objekt-Vorlage .....	851
Die Tischkarten erstellen .....	853
Zusammenfassung .....	857
<b>Teil F</b>	
<b>XML und Smart-Technologien .....</b>	<b>859</b>
<b>28 Word 2003 und XML: Eine Einführung .....</b>	<b>861</b>
Was ist XML und wozu dient es? .....	862
XML-Vokabulare .....	864
XML-Daten transformieren .....	866
Welche Aufgabe erfüllt XML in Word? .....	868
XML-Bestandteile .....	869
XML-Parser .....	870
XML-Elemente, -Tags, -Inhalt und -Attribute .....	870

Fehlende sowie mehrfach vorkommende Werte .....	871
Elemente oder Attribute? .....	872
XML-Dokumente und Deklarationen .....	872
XML-Dokumentinstanz, Markup und Inhalt .....	873
Wohlgeformte und gültige XML-Dokumente .....	874
XML-Schema-Definitionen .....	875
XML-Namensräume und Schemas .....	878
XML-Daten manipulieren .....	885
Das XML-Dokument-Objektmodell (DOM) .....	885
XSLT .....	886
XML in Word 2003 .....	890
WordProcessingML .....	890
Benutzerdefiniertes XML .....	894
Die Word-Schemabibliothek .....	896
Transformationen und Lösungen (Solutions) .....	901
Schemas und Transformationen im Word-Objektmodell .....	907
WordProcessingML außerhalb von Word generieren .....	908
Zusammenfassung .....	909
 <b>29 Smarttags .....</b>	 <b>911</b>
Die Arbeit mit Smarttags .....	912
Smarttags aus dem Blinkwinkel des Benutzers .....	912
Smarttag »Recognizers« und »Actions« .....	914
Smarttag-Optionen und Ausnahme-Listen .....	915
Die Entwicklung von Smarttags .....	916
Unterschiede zwischen den Office-Versionen .....	917
MOSTL-Smarttags entwickeln .....	917
Ein Smarttag mit VB6 entwickeln .....	924
Smarttags im Word-Objektmodell .....	932
Smarttag-Objekte in Word-VBA verwenden .....	936
Zusammenfassung .....	941
 <b>30 Smart Documents .....</b>	 <b>943</b>
Was ist ein Smart Document? .....	944
Ein MOSTL-Smart Document .....	945
Das MOSTL-Beispiel installieren .....	946
Eine benutzerfreundlichere Lösung .....	952
Wie es funktioniert .....	955
Eine Smart Document-DLL .....	966
Mit VB6 eine Smart Document-DLL erstellen .....	967
Das DLL-Beispiel installieren .....	968
Wie es funktioniert: das <i>SmartDocument</i> -Interface .....	973
Die DLL in VB6 erstellen .....	993
Das Erweiterungspaket-Manifest digital signieren .....	994
Zusammenfassung .....	995

**Teil G**

<b>Anhang</b> .....	<b>997</b>
<b>A Namenskonventionen für Word-VBA</b> .....	<b>999</b>
Variablen .....	<b>1000</b>
Benutzerdefiniertes Dialogfeld .....	<b>1000</b>
Entwicklungsumgebung .....	<b>1001</b>
Word-Objekte .....	<b>1001</b>
<b>B Allgemeines zum Thema Feldfunktionen</b> .....	<b>1003</b>
<b>C Bezeichnungen von Symbolleisten</b> .....	<b>1011</b>
<b>D Startoptionen von Word</b> .....	<b>1021</b>
Die Befehlszeilenargumente von Word .....	<b>1022</b>
Zusammenfassung .....	<b>1023</b>
<b>E Bei Problemen – Word zurücksetzen</b> .....	<b>1025</b>
<b>F Nützliche Links ins Internet</b> .....	<b>1029</b>
<b>G Inhalt der CD-ROM</b> .....	<b>1033</b>
Die Beispieldateien der CD-ROM zum Buch .....	<b>1034</b>
Zusatz-Software: Die Dateien im Anhang .....	<b>1036</b>
<b>H Über die Autoren</b> .....	<b>1039</b>
Verzeichnis zum Objektmodell .....	<b>1043</b>
Stichwortverzeichnis .....	<b>1049</b>





# Vorwort

Mitte der 90er Jahre veröffentlichte Microsoft das »Word Developer Kit« für die Word for Windows-Versionen 6.0 und 95. Dieses Standard-Werk für damalige Word-Entwickler veranschaulichte den Umgang mit WordBasic sowie der Benutzerschnittstelle und erklärte, wie mit dem Anwender kommuniziert wird. Bis zum heutigen Tag steht leider kein entsprechendes übersichtliches Nachschlagewerk dieser Art für die veröffentlichten VBA-Versionen aktueller Word-Versionen zur Verfügung, das sich eingehend mit dem Objektmodell von Word beschäftigt. Denn Microsoft Word hat seit den Versionen 6.0 und 95 seinen Funktionsumfang mächtig erweitert und lässt fast keine Bedürfnisse im Zusammenhang mit Textverarbeitung offen.

Auch um diese Lücke zu schließen, wurde das vorliegende Buch geschrieben.

Eine weitere Motivation waren die in den verschiedenen Newsgroups auf microsoft.com vermehrt auftretenden Fragen von .NET- und C#-Entwicklern zur Anbindung und Fernsteuerung von Word aus ihren Anwendungen heraus.

Damit ein Fachbuch wie das vorliegende aber entstehen kann, bedarf es eines gewissen Maßes an Fachwissen, das sich einige der Autoren in ihrem gut fünfzehnjährigen professionellen Umgang mit Microsoft Word angeeignet haben. Dabei spielt das Interesse an der Vermittlung dieses Wissens an Einsteiger und andere Interessierte eine große Rolle, denn nur so können die Inhalte und Beispiele praxisnah und zielorientiert behandelt werden.

## Für wen wurde das Buch geschrieben?

Diese Frage ist nicht einfach zu beantworten, da sie von bereits existierenden Grundkenntnissen in Word (VBA) und den persönlichen Zielen abhängt.

So wird Word von unterschiedlichen Kategorien von Anwendern genutzt, und diese erwarten die verschiedensten Funktionalitäten. Der eine freut sich über eine »Super-Schreibmaschine«, der kaufmännische Sachbearbeiter interessiert sich für Berichte und der Redakteur will damit umfangreiche Handbücher verwalten. In einer stillen Ecke sitzt der Autor, der darin das Werkzeug sieht, um seinen nächsten Bestseller zu verfassen. Dass eine einzige Anwendung alle diese Erwartungen erfüllen kann, ist bemerkenswert. Zugegeben, einige Aufgaben sind mit den Bordmitteln von Microsoft Word leichter zu realisieren als andere. Die entsprechenden Werkzeuge sind jedoch vorhanden, und als Entwickler ist es unsere Aufgabe, die Vorgänge für den Benutzer zu entflechten, zu vereinfachen und so zu erklären, dass er effizienter ans Ziel gelangt, ohne sich mit den internen Einzelheiten von Word auseinander setzen zu müssen.

Aus diesen Überlegungen heraus richtet sich das vorliegende Buch in erster Linie an folgende Anwendergruppen, die mit Word arbeiten und dessen Möglichkeiten nicht nur oberflächlich ausreizen möchten:

- Anwender, die Makros nicht nur mit dem integrierten Makrorekorder aufzeichnen, sondern selbstständig erstellen und bereits angeeignete Kenntnisse vertiefen möchten.
- VBA-Programmierer, die professionelle Lösungen entwickeln und neben dem grundlegenden Wissen vertiefende Informationen zur Basis erhalten möchten.
- Alle Programmierer (inkl. Microsoft .NET), die aus einer eigenen Applikation heraus auf Word zugreifen und dieses Programm automatisieren möchten oder müssen.

Das Buch richtet sich jedoch nicht an die Anwender, die noch keine Erfahrung mit Word oder VBA (oder einer anderen Programmiersprache) haben. Denn das vorliegende Buch beinhaltet keine Anleitung zum Einstieg in VBA; für diesen Bereich sind bereits verschiedene Bücher erschienen.

## Word-Versionen

Alle Angaben zu diesem Buch basieren auf Word 2003, der zurzeit aktuellen Version von Word. Abgesehen von den neuen Funktionen, die nur in dieser Version enthalten sind, gelten die meisten Angaben auch für die beiden Vorgängerversionen Word 2002 und Word 2000. Auf eventuelle, den Autoren bekannte Unterschiede zu diesen beiden Versionen wird zum Teil explizit hingewiesen.

Die grundlegenden Aussagen dieses Werks können auch auf Word 97 angewendet werden. Hier gilt es jedoch zu beachten, dass unterdessen bereits die vierte Nachfolgeversion von Word im Einsatz ist und deshalb viele Funktionen damals noch nicht zur Verfügung standen. Dies gilt unter anderem auch für VBA, das damals in der Version 5 ausgeliefert wurde (alle späteren Word-Versionen enthalten VBA 6.0).

### Die Zukunft von VBA

Die Rückwärtskompatibilität von VBA ist für die nächste Version von Word (»Office 12«) weiterhin gewährleistet. Somit wird das aus dem vorliegenden Buch Erlernte auch für diese Version seine Gültigkeit haben. VBA wird gemäß den aktuellen Informationen in den nächsten zwei Programmversionen auf jeden Fall noch unterstützt. Dies, obwohl XML eine immer größere und wichtigere Rolle einnehmen wird.

## Wie ist dieses Buch aufgebaut?

Dieses Buch besteht aus sechs Teilen, von denen jeder einem bestimmten Schwerpunkt gewidmet ist. Die einzelnen Teile bauen zwar aufeinander auf, stehen aber in keiner direkten Abhängigkeit zueinander. Somit ist gewährleistet, dass nicht alle Seiten gelesen werden müssen, um sich in ein spezifisches Thema zu vertiefen.

Der Inhalt der einzelnen Teile kurz zusammengefasst:

**Teil I** Eine Einführung in die Welt der Makros und VBA. Er zeigt das grundlegende Wissen zu den Möglichkeiten von Makros und stellt die eigentliche Programmierumgebung – den VB-Editor – vor. Eine Zusammenfassung zu den Grundlagen von VBA, erste allgemeine Beispiele und das Einbinden von Windows-API runden diesen Teil ab.

**Teil II** Gilt als Nachschlagewerk zum äußerst komplexen Objektmodell von Word. Es werden jeweils die wichtigsten Eigenschaften und Methoden zu den einzelnen Objekten detailliert aufgezeigt. Anhand von passenden Beispielen werden diese dem Leser näher gebracht. Dieser Teil enthält auch Beispiele in C#, welche die Schnittstellen zu den Word-PIAs veranschaulichen. Anhand dieser Kenntnisse sollte sich der .NET-Programmierer Zugang zum gesamten Objektmodell verschaffen können.

**Teil III** Zeigt die Grundlagen des Zusammenspiels mit anderen Applikationen auf. Dies ist unabhängig davon, ob Word aus anderen Applikationen heraus gesteuert wird oder ob Word selbst andere Applikationen steuert bzw. ob eingebundene Objekte in Word gesteuert werden. In diesem Teil befinden sich C#-Beispiele für die Automatisierung eingebetteter Objekte. Es werden auch die Möglichkeiten von VSTO kurz vorgestellt.

**Teil IV** Eine Zusammenfassung, die aufzeigt, auf welche unterschiedlichen Arten die Benutzerschnittstelle von Word angepasst werden kann. Dazu gehört unter anderem die Verwendung der internen Dialogfelder, das Erstellen von benutzerdefinierten Dialogfeldern und das Anpassen von Symbolleisten.

**Teil V** Eine Sammlung von praxisbezogenen Lösungsbeispielen. Diese Beispiele entstammen verschiedenen Bereichen und decken unterschiedliche Problematiken ab. Sie sollen einen vertieften Einblick in das Objektmodell bieten.

**Teil VI** Widmet sich dem Thema XML und zeigt dessen Einsatzgebiet und Möglichkeiten im Zusammenhang mit Word auf. Zu dieser Rubrik gehört auch eine Übersicht zu SmartTags und SmartDocument-Lösungen.

## Stichwortverzeichnisse

Am Ende des Buches befinden sich zwei Stichwortverzeichnisse mit unterschiedlichen Schwerpunkten:

- Das eigentliche **Stichwortverzeichnis** fasst den Inhalt nach einzelnen Themen und Inhalten zusammen. Es bietet so einen schnellen Zugriff auf die inhaltlichen Stellen im Text.
- Das **Verzeichnis zum Objektmodell** fasst alle Stellen im Dokument zusammen, die einen direkten Bezug auf ein Objekt sowie dessen Eigenschaften und Methoden haben. Somit ist ein schneller Zugriff auf die einzelnen Objekte innerhalb des Textes gewährleistet.

## CD-ROM als Beilage

Diesem Buch ist eine CD-ROM beigelegt. Diese enthält unter anderem alle Kapitel als *pdf*-Datei, damit Ihnen das Buch auch unterwegs als elektronisches Nachschlagewerk zur Verfügung steht.

Alle aufgeführten Programmsequenzen werden ebenfalls in Form von Beispieldateien mitgeliefert. Die Dateinamen werden jeweils am Ende eines Abschnitts oder Kapitels erwähnt. Die entsprechenden Dateien befinden sich auf der CD-ROM im Ordner `\Beispiele\KapXX` (wobei *XX* für die Nummer des entsprechenden Kapitels steht).

Neben diesen Beispielen sind noch weitere wertvolle Informationen auf dem Datenträger abgespeichert. Einen entsprechenden Hinweis finden Sie in den jeweiligen Kapiteln. Die Dateien befinden sich auf der CD-ROM im Ordner `\Beilagen`.

Im Buch wird an verschiedenen Stellen auf Informationsquellen im Internet verwiesen. Die entsprechende Internetadresse (URL) ist jeweils direkt im Text mit angegeben. Damit diese zum Teil recht langen und kryptischen Zeichenfolgen nicht manuell abgetippt werden müssen, sind diese Adressen in Form von einzelnen *url*-Dateien auf dem Datenträger abgespeichert. Diese Dateien befinden sich jeweils im Ordner `\Beispiele\KapXX\Internet`.

Weitere Informationen zur CD-ROM zum Buch sind im Anhang G zusammengefasst.

## Danksagung

An dieser Stelle möchten wir uns ganz besonders bei **Elisabeth Wilke-Thissen** bedanken. Sie hatte sich spontan zur Verfügung gestellt, um unsere Texte zu überarbeiten und von einer PC-deutschen bzw. einer »schweizerdeutschen« in eine deutsche Fassung zu bringen. Als wir ihre Zusage erhielten, wussten weder wir noch sie, was da alles auf sie zukam. Mit viel Energie hat sie sich fast jedem Kapitel des Manuskripts gewidmet und uns auf noch so kleine Unstimmigkeiten aufmerksam gemacht.

Lisa wurde im April 2003 in das MVP-Programm (Word) aufgenommen und ist Mit-Autorin des Buches »Microsoft Excel – die Expertentipps«.

Bedanken möchten wir uns auch bei unseren Helfern im Hintergrund. Bei **Klaus Linke** für seine Beilagen (*Interne Word-Befehle.doc*) und Hinweise zum Thema XML, bei **Steve Hudson** für die wertvollen Beiträge zum ListTemplate-Objekt und den Aufgabenbereichen, bei **Bill Coan** für den interessanten Beitrag zum Find-Objekt und bei allen anderen MVP-Kollegen generell für ihre gegenseitige Unterstützung bei Fragen zu den Produkten von Microsoft.

Im Weiteren möchten wir uns bei **Michael Greth** bedanken. Er hat uns einen Bereich auf seinem SharePoint Server zur Verfügung gestellt. Diese technische Unterstützung erlaubte uns einen einfachen Datenaustausch. Dies war unbedingt nötig, denn während des ganzen Projekts hat sich das Autorenteam kein einziges Mal getroffen.

Das Autorenteam, im November 2005

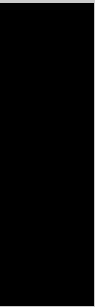


# Teil A

## Einführung in VBA

### In diesem Teil:

<b>Kapitel 1</b>	Word-Makros	29
<b>Kapitel 2</b>	Der Visual Basic-Editor	53
<b>Kapitel 3</b>	VBA-Grundlagen	77
<b>Kapitel 4</b>	Windows-APIs in VBA nutzen	137





# Kapitel 1

## Word-Makros

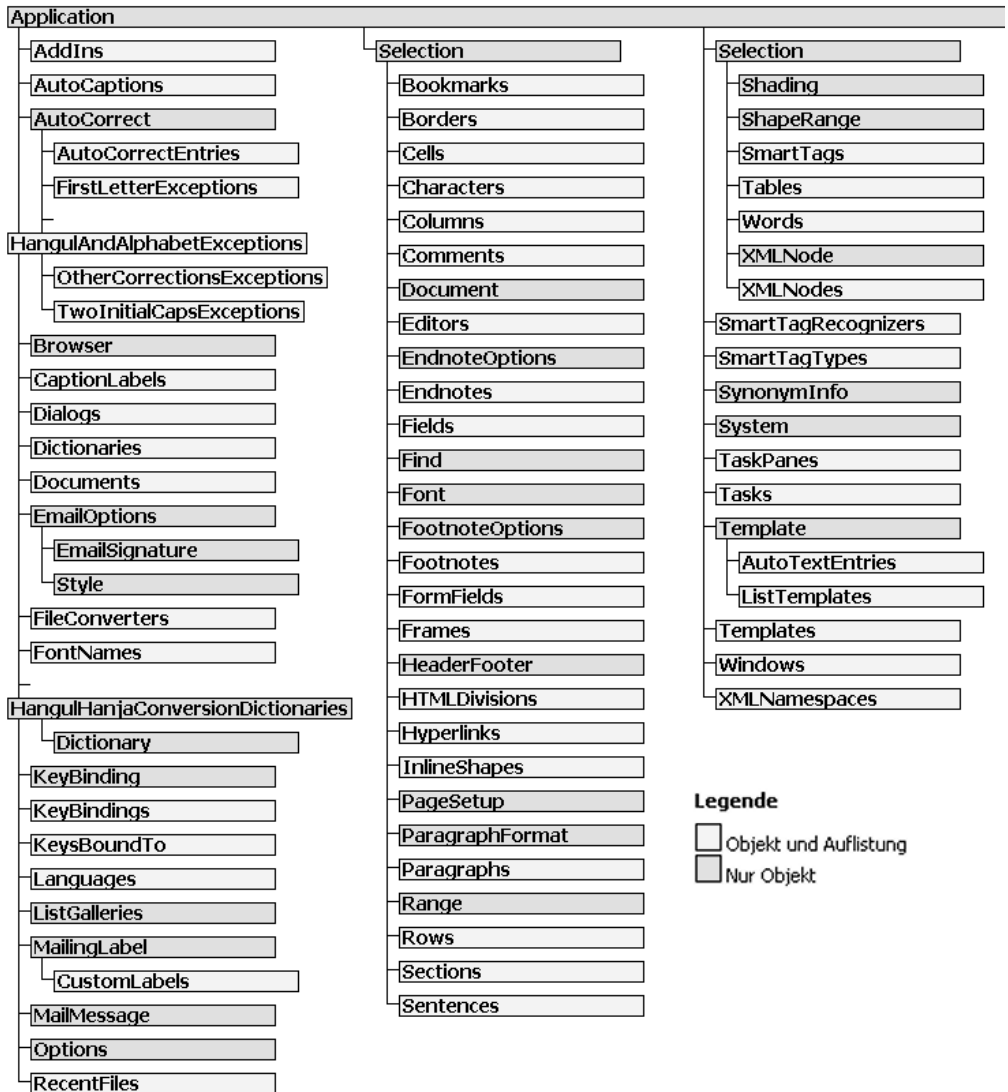
### In diesem Kapitel:

Aller Anfang ist schwer	30
Den Makrorekorder einsetzen	31
Den aufgezeichneten Code bearbeiten	33
Wo können Makros gespeichert werden?	37
Makros in der Benutzerschnittstelle einbinden und verwalten	38
Makrosicherheit	43

# Aller Anfang ist schwer

Mit jeder Aufgabe muss irgendwo begonnen werden. Dies ist bei der Automatisierung von Word nicht anders. Ein einfacher Einstiegspunkt ist schwer zu erkennen, denn das Objektmodell von Word ist groß und mächtig. In früheren Office-Versionen, als es ein gedrucktes Handbuch mit dem Inhalt der Office-Objektmodelle gab, war dies offensichtlich. Das Buch für Word war doppelt so dick wie jenes für Excel oder die Bücher für Outlook und PowerPoint zusammen.

Abbildg. 1.1 Die oberste Ebene des Word-Objektmodells



Die oberste Ebene des Objektmodells ist in Abbildung 1.1 grafisch dargestellt. Das Original der Abbildung finden Sie in der VBA-Hilfe von Word (siehe Kapitel 2) sowie im Internet auf

[http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vbawd11/html/wotocOMMap\\_HV01049667.asp](http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vbawd11/html/wotocOMMap_HV01049667.asp).

Jeder einzelnen Box in der Darstellung ist ein Hyperlink hinterlegt. Dieser führt zur Informationsseite des betreffenden Objekts oder zu dessen Auflistung, die ihrerseits zu Seiten mit den Eigenschaften und Methoden verknüpft ist. Sogar erfahrene Entwickler verlieren schnell im Seiten-Dickicht die Orientierung.

Wie kann also auf eine einfache Art der Einstiegspunkt zur Lösung des Problems gefunden werden? Die schnellste Hilfe erhält der Entwickler durch die Verwendung des Makrorekorders. Mit etwas Glück liefert dieser die zu einer Aufgabe benötigten Objekte, Eigenschaften und Methoden. Die zugehörigen Details werden in einem zweiten Schritt in der Hilfe (siehe Kapitel 2) nachgeschlagen. Danach wird der bereinigte Code in das effektive Makro eingebaut.

## Den Makrorekorder einsetzen

Den Makrorekorder rufen Sie über die Befehlsfolge *Extras/Makro/Aufzeichnen* oder mit einem Doppelklick auf das Kästchen **MAK** in der Statusleiste auf (Abbildung 1.2).

Abbildg. 1.2 Die Word-Statusleiste mit aktivierter Makrorekorder-Funktionalität



In dem nun eingeblendeten Dialogfeld *Makro aufzeichnen* (Abbildung 1.3) vergeben Sie für das Makro einen aussagekräftigen Namen (vorgeschlagen wird »Makron«, wobei *n* eine fortlaufende Nummer darstellt). Zusätzlich sollten Sie eine hilfreiche Beschreibung eintragen. Zudem besteht die Möglichkeit, das neue Makro einer Symbolleiste oder einer Tastenkombination in der Benutzeroberfläche zuzuweisen.

### TIPP

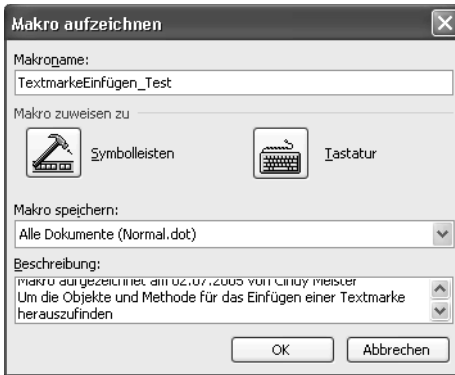
Der Makroname wird zugleich als »QuickInfo« eingeblendet, wenn der Mauszeiger über der Symbolleisten-Schaltfläche ruht.

Wichtig ist, den **Kontext** aus der Dropdownliste *Makro speichern* festzulegen. Mit dieser Option wird der eigentliche Speicherort des Makros festgelegt. Word kann Makros der ganzen Umgebung zugänglich machen oder sie nur in bestimmten Dokumenten oder Vorlagen zur Verfügung stellen. Detaillierte Informationen zu diesem Thema können im Abschnitt »Wo können Makros gespeichert werden?« in diesem Kapitel nachgeschlagen werden.

### HINWEIS

Wenn Sie einen Makronamen eingeben, der im Modul *NewMacros* bereits vorhanden ist, fragt Word, ob Sie den bestehenden überschreiben möchten, was Sie auch tun dürfen. Lehnen Sie ab, erhalten Sie Gelegenheit, einen anderen Makronamen einzugeben.

Abbildg. 1.3 Neben dem Makronamen ist es wichtig, den Speicherort für das Makro festzulegen



Nach Bestätigung des Dialogfeldes *Makro aufzeichnen* (falls Sie weder *Symbolleiste* noch *Tastatur* angeklickt haben), kehren Sie ins Dokument zurück. Die Symbolleiste *Aufzeichnung beenden* erscheint (meistens oben links über dem Dokument).

Von nun an werden fast alle in Word ausgeführten Interaktionen in eine Prozedur im Code-Modul *NewMacros* aufgezeichnet. Dabei sind folgende Punkte zu beachten:

- Das Anzeigen eines Dialogfeldes wird nicht aufgezeichnet, sondern das Endresultat der darin vorgenommenen Einstellungen. (Beispiel: Sie blenden das Dialogfeld *Datei öffnen* ein und wählen ein Dokument aus. Der Makrorekorder zeichnet das Öffnen dieses Dokuments auf, nicht aber das Einblenden des Dialogfeldes.)
- Der Makrorekorder erkennt nur jene Aktionen des Anwenders, die innerhalb der Word-Anwendungsumgebung ausgeführt werden. Fügen Sie beispielsweise ein Excel-Tabellenblatt in das Word-Dokument ein, wird das Einfügen wohl aufgezeichnet. Alle im Tabellenblatt ausgeführten Modifikationen werden allerdings nicht aufgezeichnet, da sie in der Excel-Umgebung vorgenommen werden.
- Ebenfalls nicht aufgezeichnet wird der Wechsel in ein anderes Anwendungsfenster. Ein Wechsel zwischen Word-Dokumentfenstern wird aufgezeichnet, sofern dieser über das Menü *Fenster* oder die Windows-Taskleiste erfolgt. Nicht erkannt werden Wechsel, die mit **[Alt] + [Win]** vorgenommen werden.
- Das Einfügen, Markieren und Formatieren von Grafiken wird aufgezeichnet. Es ist jedoch nicht möglich, per Mausklick zurück ins Dokument zu gelangen. Um dies zu tun, drücken Sie **[Esc]**.
- Um eine Markierung oder Handlung vorzunehmen, die mit der Tastatur oder einem Menübefehl nicht ausführbar oder ganz allgemein unerwünscht ist, können Sie den Makrorekorder vorübergehend anhalten, indem Sie auf die Schaltfläche *Aufzeichnung anhalten* in der Symbolleiste *Aufzeichnung beenden* klicken. Klicken Sie nochmals auf die Schaltfläche, die nun *Aufzeichnung fortsetzen* heißt, um mit der Aufzeichnung fortzufahren.

Mit der linken Schaltfläche *Aufzeichnung beenden* wird die Aufzeichnung beendet. Wechseln Sie zum Visual Basic-Editor (**[Alt] + [F11]**), um das Ergebnis im Modul *NewMacros* anzuschauen und anzupassen. Mehr über den VB-Editor erfahren Sie in Kapitel 2.

# Den aufgezeichneten Code bearbeiten

Der Makrorekorder hat eine lange Geschichte. Bis einschließlich Word-Version 95 (7.0) zeichnete er die Benutzerhandlungen treu in der Programmiersprache WordBasic auf. Das Resultat konnte ohne große Änderung eingesetzt werden. Seit Word 97 ist die Word-Programmiersprache VBA (»Visual Basic für Applikationen«). Diese ist auf einer objektorientierten Basis konzipiert. Dies bedeutet, dass der Code die Objekte in der Anwendung direkt ansprechen soll, statt die einzelnen Benutzerhandlungen eingabegetreu wiederzugeben. Dadurch kommt der Makrorekorder in eine Zwickmühle, weil er nur wahrnimmt, was der Benutzer während der Aufzeichnung ausführt. Die einzelnen Interaktionen können nicht abstrahiert und auf das Objektmodell im weiteren Sinne übertragen werden.

Das Ergebnis einer Aufzeichnung ist also nur bedingt einsetzbar; oft muss der Code mehr oder minder nachbearbeitet werden. Zudem ist ein aufgezeichnetes Makro schwierig zu verwalten, da direkt aus dem Code heraus kaum zu entnehmen ist, was es bezwecken soll. Leider werden die Programmzeilen vom Hersteller selten ausreichend kommentiert. Wird ein solches Makro innerhalb einer Firma weiter »vererbt«, ist es schwer bis unmöglich, dieses zu einem späteren Zeitpunkt den eventuell veränderten Bedürfnissen oder Änderungen in der Word-Umgebung anzupassen. Ein seit längerer Zeit eingesetztes Werkzeug fällt dann weg oder es müssen viele Stunden investiert werden, um das Makro wieder lauffähig zu machen.

Nehmen wir als extremes Beispiel die aufgezeichnete Prozedur in Listing 1.1. Die Einfügemarke befand sich zu Beginn der Aufzeichnung in der ersten Zelle einer Tabelle. Die Markierung wurde nach rechts über die ganze Zeile (fünf Zellen) erweitert und diese fett formatiert. Danach wurden Spaltenüberschriften in jede Zelle dieser Zeile eingegeben. Am Schluss steht die Einfügemarke in der ersten Spalte der zweiten Zeile (Abbildung 1.4). Damit das aufgezeichnete Makro wunschgemäß arbeitet, muss der Anwender die Einfügemarke vor dem Ausführen unbedingt in der ersten Zelle positionieren. Ansonsten tritt ein Laufzeitfehler auf.

Hätten Sie diesen Ablauf und die Bedingung beim Lesen der Programmzeilen wirklich erraten? Vielleicht, aber es hätte wohl einiges an Kopfzerbrechen benötigt. In Listing 1.1 sind Zweck und Ort der Handlung schwer erkennbar.

**Abbildg. 1.4** So soll die Tabelle *immer* aussehen, was durch das aufgezeichnete Makro nicht gewährleistet ist

Spalte 1	Spalte 2	Spalte 3	Spalte 4	Spalte 5

**Listing 1.1** Die mit dem Makrorekorder aufgezeichnete Prozedur

```
Sub TabelleVorbereiten()
'
' TabelleVorbereiten Makro
' Makro aufgezeichnet am 02.07.2005 von Cindy Meister
'
Selection.EndKey Unit:=wdLine, Extend:=wdExtend
Selection.EndKey Unit:=wdLine, Extend:=wdExtend
Selection.EndKey Unit:=wdLine, Extend:=wdExtend
Selection.EndKey Unit:=wdLine, Extend:=wdExtend
Selection.EndKey Unit:=wdLine, Extend:=wdExtend
Selection.EndKey Unit:=wdLine, Extend:=wdExtend
Selection.Font.Bold = wdToggle
```

**Listing 1.1** Die mit dem Makrorekorder aufgezeichnete Prozedur (Fortsetzung)

```

Selection.MoveLeft Unit:=wdCharacter, Count:=1
Selection.TypeText Text:="Spalte 1"
Selection.MoveRight Unit:=wdCell
Selection.TypeText Text:="Spalte 2"
Selection.MoveRight Unit:=wdCell
Selection.TypeText Text:="Spalte 3"
Selection.MoveRight Unit:=wdCell
Selection.TypeText Text:="Spalte 4"
Selection.MoveRight Unit:=wdCell
Selection.TypeText Text:="Spalte 5"
Selection.MoveRight Unit:=wdCell
End Sub

```

Sehen Sie sich jetzt das Listing 1.2 an. Diese Prozedur erfordert lediglich, dass sich die Einfügemarke innerhalb der Tabelle befindet. Sie formatiert die erste Zeile dieser Tabelle fett, beschriftet die Spalten und positioniert die Einfügemarke am Schluss in die erste Spalte der zweiten Zeile.

Es fällt sofort auf, dass diese Prozedur deutlich kürzer und viel aussagekräftiger ist (sofern der Leser über einige Englischkenntnisse verfügt). In einem ersten Schritt wird kontrolliert, ob sich die Einfügemarke innerhalb einer Tabelle befindet. Ist das der Fall (egal wo in der Tabelle), werden je eine Objektvariable auf die aktuelle Tabelle und auf die erste Zeile gesetzt. Diese Zeile wird fett formatiert. Danach schleift die Prozedur durch alle Zellen dieser Zeile und fügt den Text »Spalte « plus deren Zellenindex in jede Zelle ein. Abschließend wird die erste Zelle in der zweiten Zeile markiert. Die Markierung wird auf einen Punkt kollabiert, so dass der Benutzer sofort mit der Texteingabe weiterarbeiten kann.

**Listing 1.2** Die programmierte Prozedur, die eine Tabelle ebenso formatiert wie Listing 1.1

```

Sub TabelleVorbereiten2()
    Dim tbl As Word.Table
    Dim row As Word.Row

    If Selection.Range.Information(wdWithInTable) Then
        Set tbl = Selection.Tables(1)
        Set rw = tbl.Rows(1)
        rw.Range.Bold = True
        For i = 1 To rw.Cells.Count
            rw.Cells(i).Range.Text = "Spalte " & CStr(i)
        Next
        tbl.Cell(2, 1).Range.Select
        Selection.Collapse
    End If
End Sub

```

In C# entspricht der Code für das Beispiel der Darstellung in Listing 1.3. Der Code ist Teil eines Windows Form-Projekts und wird durch die oberste Schaltfläche in Abbildung 1.5 ausgeführt. Die Prozedur enthält eine minimale Fehlerbehandlung (einen Try-Block), da Word von außen automatisiert wird. Zunächst muss der Kontakt zur Word-Anwendung hergestellt werden, was mit der Methode `GetActiveObject` möglich ist. Schlägt diese fehl, springt die Ausführung zum Catch-Block am Ende der Prozedur. Sind keine Dokumente vorhanden, wird eine entsprechende Meldung angezeigt und die Ausführung abgebrochen. Nach erfolgreicher Ausführung wird ebenfalls eine Meldung eingeblendet und abschließend das `wdApp`-Objekt wieder freigestellt.

**HINWEIS**

Aus Platzgründen wird die Verbindung zur Word-Anwendung in den im Buch abgedruckten C#-Code-Beispielen meistens nicht angezeigt. Die verwendete Methode ist jedoch in den Beispiel-Projekten auf der CD-ROM ersichtlich.

**Abbildg. 1.5** Diese Windows-Form wurde in C# codiert. Die oberste Schaltfläche führt den Code in Listing 1.3 aus, um eine Tabelle zu formatieren.



**Listing 1.3** Die C#-Version von Listing 1.2

```
//zusätzliche Deklarationen am Projektanfang
using wd = Microsoft.Office.Interop.Word;
using wdMarshal = System.Runtime.InteropServices.Marshal;

private void TabelleVorbereiten2_CS()
{
    try
    {
        wd.Application wdApp = (wd.Application)
            wdMarshal.GetActiveObject("Word.Application");
        if (wdApp.Documents.Count == 0)
        {
            MessageBox.Show("Kein geöffnetes Dokument gefunden.");
            return;
        }
        object objDirectionEnd = wd.WdCollapseDirection.wdCollapseEnd;
        wd.Selection sel = wdApp.Selection;
        if ((bool) sel.Range.get_Information(wd.WdInformation.wdWithInTable))
        {
            wd.Table tbl = sel.Tables[1];
            wd.Row row = tbl.Rows[1];
            row.Range.Bold = 1;
            for (int i=1; i <= row.Cells.Count; i++)
            {
                row.Cells[i].Range.Text = "Spalte " + i;
            }
            tbl.Cell(2,1).Range.Select();
            sel.Collapse(ref objDirectionEnd);
        }
        //Das Word-Fenster anzeigen
        wdApp.Activate();
        wdMarshal.ReleaseComObject(wdApp);
        wdApp = null;
        MessageBox.Show("Fertig!");
        catch (System.Runtime.InteropServices.COMException ex)
```

**Listing 1.3** Die C#-Version von Listing 1.2 (*Fortsetzung*)

```
{
    MessageBox.Show("Word läuft nicht.");
}
```

Das oben genannte Beispiel zeigt den Unterschied zwischen einer alten »Makrosprache«, wie Word-Basic, und einer objektorientierten Programmiersprache auf. Gut konzipierter Code einer objektorientierten Programmiersprache ist »selbst dokumentierend«, der Programmablauf ist ohne jeglichen Kommentar zu erkennen.

Nicht alle aufgezeichneten Makros sind derart kryptisch, was die verwendeten Objekte anbelangt. Nachstehend finden Sie ein kleines aufgezeichnetes Makro, welches eine AutoForm (ein Rechteck) in das Dokument einfügt und anschließend nach links verschiebt.

```
Sub Macro2()
    ActiveDocument.Shapes.AddShape(msoShapeRectangle, 234#, 135#, 99#, 63#).Select
    Selection.ShapeRange.IncrementLeft -9#
    Selection.ShapeRange.IncrementLeft -9#
    Selection.ShapeRange.IncrementTop -9#
End Sub
```

Das nächste Makro fügt der Symbolleiste *Seriendruck* eine zusätzliche Symbolleisten-Schaltfläche hinzu.

```
Sub Macro4()
    CommandBars("Mail Merge").Controls.Add Type:=msoControlButton, ID:=245, Before:=7
End Sub
```

Die Code-Zeilen der beiden Makros lassen erkennen, dass es sich bei einer AutoForm um ein Shape-Objekt, bei einer Symbolleiste um ein CommandBar-Objekt handelt. Ferner verfügt das Shape-Objekt über die Methode `IncrementLeft` (in kleinen Schritten nach links verschieben), das CommandBar-Objekt enthält `Controls` (Steuerelemente), denen die Methode `Add` (hinzufügen) zugeordnet ist. Mit diesen Informationen können in der Hilfe (siehe Kapitel 2) die beiden Objekte erforscht werden.

**TIPP**

Gibt das Resultat der aufgezeichneten Interaktionen, wie in Listing 1.1, keinen direkten Aufschluss über das benötigte Objekt, kann ersatzweise das Erstellen oder das Einfügen des benötigten Objekts aufgezeichnet werden.

Wie wird jetzt aus einem wie in Listing 1.1 aufzeichneten Makro eine strukturierte, übersichtliche Prozedur, wie dies in Listing 1.2 der Fall ist? Dazu verhilft ein Programm, das auf Objekten basiert und auf deren Eigenschaften und Methoden zurückgreift. Dies ist ein Hauptziel dieses Handbuchs, dem vor allem die ersten beiden Teile gewidmet sind. Im ersten Teil stellen wir in Kapitel 2 die Programmierungsumgebung sowie den Visual Basic-Editor vor. In Kapitel 3 erhalten Sie eine Einführung in die Grundlagen der VBA-Sprache. Im darauf folgenden Teil befassen wir uns mit dem Word-Objektmodell.





Die Beispieldatei *Bsp01\_01.doc* mit den beiden Codebeispielen finden Sie auf der CD-ROM zu diesem Buch im Ordner `\Beispiele\Kap01`.

## Wo können Makros gespeichert werden?

Bevor Sie beginnen, Ihre persönliche Arbeitsumgebung mit den unterschiedlichsten Makros zu optimieren, sollten Sie sich über zwei Punkte im Klaren sein. Erstens müssen Sie wissen, wo die Makros überhaupt abgespeichert werden können. Zweitens müssen Sie sich Gedanken darüber machen, wer als Anwender dieser Makros in Frage kommt. Für Word gibt es vier verschiedene Möglichkeiten, den Programmcode zu speichern:

- im aktuellen Dokument,
- in einer beliebigen Dokumentvorlage,
- in der Standardvorlage *Normal.dot* sowie
- in einem Add-In.

Vielleicht haben Sie in verschiedenen Dialogfeldern bemerkt, dass für Makros die gleichen Speicherorte wie für Formatvorlagen und Symbolleisten zur Verfügung stehen. Außer in Dokumenten (\*.doc) können Sie auch AutoTexte an den genannten Orten speichern.

Jeder einzelne dieser Speicherorte hat seine Vor- und Nachteile, die zur Veranschaulichung in Tabelle 1.1 kurz zusammengefasst sind.

**Tabelle 1.1** Speicherorte für Makros und deren Vor- bzw. Nachteile

Speicherort	Vorteil	Nachteil
Dokument	Die Makros stehen immer zur Verfügung, egal auf welcher Arbeitsstation das Dokument bearbeitet wird.	Die Sicherheit für Makros muss mindestens auf »Mittel« gesetzt oder das VBA-Projekt muss signiert werden (siehe den Abschnitt »Makrosicherheit« in diesem Kapitel).
Dokumentvorlage	Die Makros stehen nur für Dokumente zur Verfügung, die auf der betreffenden Dokumentvorlage basieren.	Der Zugriff auf die Dokumentvorlage muss stets gewährleistet sein.
Normal.dot	Die Makros stehen für alle Dokumente, auch jene aus fremder Quelle, zur Verfügung.	Die Makros stehen nur einem einzelnen Anwender zur Verfügung.
Add-In	Die Makros werden über eine geschlossene Programmerweiterung eingesetzt.	Das Aktualisieren der Makros innerhalb einer Firmenumgebung benötigt normalerweise die Unterstützung des Netzwerkadministrators.

Weiterhin umfasst jeder dieser genannten Speicherorte eine definierte Zielgruppe, die als Anwender der Makros in Frage kommt.

**Tabelle 1.2** Speicherorte für Makros und deren mögliche Anwendergruppen

Speicherort	Mögliche Anwender
Dokument	Jeder Anwender, der das Dokument bearbeitet und hierfür zwingend auf die Makros angewiesen ist.
Dokumentvorlage	Jeder Anwender, der Dokumente auf der Basis dieser Dokumentvorlage erstellt oder bearbeitet (beispielsweise ein Protokoll).
Normal.dot	Persönlich genutzte Makros zur Optimierung des Arbeitsumfelds.
Add-In	Steht allen Anwendern uneingeschränkt zur Verfügung.

Mehr zum Thema *Normal.dot* lesen Sie in Kapitel 12.

### WICHTIG Hände weg von der *Normal.dot*



Normal.dot

Professionell erstellte Makros werden nicht in der *Normal.dot* abgelegt, diese ist dem Anwender vorbehalten.

Wir Autoren müssen immer wieder feststellen, dass Hersteller von Programmerweiterungen die entsprechenden Makros in der *Normal.dot* abspeichern. So wird die Datei mit Makros und Symbolleisten erweitert oder – noch schlimmer – während der Programminstallation sogar vollkommen ersetzt.

Wir vertreten grundsätzlich die Meinung, dass die *Normal.dot* für den Anwender bestimmt ist und nur dessen persönliche Einstellungen und Makros enthalten soll.

Werden Programmerweiterungen installiert, können die enthaltenen Makros an zwei Orten gespeichert werden:

- Werden spezielle Dokumente automatisiert (beispielsweise ein Protokoll), werden die zugehörigen Makros, Dialogfelder, Symbolleisten usw. in der entsprechenden Dokumentvorlage hinterlegt.
- Werden allgemeine Programmerweiterungen erstellt, die für alle Dokumente zur Verfügung stehen müssen (beispielsweise ein eigenes Dialogfeld »Drucken«), werden die zugehörigen Makros, Dialogfelder, Symbolleisten usw. in einem Add-In hinterlegt. Mehr zum Thema »Erstellen eines Add-Ins« erfahren Sie in Kapitel 13.

## Makros in der Benutzerschnittstelle einbinden und verwalten

Wie im Abschnitt »Den Makrorekorder einsetzen« in diesem Kapitel beschrieben, können Makros Symbolleisten und Tastenkombinationen zugewiesen werden. Was ist aber, wenn Sie diesen Schritt erst später vollziehen wollen? Das kann problemlos über die Benutzerschnittstelle erfolgen.

## Makro einer Symbolleiste-Schaltfläche zuweisen

Um ein Makro einer Symbolleiste oder einem Menü zuzuweisen, führen Sie die folgenden Schritte durch:

1. Blenden Sie in Word über die Befehlsfolge *Extras/Anpassen* das Dialogfeld *Anpassen* ein.
2. Wechseln Sie zur Registerkarte *Befehle*.
3. Legen Sie den Speicherort fest (siehe dazu den Abschnitt »Wo können Makros gespeichert werden?« in diesem Kapitel).
4. Wählen Sie die Kategorie *Makros* aus.
5. Markieren Sie in der rechten Liste den gewünschten Eintrag (Abbildung 1.6) und ziehen Sie ihn mit gedrückter linker Maustaste an die Stelle eines Menüs oder einer Symbolleiste, von wo aus das Makro später aufgerufen werden soll.
6. Nach einem Klick mit der rechten Maustaste auf die neu erstellte Schaltfläche öffnet sich ein Kontextmenü (Abbildung 1.7), über welches die Schaltfläche entsprechend den eigenen Vorstellungen verändert werden kann. (Alternativ können Sie im Dialogfeld *Anpassen* die Schaltfläche *Auswahl ändern* betätigen.) Von besonderem Interesse sind die Einträge *Name* (für die Beschriftung), *Schaltflächensymbol ändern* (um ein eigenes Symbol zu entwerfen oder ein existierendes anzupassen) sowie die Liste mit *Standard* (nur Symbol), *Nur Text (immer)*, *Nur Text (in Menüs)* und *Schaltflächensymbol und Text*.

### Präzedenz in mehreren geladenen Dokumenten

Konflikte könnten zwischen Makros, Symbolleisten und Tastenkombinationen in Dokumenten und jenen in Vorlagen entstehen. Was ist, wenn zwei Dateien gleichnamige Makros und Symbolleisten enthalten oder wenn gleiche Tastenkombinationen mit verschiedenen Befehlen belegt sind?

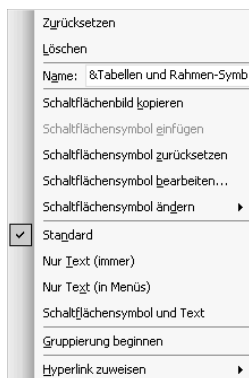
Auch daran hat Microsoft gedacht und die unten stehende Reihenfolge für Word festgelegt:

- Makros werden durch *ProjektName.ModulName.MakroName* identifiziert, wie auch die Liste *Befehle* in Abbildung 1.6 zeigt. Es ist deshalb wichtig, jedes VBA-Projekt eindeutig zu benennen (siehe dazu Kapitel 2). Haben zwei Makros genau den gleichen Namen, muss einer der Namen geändert werden. Wurde das zugehörige Makro zuvor einer Symbolleiste und einer Tastenkombination zugewiesen, geht diese Verbindung verloren und muss neu hergestellt werden.
- Für Symbolleisten und Tastenkombinationen gilt:
  - Vorrang hat, was in einem einzelnen Dokument definiert ist.
  - Danach wird berücksichtigt, was in der dem Dokument angehängten Dokumentvorlage festgelegt ist.
  - Es folgt in der Hierarchie ein eventuell geladenes Add-In.
  - Und an letzter Stelle rangieren Zuweisungen aus der *Normal.dot*.

**Abbildg. 1.6** Durch Ziehen eines Eintrags aus der Liste *Befehle* das Makro einer Symbolleiste oder einem Menü hinzufügen



**Abbildg. 1.7** Das Kontextmenü der neu erstellten Schaltfläche ist nur bei geöffnetem *Anpassen*-Dialogfeld erreichbar



### **WICHTIG** Ein Makro erscheint nicht in der Liste in der Benutzerschnittstelle


Es wird Ihnen auffallen, dass nicht alle Prozeduren eines Moduls in der Liste unter *Extras/Makro/Makros* oder unter *Extras/Anpassen* aufgeführt werden. Im Grunde genommen können nur einfache, öffentliche Prozeduren vom Benutzer ausgeführt werden. Folgende Prozedurarten erscheinen nicht in diesen Listen:

- Prozeduren, die als *Private* bezeichnet sind (mehr zu *Private* lesen Sie in Kapitel 3).
- Prozeduren, für die Argumente definiert sind.
- Prozeduren, die einen Wert zurückgeben (Funktionen).
- Prozeduren, die sich in einem Modul befinden, das die Anweisung `Option Private Module` enthält.
- Prozeduren, die in einem Klassenmodul oder UserForm-Modul stehen.



**HINWEIS** In Kapitel 15 wird erläutert, wie Symbolleisten und deren Schaltflächen durch das Objektmodell erstellt und verwaltet werden.

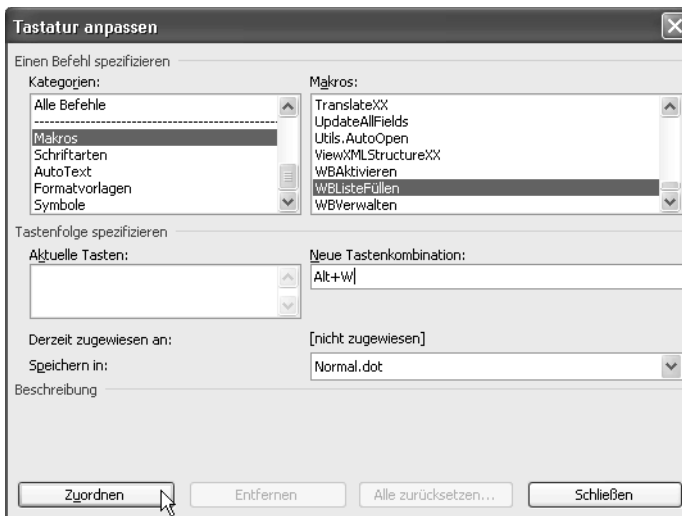
Makro  
Tasten-  
kombi-  
nation  
zuweisen

Um ein Makro einer Tastenkombination zuzuweisen, gehen Sie wie folgt vor:

1. Blenden Sie über die Befehlsfolge *Extras/Anpassen* das Dialogfeld *Anpassen* ein.
2. Klicken Sie auf die Schaltfläche *Tastatur* und legen Sie im geöffneten Dialogfeld *Tastatur anpassen* (Abbildung 1.8) den Speicherort fest (siehe dazu den Abschnitt »Wo können Makros gespeichert werden?« in diesem Kapitel).
3. Wählen Sie die Kategorie *Makros* aus.
4. Markieren Sie in der Liste *Makros* den Eintrag, dem eine Tastenkombination zugewiesen werden soll.
5. Klicken Sie in das Feld *Neue Tastenkombination*.
6. Drücken Sie auf Ihrer Tastatur die gewünschte Tastenfolge. Beachten Sie anschließend im Dialogfeld *Tastatur anpassen* den Eintrag rechts neben *Derzeit zugewiesen an*. Falls hier *[nicht zugewiesen]* steht, dürfen Sie dieses Kürzel problemlos festlegen. Wird hier jedoch ein Befehlsname angezeigt, müssen Sie sich entscheiden, ob Sie die Tastenkombination tatsächlich neu belegen oder es mit einer anderen Kombination versuchen möchten. Hierzu drücken Sie die -Taste und geben ein anderes Kürzel ein.

Abbildg. 1.8

Die Tastenkombination  +  ist noch nicht belegt und darf dem Makro *WBListeFüllen* zugewiesen werden



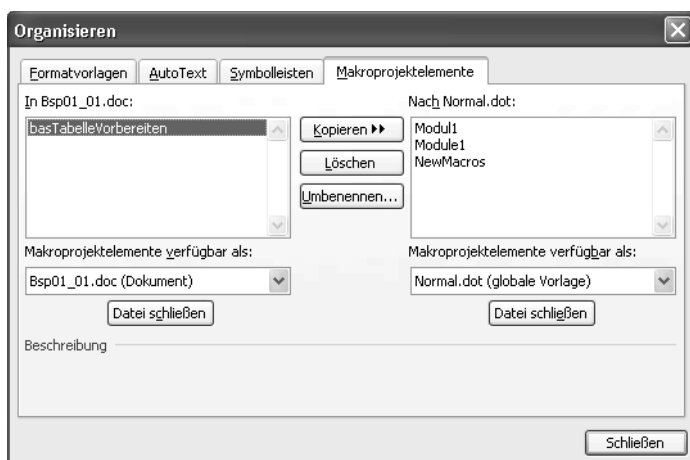
**HINWEIS** In Kapitel 16 wird erklärt, wie Tastenkombinationen durch das Objektmodell definiert werden.

## Makros kopieren

Oft werden Makros zuerst in der *Normal.dot* erstellt, weil dies der standardmäßige Speicherort ist. Erst später wird realisiert, dass das Makro besser in einem einzelnen Dokument oder in einer Vorlage untergebracht wäre. Der Code kann dann problemlos im VB-Editor kopiert, gelöscht und eingefügt werden.

Was aber, wenn Sie Makros mit anderen Anwendern teilen wollen und diese Personen sich nicht mit dem VB-Editor auseinander setzen möchten? Dazu bietet Word ein tolles Werkzeug an, mit dem einzelne Code-Module (oder auch Symbolleisten, AutoTexte und Formatvorlagen) zwischen Word-Dateien kopiert werden können: das Dialogfeld *Organisieren* (siehe Abbildung 1.9).

**Abbildg. 1.9** Mit dem Dialogfeld *Organisieren* werden Makros, Symbolleisten etc. problemlos verwaltet und zwischen Word-Dateien kopiert



Die Schaltfläche *Organisieren* steht in verschiedenen Dialogfeldern zur Verfügung, u.a. in *Extras/Makro/Makros*. Bitte beachten Sie, dass nur ganze Module und nicht einzelne Prozeduren verwaltet werden können. Falls die Makros mit einer Symbolleiste verbunden sind, empfiehlt es sich, zuerst die Makros und anschließend die Symbolleiste in die andere Datei zu kopieren.

Um einen Eintrag zu kopieren, markieren Sie ihn in einem der Listenfelder und klicken dann auf die Schaltfläche *Kopieren*. Möchten Sie eine der Dateien auswechseln, müssen Sie diese erst schließen. Klicken Sie dazu auf die Schaltfläche *Datei schließen*. Die Schaltflächenbeschriftung ändert sich danach in *Datei öffnen*, um das Öffnen eines anderen Dokuments oder einer Dokumentvorlage zu ermöglichen.

### HINWEIS

Die Funktionalität *Organisieren* ist auch im Word-Objektmodell vorhanden. Schlagen Sie in der VBA-Hilfe die Begriffe *OrganizerCopy*, *OrganizerDelete* und *OrganizerRename* nach. Die programmatische Erstellung von Makros wird in Kapitel 19 bei der Diskussion über die Automatisierung des VB-Editors vorgestellt.

# Makrosicherheit

Als in Word 2.0 die Programmiersprache »WordBasic« zur Erstellung von Makros implementiert wurde, haben Programmierer mit nicht allzu edlen Absichten erkannt, dass mit diesem Werkzeug nicht nur der Anwender in seiner täglichen Arbeit unterstützt, sondern auch allerlei Unfug angestellt werden kann. Mit der Einführung von Word 95 tauchten die ersten so genannten Makroviren auf. Die damit verseuchten Dokumente enthielten Makros mit schadhaftem Programmcode. Diese Viren richteten zum Teil beträchtlichen Schaden an und infizierten jeweils alle bearbeiteten Dokumente.

In Word 97 hat Microsoft als Gegenmaßnahme eine mehrstufig aufgebaute so genannte *Makrosicherheit* integriert. Jetzt kann der Anwender festlegen, ob die Makros innerhalb von Dokumenten überhaupt aktiviert und ausgeführt werden oder nicht.

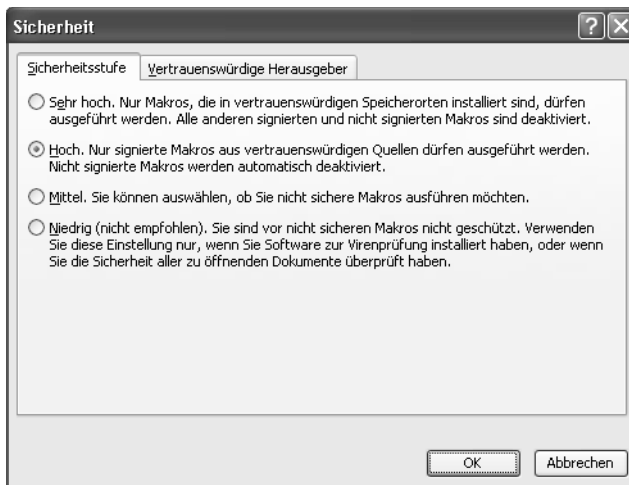
## Sicherheitsstufe anpassen

Um die Sicherheitsstufe auf Ihrer Arbeitsstation anzupassen, wählen Sie den Menübefehl *Extras/Makro/Sicherheit*.

Wir Autoren empfehlen Ihnen, die Sicherheitsstufe auf der Registerkarte *Sicherheitsstufe* auf den Wert *Hoch* zu setzen. So stellen Sie sicher, dass nur Makros ausgeführt werden, deren Code Sie kennen oder deren Quellen Sie vertrauen.

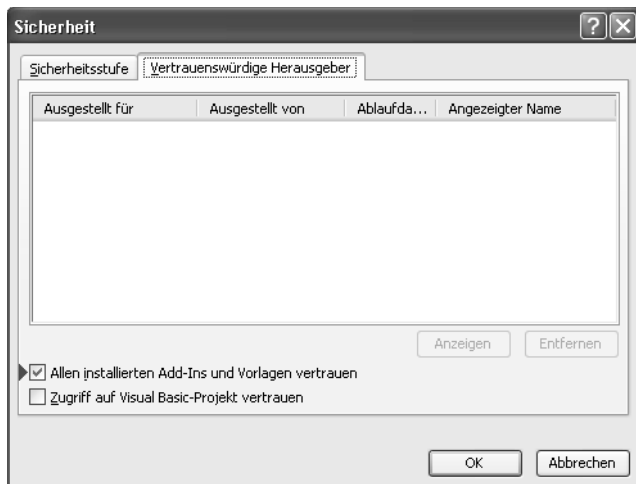
Die in Abbildung 1.10 ersichtliche Sicherheitsstufe *Sehr hoch* wurde erst mit Word 2003 eingeführt.

**Abbildg. 1.10** Setzen Sie die Sicherheitsstufe gemäß unserer Empfehlung auf *Hoch*



Zusätzlich empfehlen wir, das Kontrollkästchen *Allen installierten Add-Ins und Vorlagen vertrauen* auf der Registerkarte *Vertrauenswürdige Herausgeber* zu aktivieren (unter Word 2000 und 2002 heißt die Registerkarte übrigens *Vertrauenswürdige Quellen*). Wir stellen uns auf den Standpunkt, dass das Abspeichern von Dateien in diese Ordner viel bewusster vorgenommen wird und deshalb diesen Dateien im Allgemeinen vertraut werden kann.

**Abbildung. 1.11** Aktivieren Sie das markierte Kontrollkästchen



#### HINWEIS

Durch das Aktivieren des in Abbildung 1.11 gezeigten Kontrollkästchens wird den Add-Ins und Vorlagen der folgenden Ordner vertraut:

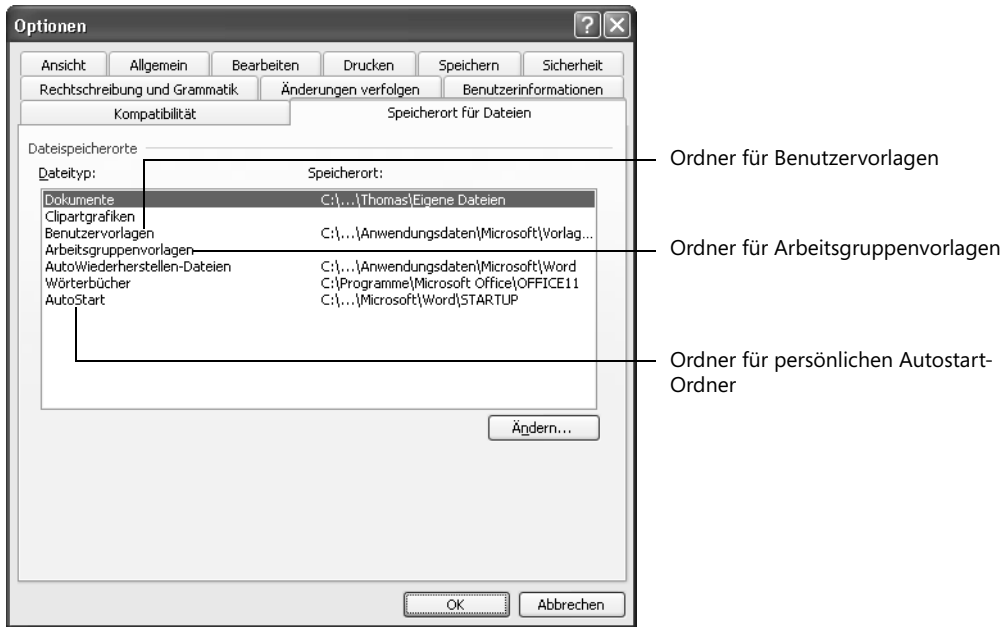
- C:\Programme\Microsoft Office\Office11\Startup als allgemeiner Autostart-Ordner
- C:\Dokumente und Einstellungen\[Benutzername]\Anwendungsdaten\Microsoft\Word\StartUp als persönlicher Autostart-Ordner, sofern in den Optionen kein anderer Ordner eingetragen ist
- C:\Dokumente und Einstellungen\[Benutzername]\Anwendungsdaten\Microsoft\Vorlagen mit den Benutzervorlagen, sofern in den Optionen kein anderer Ordner eingetragen ist
- Dem Ordner für die Arbeitsgruppenvorlagen, sofern in den Optionen ein solcher festgelegt wurde.

Wie in Abbildung 1.12 ersichtlich, können die einzelnen Speicherorte bei Bedarf individuell festgelegt werden. Werden keine Einträge gesetzt, sind die oben aufgezählten Ordner als Standardwerte gültig. Um die Speicherorte anzupassen, wählen Sie den Menübefehl *Extras/Optionen* und wechseln zur Registerkarte *Speicherort für Dateien*. Tragen Sie die gewünschten Ordner in die entsprechenden Kategorien ein.

In Word 2003 wird nur noch einem Autostart-Ordner vertraut, es handelt sich dabei um jenen Ordner, der in den Optionen als *AutoStart*-Ordner festgelegt wurde.



Abbildg. 1.12 Legen Sie die gewünschten Verzeichnisse für die Add-Ins und Vorlagen fest



In einem späteren Schritt zeigen wir Ihnen, wie Sie trotz dieser Sicherheitseinstellungen bei Bedarf Makros in einzelnen Dokumenten abspeichern können, so dass diese weiterhin funktionsfähig bleiben.

## Zertifikate und Signaturen

Um die Integrität (Identifikation des Herstellers und Unveränderlichkeit des Inhalts) von Programmen und Dokumenten feststellen zu können, werden in der Informatik digitale Zertifikate eingesetzt.



Zertifikat

Digitale Zertifikate bauen auf zwei Komponenten auf: dem eigentlichen *Zertifikat* und der zugehörigen *Signatur*.

Das Zertifikat beinhaltet den öffentlichen Schlüssel des Zertifikatinhabers. Neben diesem Schlüssel beinhaltet das Zertifikat den Zertifikatsnamen, Seriennummer, Gültigkeitsdauer, Name der Zertifizierungsstelle usw.

Das Zertifikat wird von einer anerkannten Zertifizierungsstelle ausgestellt<sup>a</sup>. Diese gewährleistet, dass der Antragsteller auch wirklich diejenige Person ist, für die er sich ausgibt. Die Person, also der Hersteller bzw. der Entwickler, wird folglich identifiziert.

Bei der Ausstellung eines Zertifikats wird dem Antragsteller durch die Zertifizierungsstelle ein Schlüsselpaar (privater und öffentlicher Schlüssel<sup>b</sup>) geliefert. Um die Integrität dieses Zertifikats zu garantieren, wird es mit dem privaten Schlüssel der Zertifizierungsstelle digital unterschrieben.<sup>c</sup>

## Zertifikate und Signaturen

Signatur

Signierte Programme, Treiber, Dokumente oder eben auch Makros können mit dem zugehörigen Zertifikat »geöffnet« werden. Als Analogie zur realen Welt dient der eigene Reisepass.

Bei der Signatur handelt es sich um einen privaten Schlüssel. Mit diesem können Programme, Treiber, Dokumente oder eben auch Makros signiert werden. Als Analogie zur realen Welt dient die persönliche Unterschrift.

Beim Signieren einer Datei wird ein Hash über die Datei gelegt. Dieser wird mit dem privaten Schlüssel des Zertifikats verschlüsselt und als Signatur der Datei hinzugefügt. Der verschlüsselte Hash kann nur mit dem öffentlichen Schlüssel entschlüsselt werden. Wird der Hash auf der Arbeitsstation des Empfängers erneut gebildet, so muss dieser mit dem entschlüsselten übereinstimmen.

Das Zertifikat muss durch den Eigentümer allgemein zugänglich gemacht werden. Es muss auf der Arbeitsstation, auf welcher die signierten Daten bearbeitet werden, installiert sein. Das Zertifikat wird sodann in die Liste der vertrauenswürdigen Herausgeber aufgenommen.

Ein signiertes Makro kann nur zusammen mit dem zugehörigen Zertifikat aktiviert werden, sofern die Sicherheitseinstellungen, wie im Abschnitt »Sicherheitsstufe anpassen« in diesem Kapitel empfohlen, auf *Hoch* gesetzt wurden. Auf diese Weise ist sichergestellt, dass das Makro seit der Auslieferung von keinem Unbefugten geändert wurde.

Als Analogie zur realen Welt könnte man beispielsweise das Einlösen eines Schecks am Schalter verwenden. Die Person am Schalter vertraut darauf, dass der vorgewiesene Reisepass echt ist. Anhand des Fotos kann der Inhaber identifiziert werden. Durch dessen Unterschrift, die im Reisepass ebenfalls vorhanden ist, wird bewiesen, dass der Pass nicht geändert wurde (Foto ausgetauscht).

- Bekannte und anerkannte Zertifizierungsstellen sind beispielsweise VeriSign (<http://www.verisign.de>) oder Thawte (<http://www.thawte.com>).
- Der private Schlüssel ist geheim und sollte niemandem zugänglich gemacht werden. Er dient zum Verschlüsseln einer Botschaft. Der öffentliche Schlüssel hingegen muss dem Empfänger einer verschlüsselten Botschaft zugänglich gemacht werden. Er dient zum Entschlüsseln dieser Botschaft.
- Digitales Unterschreiben eines Zertifikats: Ein Hash wird über das Zertifikat gebildet, mit dem privaten Schlüssel der Zertifizierungsstelle verschlüsselt und als Signatur dem Zertifikat angehängt.

Die Verwendung von digitalen Signaturen hat jedoch auch seine Vor- und Nachteile. Die wichtigsten davon sind kurz zusammengefasst.

Tabelle 1.3 Gegenüberstellung der Vor- und Nachteile von digitalen Signaturen

Vorteil	Nachteil
Es ist sichergestellt, dass der Inhalt einer Datei durch keine unberechtigte Person verändert wurde.	Die Datei kann nach einer Modifikation nur auf derjenigen Arbeitsstation neu signiert werden, auf welcher die Signatur tatsächlich installiert ist.
Der Hersteller der Datei ist bekannt bzw. kann identifiziert werden.	Der öffentliche Schlüssel, also das Zertifikat, muss jederzeit zugänglich sein oder zusammen mit der Datei geliefert werden.
Höhere Sicherheit	Zusätzlicher Verwaltungsaufwand

## Signatur mittels *selfcert.exe* erstellen



Eigene  
Signatur  
erstellen

Für den privaten Umgang mit Makros lohnt es sich nicht, eine Signatur bei einer offiziellen Zertifizierungsstelle zu erwerben. Dennoch ist es sinnvoll, wenn jedes VBA-Projekt signiert wird.

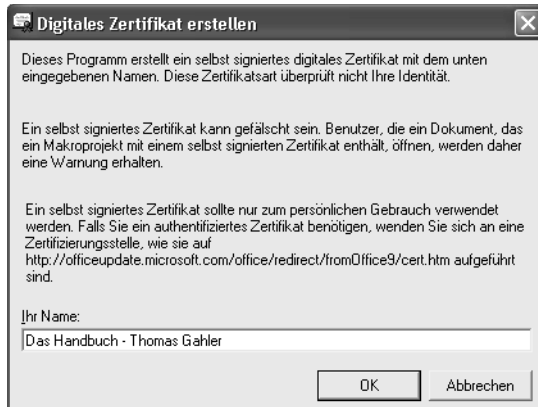
Zum Lieferumfang von Microsoft Office gehört das kleine Programm *selfcert.exe*. Die Datei wird, sofern die entsprechende Option ausgewählt wurde, während der Programminstallation im Verzeichnis *C:\Programme\Microsoft Office\Office11* angelegt. Mit diesem Hilfsprogramm lassen sich auf der eigenen Arbeitsstation Signaturen erstellen und anschließend zum Signieren von VBA-Projekten verwenden.

Um eine Signatur mittels *selfcert.exe* zu erstellen, gehen Sie wie folgt vor.

1. Starten Sie im Verzeichnis *C:\Programme\Microsoft Office\Office11* das Programm *selfcert.exe*.
2. Legen Sie für Ihre Signatur einen aussagekräftigen Namen fest und bestätigen Sie diese Angaben mit OK.

Abbildg. 1.13

Dem neuen Zertifikat einen aussagekräftigen Namen zuweisen



### WICHTIG

Beachten Sie dabei, dass Sie den Namen des neuen Zertifikats, also die Signatur, nicht mehr ändern können. Es besteht auch keine Möglichkeit, das erstellte Zertifikat zusammen mit der Signatur zu exportieren und auf einer anderen oder zweiten Arbeitsstation zu installieren, um an diesem Arbeitsplatz ebenfalls Makros entwickeln und mit der gleichen Signatur versehen zu können.

Mit *selfcert.exe* erstellte Zertifikate sollten wirklich nur für den privaten Gebrauch verwendet werden und keinesfalls bei professionell erstellten Dokumentvorlagen und Add-Ins Anwendung finden.

Mit Windows Server 2003 können ebenfalls Zertifikate erstellt werden. Diese sind in erster Linie für den internen Gebrauch innerhalb der Firmenumgebung zu verwenden, können aber bei Bedarf auch an Dritte weitergegeben werden. In diesem Fall wird jedoch bewusst auf eine Bürgschaft einer offiziellen Zertifizierungsstelle verzichtet, welche die Echtheit des Zertifikats garantiert.

## VBA-Projekte mit einer Signatur versehen

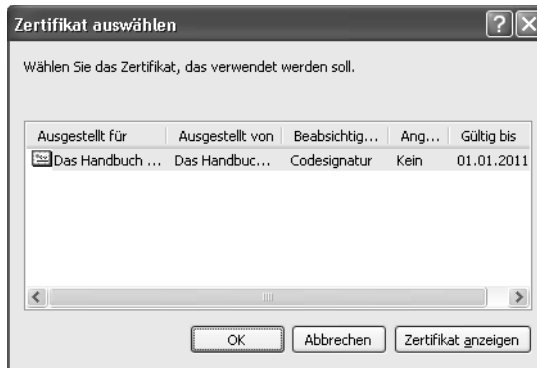
Damit die Makros innerhalb eines Add-Ins, einer Vorlage oder eines Dokuments im Zusammenspiel mit den empfohlenen Sicherheitseinstellungen aktiviert werden können, müssen die einzelnen Projekte mit einer digitalen Signatur ausgestattet werden. Diese Signatur stellt sicher, dass die Makros von keiner unbefugten Person geändert werden.

Um ein Projekt mit einer digitalen Signatur auszustatten, gehen Sie wie folgt vor:

1. Öffnen Sie die entsprechende Datei und wechseln Sie in den Visual Basic-Editor.
2. Rufen Sie den Menübefehl *Extras/Digitale Signatur* auf und betätigen Sie im Dialogfeld *Digitale Signatur* die Schaltfläche *Wählen*.
3. Markieren Sie im Dialogfeld *Zertifikat auswählen* das gewünschte Zertifikat in der Liste der vorhandenen Zertifikate und bestätigen Sie die Auswahl mit *OK*.
4. Bestätigen Sie anschließend die erfolgte Zuweisung ebenfalls mit *OK*.

Abbildg. 1.14

Weisen Sie das gewünschte Zertifikat dem Projekt zu



### HINWEIS

Die Zuweisung der Signatur muss für jedes Projekt einzeln erfolgen. Solange das Projekt an der gleichen Arbeitsstation bearbeitet wird, bleibt diese Zuweisung bestehen.

## Zertifikat zur Signatur erstellen



Zertifikat  
exportie-  
ren

Wurden Makros mit der gemäß dem Abschnitt »Signatur mittels *selfcert.exe* erstellen« erzeugten Signatur signiert, muss in einem zweiten Schritt das zugehörige Zertifikat erzeugt werden, damit es zusammen mit den zum Projekt gehörenden Dateien weitergereicht werden kann. Dieses Zertifikat stellt den so genannten öffentlichen Schlüssel dar.

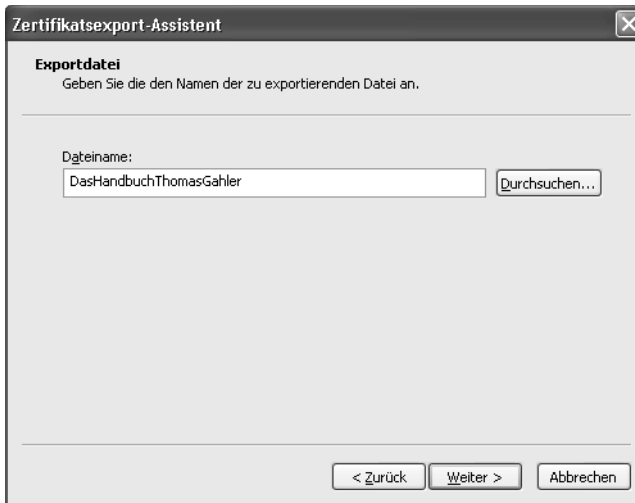
Um das Zertifikat zu einer auf *selfcert.exe* basierenden Signatur zu erstellen, gehen Sie wie folgt vor:

1. Rufen Sie den Visual Basic-Editor auf und wählen Sie den Menübefehl *Extras/Digitale Signatur*.
2. Klicken Sie im Dialogfeld *Digitale Signatur* auf die Schaltfläche *Wählen*.
3. Im Dialogfeld *Zertifikat auswählen* betätigen Sie die Schaltfläche *Zertifikat anzeigen* und wechseln dann zur Registerkarte *Details*.
4. Betätigen Sie die Schaltfläche *In Datei kopieren* und folgen Sie den Anweisungen des Assistenten.

5. Weisen Sie der zukünftigen Datei im vierten Arbeitsschritt einen aussagekräftigen Namen zu und folgen Sie den weiteren Anweisungen des Assistenten.

Abbildg. 1.15

Bestimmen Sie den Dateinamen der Zertifikatsdatei

**HINWEIS**

Die soeben erstellte Datei muss öffentlich zugänglich sein oder zusammen mit derjenigen Datei, die signiert wurde, weitergereicht werden.

Bitte beachten Sie, dass wir Autoren unter dem Begriff »weiterreichen« nur die Verbreitung zum privaten Gebrauch (beispielsweise Familie, Freunde und Bekannte usw.) verstehen.

## Zertifikat einlesen

Vertrauens-  
würdige  
Quellen

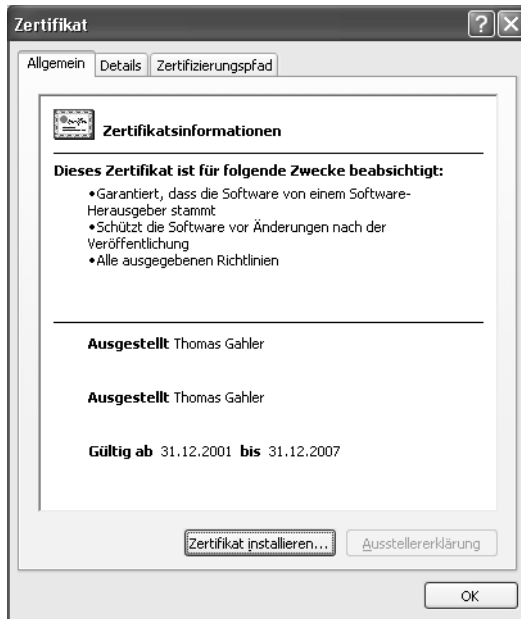
Wird eine Datei, wie im Abschnitt »VBA-Projekte mit einer Signatur versehen« in diesem Kapitel beschrieben, signiert und weitergereicht, muss das zugehörige Zertifikat vorab auf der Arbeitsstation eingelesen werden, damit die Makros aktiviert werden können.

Um ein Zertifikat in die Liste der vertrauenswürdigen Quellen aufzunehmen, gehen Sie wie folgt vor:

1. Beschaffen Sie sich die entsprechende Zertifikatsdatei beim Hersteller der Makros bzw. beim Autor der Datei, sofern diese nicht bereits mitgeliefert wurde.
2. Starten Sie die Installation des Zertifikats, indem Sie mit der Maus einen Doppelklick auf die betreffende Datei (Dateierweiterung *.cer*) ausführen.
3. Betätigen Sie im Dialogfeld *Zertifikat* die Schaltfläche *Zertifikat installieren*. Folgen Sie den Anweisungen des Assistenten.

Abbildg. 1.16

Das Zertifikat muss installiert werden, bevor es in der Liste der vertrauenswürdigen Quellen zur Verfügung steht



Die Aufnahme des Zertifikats in den Zertifikatsspeicher ermöglicht das Aktivieren der Makros all jener Dateien, die mit der gleichen Signatur signiert wurden. Trotzdem wird beim Öffnen einer solchen Datei weiterhin eine Sicherheitswarnung ausgegeben. Die entsprechende Warnung kann definitiv quittiert werden, indem das Kontrollkästchen *Makros aus dieser Quelle immer vertrauen* aktiviert wird (siehe Abbildung 1.17).

Abbildg. 1.17 Sicherheitswarnung quittieren und Makros aktivieren



# Zusammenfassung

In diesem ersten Kapitel wurde der Einstieg in die Word-Makros vermittelt. Dies ist wichtig, damit sich ein Anwender ohne Vorkenntnisse in VBA schnell damit zurechtfinden kann.

- Es wurde besprochen wie einfache Makros mit dem so genannten Makrorekorder aufgezeichnet (Seite 31 ff.) und wie diese in einem zweiten Schritt nachbearbeitet werden können (Seite 33 ff.).
- Es wurde vermittelt, wo die einzelnen Makros gespeichert (Seite 37 ff.) und wie diese in die Benutzerschnittstelle von Word eingebunden werden können (Seite 38 ff.).
- Ein weiterer Abschnitt beschäftigte sich mit der integrierten Makrosicherheit von Word (Seite 43) und wie ein VBA-Projekt digital signiert werden kann (Seite 48 ff.).





## Kapitel 2

# Der Visual Basic-Editor

### In diesem Kapitel:

Der Blick ins Innere: Die Fenster	54
Hinter den Kulissen: Die Optionen	61
Die VBA-Hilfe – eine versteckte Schatzkammer	66
Wo alle Fäden zusammenlaufen: Der Objektkatalog	69
The Show must go on: Code bearbeiten	73

In allen Office-Anwendungen steht die gleiche Programmiersprache – VBA (Visual Basic für Applikationen) – zur Verfügung. Als gemeinsame Programmierumgebung wird der Visual Basic-Editor genutzt. Hier wird der Code erfasst und verwaltet. Kennen Sie sich im VB-Editor einer anderen Office-Anwendung aus, müssen Sie nur noch das Objektmodell von Word erlernen. Falls Sie noch nie mit dem VB-Editor gearbeitet haben, seien Sie beruhigt: Im Gegensatz zum Objektmodell ist das Kennenlernen der VB-Editor-Umgebung nicht schwer.

Wenn Sie vorhaben, Word mit einer anderen Programmiersprache zu automatisieren – beispielsweise aus VB6 oder aus dem .NET-Framework heraus, sollten Sie sich trotzdem mit dem VB-Editor vertraut machen. Um Einsicht in das Word-Objektmodell zu erhalten, empfiehlt sich das Aufzeichnen von Makros. Diese Makros finden sich schließlich im VB-Editor wieder.

Den VB-Editor rufen Sie entweder über die Menüfolge *Extras/Makro/Visual Basic- Editor* oder durch Drücken der Tastenkombination **Alt** + **F11** auf.

Alle Teile und Befehle des VB-Editors werden in der Hilfedatei zur »Microsoft Visual Basic Documentation« erläutert (mehr zum Umgang mit der VBA-Hilfe ist im Abschnitt »Die VBA-Hilfe – eine versteckte Schatzkammer« in diesem Kapitel beschrieben). In diesem Kapitel werden wir die ersten Schritte aufzeigen sowie besondere Aspekte hervorheben, die spezifisch für Word sind oder in der Hilfe nicht behandelt werden.

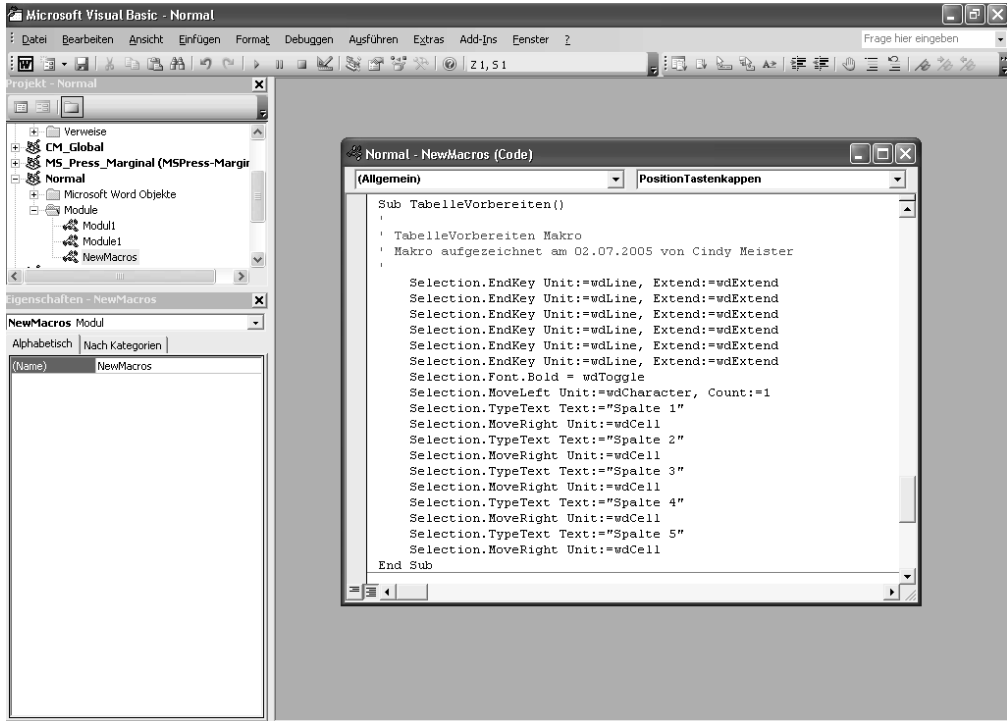
**HINWEIS**

Der VB-Editor weist ebenfalls eine Automatisierungsschnittstelle auf. Es ist möglich, die Fenster ein- und auszublenden, Module und UserForms zu erstellen und zu bearbeiten sowie Verweise zu anderen Code-Bibliotheken zu verwalten. Dieser Aspekt wird in Kapitel 19 vorgestellt.

## Der Blick ins Innere: Die Fenster

Beim allerersten Betrachten des VB-Editors kann eine gewisse Unsicherheit auftreten, da kein eigentlicher Blickfang vorhanden ist. Im Gegensatz zu Word, wo die Einfügemarke auf einem großen, weißen »Blatt« blinkt, präsentiert der VB-Editor zwei kleinere Fenster auf der linken Seite, den *Projekt-Explorer* sowie das *Eigenschaftenfenster*, und rechts daneben entweder eine große Leere, ein großes Fenster oder, wie in Abbildung 2.1 ersichtlich, ein kleineres Fenster innerhalb der großen Leere. Das Format im rechten Teilbereich hängt davon ab, in welchem Zustand sich das Code-Fenster befand, als der VB-Editor das letzte Mal geschlossen wurde, und ob die Code-Ansicht irgendeines Moduls aktiviert ist (mehr dazu im Abschnitt »The Show must go on: Code bearbeiten« in diesem Kapitel).

Abbildg. 2.1 Der Visual Basic-Editor

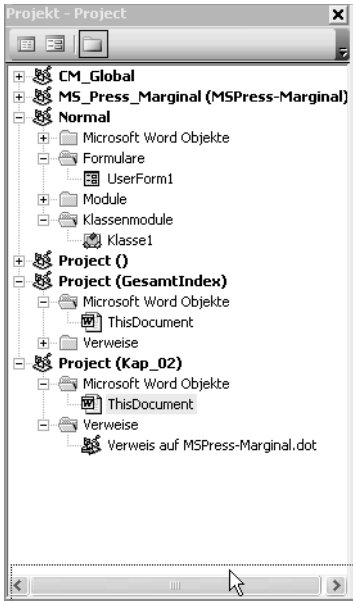


Gefällt Ihnen das Layout dieser Fenster nicht, dürfen Sie sie beliebig im VB-Editor platzieren. Ziehen Sie einfach mit der Maus an einer Titelleiste. Der *Projekt-Explorer* und das *Eigenschaftenfenster* dürfen frei stehen oder an einer der vier Seiten verankert werden. Dem Code-Fenster steht die restliche Fläche des Bildschirms zur Verfügung.

#### TIPP

Das Fixieren von Projekt-Explorer und Eigenschaftsfenster untereinander in der gleichen Spalte kann eine echte Herausforderung darstellen. Am besten gehen Sie wie in Abbildung 2.2 dargestellt vor. Ziehen Sie das Eigenschaftsfenster erst über das Fenster des Projekt-Explorers, dann langsam nach unten, bis der Umriss sich der Breite des anderen Fensters anpasst und als gepunktete Linie angezeigt wird. Wenn Sie nun die Maustaste loslassen, stehen beide Fenster (hoffentlich) wieder schön untereinander.

Abbildg. 2.2 Den *Projekt-Explorer* und das *Eigenschaftenfenster* untereinander platzieren



Das Größenverhältnis zwischen Projekt-Explorer und Eigenschaftenfenster lässt sich ändern, indem die dazwischen liegende Leiste nach oben oder nach unten verschoben wird.

Über die für Sie passende Positionierung können Sie aber erst entscheiden, wenn Sie wissen, wozu die verschiedenen Fenster dienen.

## Der Projekt-Explorer

Im Projekt-Explorer finden Sie alle in Word geladenen Dokumente. Des Weiteren enthält diese Liste alle angehängten Dokumentvorlagen der geöffneten Dokumente sowie alle globalen Vorlagen und geladenen Add-Ins.

Links neben jedem Projektnamen befindet sich ein »+«- oder »-«-Zeichen, das den Projekteinhalt ein- bzw. ausblendet. Falls eine Dokumentvorlage als Add-In geladen ist, kann der Projekteinhalt nicht angezeigt werden. Die entsprechende Datei muss zuerst in Word geöffnet werden. Die Ausnahme zu dieser Regel bildet die standardmäßige, globale Vorlage *Normal.dot* (mehr über *Normal.dot*, Vorlagen, Add-Ins und deren Verwaltung lesen Sie in Kapitel 1).

Nicht sichtbar sind Projekte in Dokumenten und Vorlagen, deren Makros nicht vertraut wurde (die Makrosicherheit wurde in Kapitel 1 vorgestellt), sowie solche, die mit einem Kennwort geschützt wurden. Im letzten Fall blendet der VB-Editor eine Eingabeaufforderung für das Kennwort ein.

Versuchen Sie, ein gesperrtes Projekt anzuzeigen, meldet Word »Projekt kann nicht angezeigt werden«.

Der Inhalt eines VBA-Projekts ist auf mehrere Ordner verteilt:

- *Microsoft Word-Objekte* umfasst das ThisDocument-Modul. Dies ist eine besondere Art von Klassenmodul, das Teile des Dokuments, worin das Projekt sich befindet, der Programmierung zur Verfügung stellt. Beispiele sind die Dokumentereignisse Document\_Open, Document\_New und Document\_Close und im Dokument eingebettete ActiveX-Steuerelemente (mehr darüber finden Sie in Kapitel 11).

---

**HINWEIS** Die Dokumentereignisse Document\_Open, Document\_New und Document\_Close existieren parallel zu den Makros AutoOpen, AutoNew und AutoClose. Sie alle werden in Kapitel 7 näher vorgestellt.

---

Öffentliche Prozeduren im ThisDocument-Modul werden zudem als Methoden und Eigenschaften des Dokument-Projekts aufgelistet, wenn sie in einem anderen Projekt mit einem Verweis eingebunden sind (siehe Kapitel 9).

- Jedes Projekt enthält auch einen Ordner *Verweise*. Hier werden alle Word-Projekte verzeichnet, auf welche direkt zugegriffen werden kann. Standardmäßig ist *Normal.dot* immer aufgeführt. Weitere fügen Sie über die Befehlsfolge *Extras/Verweise* hinzu (mehr zu diesem Dialogfeld finden Sie in Kapitel 8).
- Zusätzlich gibt es die Ordner *Module*, *Klassenmodule* sowie *Formulare*, sofern dem Projekt mindestens ein Modul, Klassenmodul oder eine UserForm hinzugefügt wurde.

---

**TIPP** Aufgezeichnete Makros befinden sich im Ordner *Module* und werden im Modul *NewMacros* des Projekts *Normal* abgelegt.

---

Um den Code eines Moduls zu sehen, doppelklicken Sie auf den Moduleintrag. Ein Code-Fenster wird eingeblendet.

Die Modul-Eigenschaften werden mit einem Einzelklick auf den Moduleintrag im Eigenschaftenfenster angezeigt.

---

**HINWEIS** Mehr Informationen zum Inhalt dieses Fensters finden Sie im Hilfethema *Hilfe zur Benutzeroberfläche von Visual Basic/Fenster/Projekt-Explorer*.

---

## Das Eigenschaftenfenster

Im Eigenschaftenfenster werden alle standardmäßigen Einstellungen für ein Objekt vorgenommen. Projekte und Module bieten nur die Eigenschaft Name an.

---

**WICHTIG** Machen Sie es sich zur Gewohnheit, jedes Projekt und jedes Modul eindeutig zu benennen. Liegen zwei Projekte mit dem gleichen Namen vor, kann es zu Konflikten kommen. Eindeutige Namen helfen auch bei der Zuweisung von Makros zu Symbolleisten-Schaltflächen, Menüs und Tastaturkombinationen, da das Feld *Befehle* im Dialogfeld *Extras/Anpassen/Befehle* relativ schmal ist. Meistens sind Projekt-, Modul- und lediglich der Anfang des Makronamens sichtbar (siehe auch Kapitel 1).

---

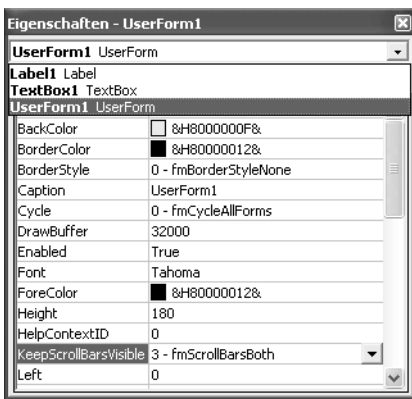
UserForms und die darin eingefügten Steuerelemente stellen mehr Eigenschaften zur Verfügung. Damit können beispielsweise Höhe und Breite sowie Farben und Schriftarten festgelegt werden. Mehr zum Thema *UserForms* lesen Sie in Kapitel 14.

Die mit Abstand meisten Eigenschaften werden für das ThisDocument-Objekt aufgeführt. Hier können etliche Einstellungen für das Dokument festgelegt werden, ohne sich durch mehrere Dialogfelder in der Benutzerschnittstelle quälen zu müssen.

Das Eigenschaftensfenster besteht aus einer Dropdown-Liste im oberen Teil sowie einer zweispaltigen Tabelle darunter (siehe Abbildung 2.3). Die Dropdown-Liste beinhaltet sämtliche Objekte, die zu einer Klasse oder einem Objekt gehören. Da es sich bei der UserForm um eine besondere Art von Klassenmodul handelt, werden alle auf dem Formular stehenden Steuerelemente zusammen mit der UserForm in der Liste aufgeführt. Um die Eigenschaften für ein Steuerelement zu bearbeiten, können Sie diesen UserForm-Entwurf markieren oder das Steuerelement aus der Dropdown-Liste wählen.

In der Tabelle unterhalb der Dropdown-Liste werden die Eigenschaften in der linken und deren Werte in der rechten Spalte angezeigt. Je nach Datentyp einer Eigenschaft wird der Wert eingetippt (Beispiel *Caption*), aus einer Liste gewählt (*KeepScrollBarsVisible*) oder in einem Dialogfeld festgelegt (*BackColor*).

Abbildg. 2.3 Das *Eigenschaftensfenster* macht es leicht, Einstellungen festzulegen



Markieren Sie eine Eigenschaft und betätigen Sie dann die Taste **[F1]**, um über die Hilfe nähere Informationen zu erhalten.

Wenn Sie – wie wir – lieber mit der Tastatur als mit der Maus arbeiten, hat uns dies Microsoft nicht besonders leicht, aber doch möglich gemacht. Die **[Tab]**-Taste wechselt in Folge zwischen der linken Spalte, der rechten Spalte, der Dropdown-Liste und der Registerkarte (in Abbildung 2.1 ersichtlich). Mit **[Shift] + [Tab]** geht's in die umgekehrte Richtung. Wenn Sie in der rechten Spalte die **[Enter]**-Taste drücken, kehren Sie zurück in die linke Spalte. Die Dropdown-Liste sowie eine Liste in der rechten Spalte werden mit **[Alt] + [Down]** aufgeklappt. **[Up]** und **[Down]** bewegen die Markierung in der linken Spalte sowie innerhalb der aufgeklappten Listen.

#### HINWEIS

Mehr Informationen zum Inhalt dieses Fensters finden Sie im Hilfethema *Hilfe zur Benutzeroberfläche von Visual Basic/Fenster/Eigenschaftensfenster*.

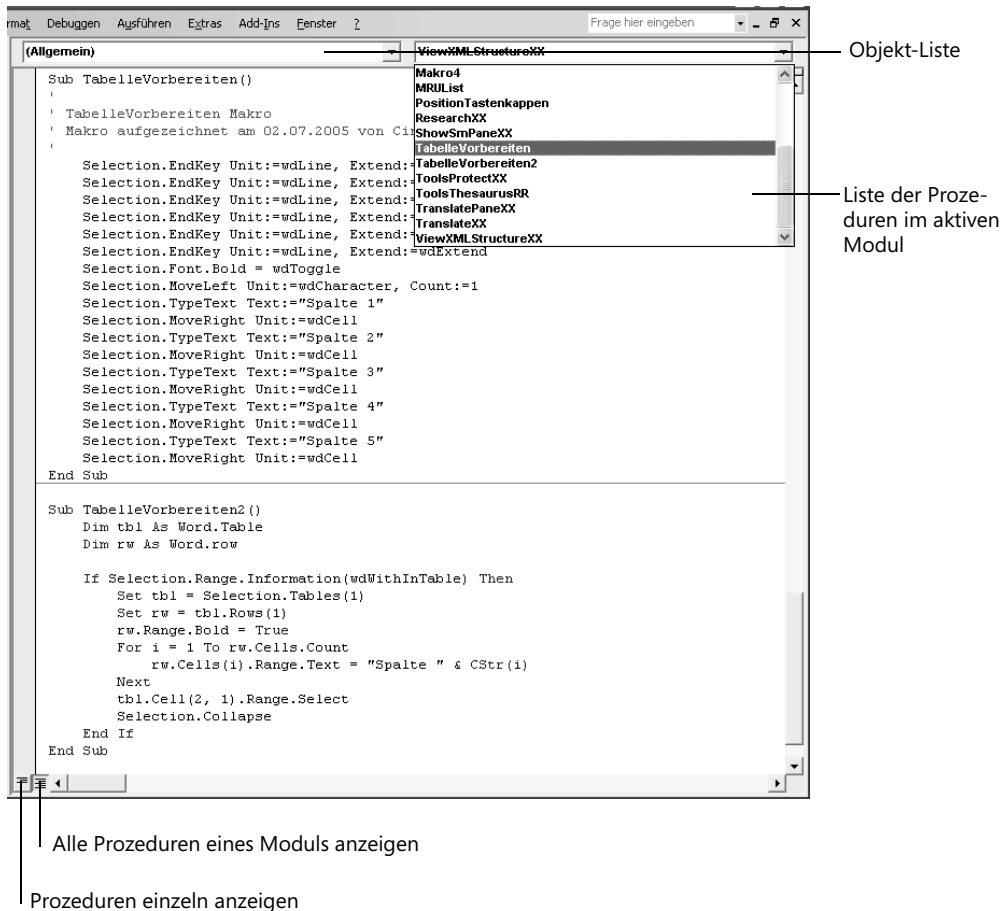
## Das Code-Fenster

Bei diesem dritten Fenster (Abbildung 2.4) handelt es sich um eine Art Text-Editor, in dessen Bereich der Code bearbeitet wird. Falls das Fenster nicht maximiert ist, wird in seiner Titelleiste der Name des Projekts sowie des Moduls, worin der Code »lebt«, angezeigt. Bei maximiertem Fenster stehen diese Angaben in der Titelleiste des VB-Editors.

Unmittelbar unter der Titelleiste befinden sich zwei Dropdown-Listen. In der linken Liste erscheinen – ähnlich wie in der Dropdown-Liste des Eigenschaftensfensters – alle Objekte einer Klasse. Handelt es sich um ein Standardmodul (wie *NewMacros*), steht nur der Eintrag (*Allgemein*) darin.




Rechts werden alle Prozeduren alphabetisch aufgelistet, die zur Auswahl in der linken Liste gehören. So erhalten Sie eine Übersicht eines Code-Moduls und können schnell zu einer Prozedur wechseln, indem Sie ihren Eintrag auswählen.

**Abbildg. 2.4** Das Code-Fenster. Die Option (unten rechts), um alle Prozeduren untereinander im gleichen Fenster anzuzeigen, ist aktiviert.



Unten links, neben der horizontalen Bildlaufleiste, befinden sich zwei kleine Schaltflächen. Hierüber legen Sie fest, ob alle Prozeduren untereinander in der Erfassungsreihenfolge oder einzeln angezeigt werden.

Hätte dieses Buch farbige Abbildungen, würden Sie erkennen, dass der Text im Code-Fenster mehrfarbig dargestellt wird. So wird schnell sichtbar, ob der Text ein Schlüsselbegriff oder ein Kommentar ist, oder ob beim Kompilieren ein Fehler gefunden wurde. Wie Sie diese Farben einstellen, ist im Abschnitt »Optionen für das Editorformat« in diesem Kapitel beschrieben.

Die Einzüge im abgebildeten Code-Fenster heben Code-Blöcke hervor und verleihen einer Prozedur optisch mehr Struktur. Der Code wird dadurch lesbarer und einfacher zu verwalten. Das Drücken der -Taste rückt den Text der markierten Zeilen eine Stufe nach rechts ein. Mit der Tastenkombination  +  rücken Sie ihn eine Stufe nach links wieder aus.

### HINWEIS

Mehr Informationen zu diesem Fenster finden Sie im Hilfethema *Hilfe zur Benutzeroberfläche von Visual Basic/Fenster/Code-Fenster*.

## Die Symbolleisten

Neben den Fenstern und der Menüleiste stellt der VB-Editor noch Symbolleisten zur Verfügung. Wie üblich in Microsoft-Anwendungen, sind diese in der Menüfolge *Ansicht/Symbolleisten* oder im Kontextmenü der Menüleiste aufgelistet (Abbildung 2.5).

Abbildg. 2.5 Die im VB-Editor zur Verfügung stehenden Symbolleisten



Die Symbolleiste *Voreinstellung* ist in Abbildung 2.1 zu sehen. Sie stellt ein breites Spektrum an Werkzeugen bereit, wie *Speichern*, *Kopieren*, *Einfügen*, *Rückgängig machen* und *Suchen*. Zudem enthält sie auch Symbolschaltflächen für die Makroausführung, die Fenster-Anzeige und die Hilfe. Ihr letztes Element zeigt die aktuelle Position (Zeile und Spalte) der Einfügemarke im Code-Fenster an.

Die Symbolleiste *Bearbeiten* enthält nützliche Befehle für die Arbeit im Code-Fenster, während *Debuggen* solche für das Testen von Makros anbietet. Diese Werkzeuge werden in den entsprechenden Diskussionen zu diesen Themen vorgestellt.

Die Symbolleiste *UserForm* wird automatisch eingeblendet, wenn Sie auf ein Formular im Entwurfsfenster klicken.

Über den Menübefehl *Anpassen* können Sie Befehle anders ordnen und eigene Symbolleisten mit ausgewählten Befehlen erstellen.



**HINWEIS** Die Symbolleisten werden ausführlich erklärt im Hilfethema *Hilfe zur Benutzeroberfläche von Visual Basic/Symbolleisten*. Wie Schaltflächen mit eigenen Befehlen erstellt werden können, ist in Kapitel 15 erläutert.

# Hinter den Kulissen: Die Optionen

Eine korrekt eingerichtete Arbeitsumgebung ist im VB-Editor genauso wichtig wie in jeder anderen Anwendung. Bislang haben wir uns nur mit der Oberfläche befasst. Der nächste Schritt führt uns hinter die Kulissen des VB-Editors.

Über die Registerkarten des Dialogfeldes zum Menübefehl *Extras/Optionen* werden die Einstellungen für den VB-Editor verwaltet. Hilfe zum Inhalt dieser Registerkarten erhalten Sie schnell und kontextabhängig über die entsprechende Schaltfläche.

## Optionen für den Editor

Die erste Registerkarte befasst sich mit dem Verhalten des Editors im Code-Fenster. Wir möchten Sie vor allem auf die zweite Option, *Variablendeklaration erforderlich*, aufmerksam machen. Standardmäßig ist sie nicht eingeschaltet, die Autoren sind jedoch der Meinung, sie müsste es sein.

Option  
Explicit

Im eingeschalteten Zustand wird automatisch bei der Einfügung eines Moduls an dessen Anfang die Anweisung `Option Explicit` hinzugefügt. Damit muss jede Variable, die im Code verwendet wird, explizit deklariert werden (mehr über die Deklaration von Variablen steht im Kapitel 3). Zunächst werden Sie vielleicht etwas verwundert fragen, warum Sie das tun sollen. Das bedeutet doch mehr Arbeit, es geht doch ganz gut, einfach eine Variable einzutippen, wenn man sie braucht?

Ja, Visual Basic erlaubt dies, die meisten anderen Programmiersprachen jedoch nicht. Aber was zuerst wie eine Arbeitserleichterung aussieht, rächt sich irgendwann, und es geht mehr Zeit verloren, als jemals gespart wurde. `Option Explicit` bewahrt vor Fehlern im Code, die schwierig zu finden sind.

Nehmen wir als einfaches Beispiel die Funktion in Listing 2.1. Der Benutzer wird zur Eingabe mehrerer Zahlen aufgefordert. Diese werden in einer Berechnung weiterverwendet, die ihrerseits das Funktionsergebnis liefert, das in weiteren Berechnungen eingesetzt wird.

Listing 2.1

Diese Funktion gibt das Ergebnis einer Berechnung nur dann zurück, wenn *Option Explicit* nicht vorhanden ist, ansonsten erscheint ein Kompilierungsfehler.

```
Private Const sngMITARBEITERSTUNDENANSATZ as Single = 10.50
Private Const sngFÜHRUNGSKRÄFTESTUNDENANSATZ as Single = 18.70
Private Const sngPLANERSTUNDENANSATZ as Single = 25.65

Function ProjektLohnkosten() as Single
    Dim lAnzahlMitarbeiter as Long
    Dim lAnzahlFührungskräfte as Long
    Dim lAnzahlPlaner as Long
    Dim sngLohnKosten as Single

    lAnzahlMitarbeiter = InputBox("Anzahl Mitarbeiter eingeben.")
    lAnzahlFührungskräfte = InputBox("Anzahl Führungskräfte eingeben")
```

**Listing 2.1**

Diese Funktion gibt das Ergebnis einer Berechnung nur dann zurück, wenn *Option Explicit* nicht vorhanden ist, ansonsten erscheint ein Kompilierungsfehler. (Fortsetzung)

```
lAnzahlPlaner = InputBox("Anzahl Planer eingeben")
sngLohnKosten = (lAnzahlMitarbeiter * sngMITARBEITERSTUNDENANSATZ) + _
    (lAnzahlFührungskräfte * sngFÜHRUNGSKRÄFTESTUNDENANSATZ) + _
    (lAnzahlPlaner * sngPLANERSTUNDENANSATZ)
ProjektLohnkosten = sngLohnKosten
End Function
```

Es fällt vielleicht erst bei einer Revisorenprüfung auf, dass die Projektkosten ständig zu tief be- und verrechnet wurden. Nach der Ursache wird tagelang geforscht, und erst nachdem der Wert jeder Variablen in dieser Funktion kontrolliert wird, entdeckt man, dass

```
lAnzahlMitarbeiter * MitarbeiterStundenansatz
```

immer gleich Null (Zero) ist. Es wird festgestellt, dass lAnzahlMitarbeiter nie ein Wert zugewiesen wird, weil die Variable in der Codezeile mit der InputBox-Funktion nicht als lAnzahlMitarbeiter sondern als lAnzahlMitarbeitre definiert wurde.

Wäre Option Explicit am Anfang dieses Moduls eingetragen, hätte der VB-Editor beim ersten Testversuch gemeldet: »Fehler beim Kompilieren. Variable wurde nicht deklariert« und diese Variable hervorgehoben.

Fazit: Immer am richtigen Ort sparen!

Automa-  
tische  
Syntax-  
über-  
prüfung

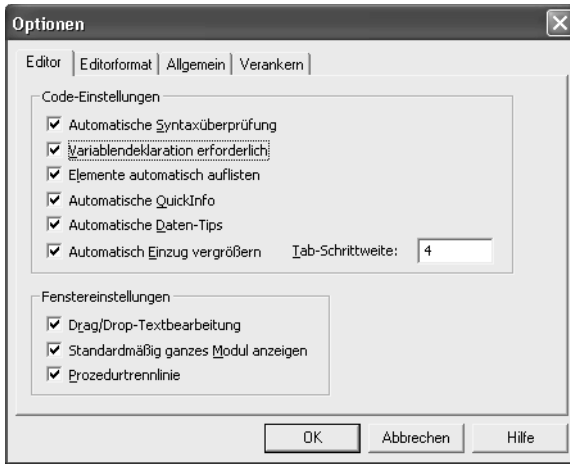
Die weiteren Optionen sind in der Hilfe detailliert beschrieben. Sie sind alle hilfreich und sollten, nach Auffassung der Autoren, aktiviert bleiben. Die erste in der Liste, *Automatische Syntaxüberprüfung*, kann allerdings nervenaufreibend sein, wenn Sie oft mitten in einer Codezeile abbrechen, um an anderer Stelle eine Änderung vorzunehmen. Beispiel: Sie merken beim Schreiben einer Schleife, dass eine Variable für den Zähler deklariert werden muss. Statt die Zeile fertig zu schreiben, wollen Sie zuerst am Prozeduranfang die Variablendeklaration vornehmen:

```
Sub Beispiel()
    'Variablendeklaration käme hier
    For
End Sub
```

Nach Eingabe von For probieren Sie, in die Zeile unterhalb des Prozedurnamens zu klicken. Der VB-Editor unterbricht jäh mit der Meldung »Fehler beim Kompilieren. Erwartet: Variable« und färbt For rot ein. Um weiterzuarbeiten, müssen Sie auf OK klicken, um die Meldung zu schließen. (Sie haben auch die Möglichkeit, in der Hilfe nach weiteren Fehlermeldungen zu suchen.)

Wie gesagt, diese Meldung kann lästig sein, wenn man weiß, was man tut. Wir meinen aber, die Vorteile überwiegen die Nachteile, und lassen die Option eingeschaltet.

Abbildg. 2.6 Die Optionen für die Arbeit im Code-Fenster sollten möglichst alle aktiviert sein



## Optionen für das Editorformat

Wie bereits kurz erwähnt, kann der VB-Editor gewisse Code-Teile besonders hervorheben, so dass sie auf den ersten Blick erkennbar sind. Der Text von Syntax-Fehlern (wie gerade besprochen) wird leuchtend rot dargestellt. Führen Sie den Code schrittweise aus (was in Kapitel 3 näher erläutert wird), ist die gegenwärtig aktive Codezeile gelb hinterlegt. Schlüsselwörter sind blau gefärbt, Kommentare grün und Haltepunkte sind standardmäßig dunkelrot hervorgehoben.

### HINWEIS

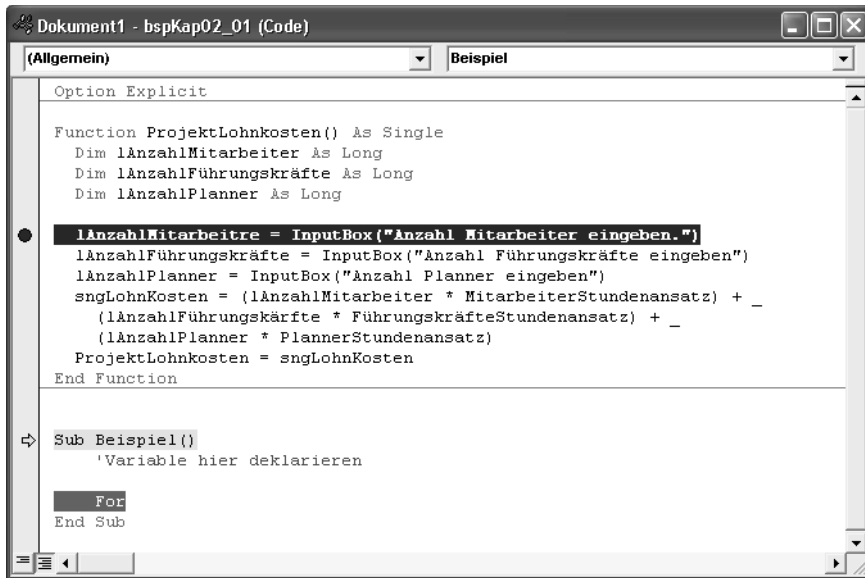
Bei einem *Schlüsselwort* handelt es sich um ein Wort oder ein Symbol, das als Teil der Programmiersprache von Visual Basic erkannt wird, wie beispielsweise eine Anweisung, ein Funktionsname oder ein Operator. Die Ausdrücke »Function«, »As« und »Single« sind Beispiele für Schlüsselwörter. Diese dürfen nicht als Namen einer Variablen verwendet werden.

Mit den Voreinstellungen müssen Sie sich aber nicht zufrieden geben. Wie der Abbildung 2.8 zu entnehmen ist, kann die Farbe jeweils für Text (*Vordergrund*), Hintergrund sowie Anzeiger (wo ein Anzeiger vorhanden ist) für jeden Listeneintrag individuell festgelegt werden. Die Wirkung dieser Optionen kann in einem Buch mit Schwarzweiß-Bildern nicht gebührend illustriert werden. Aber gerade wenn Sie farbenblind sind, dürften die Optionen hilfreich sein, da sie verschiedene Kontrastwirkungen ermöglichen, wie in Abbildung 2.7 ersichtlich. Hier wurde die Farbe der Schlüsselwörter auf grell-rosa gestellt, um diese Wörter vom übrigen Code zu unterscheiden.

### HINWEIS

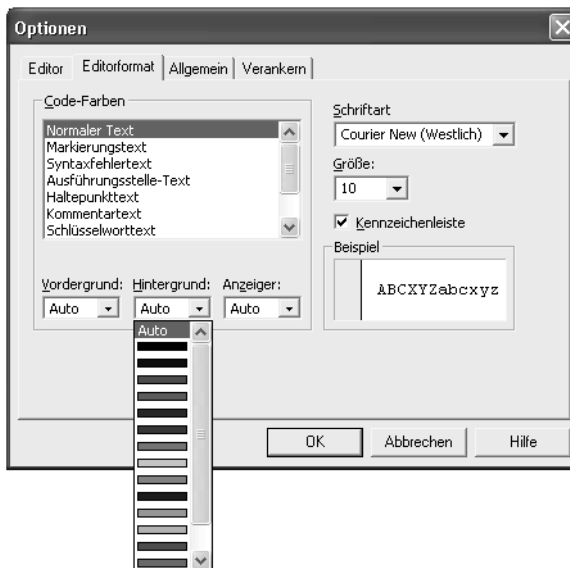
Wir raten Ihnen, eventuelle Änderungen an den Optionen erst nach Beendigung dieses Teils vorzunehmen, da sich die weitere Diskussion auf die Installationseinstellungen bezieht.

Abbildg. 2.7 Die *Code-Farben* ermöglichen es, besondere Textstellen auf schnell erkennbare Art und Weise zu kennzeichnen



Ferner können Sie die Schriftart und -größe für den Code-Text selbst bestimmen. Wir raten jedoch davon ab, eine proportionale Schriftart zu verwenden, da Code allgemein lesbarer ist, wenn die Buchstaben ordentlich in Spalten stehen.

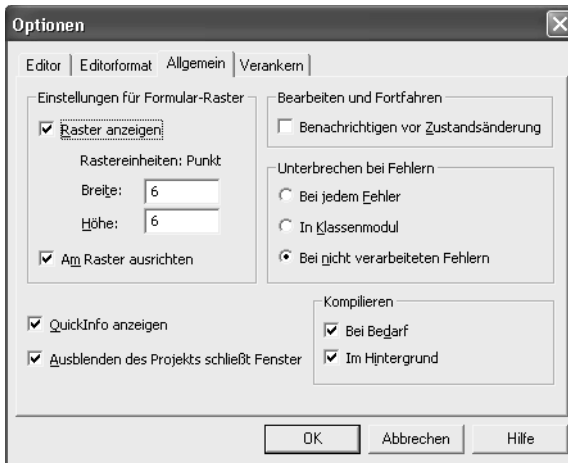
Abbildg. 2.8 Die Optionen, die die Formatierung des Codes beeinflussen



# Allgemeine Optionen

Die Registerkarten *Allgemein* (Abbildung 2.9) und *Verankern* (Abbildung 2.10) bieten wenig, was in diesem Kapitel unbedingt diskutiert werden muss. Die restlichen Einstellungen sind selbsterklärend oder werden in der Hilfe entsprechend vorgestellt.

**Abbildg. 2.9** Die Registerkarte *Allgemein* enthält Optionen für verschiedene Bereiche wie UserForms (Formulare) usw.



**Abbildg. 2.10** Darf das jeweilige Fenster am Rand des Anwendungsfensters bzw. eines anderen verankerbaren Fensters positioniert werden?



# Die VBA-Hilfe – eine versteckte Schatzkammer

Als Office-Anwender sind wir oft versucht zu sagen »Die Hilfe braucht Hilfe«. Das Hilfeformat wurde für jede Office-Version der letzten zehn Jahre regelrecht umgekrempelt. Dies gilt gleichermaßen für deren Inhalt sowie Struktur. Leider können wir nicht behaupten, dass diese Änderungen eine echte Verbesserung bewirkt hätten. In Office 2003 bauen die Benutzer- und die VBA-Hilfe-Schnittstellen sogar auf unterschiedlichen Technologien auf.

In diesem Abschnitt werden wir eine Übersicht der VBA-Hilfe-Schnittstellen in Word 2000 bis 2003 vorstellen. Insbesondere werden Möglichkeiten, die Hilfe aufzurufen und Informationen zu finden, diskutiert.

Info-Menü

Allen Versionen gemeinsam ist das Info-Menü (?), in dem sich der Eintrag *Microsoft Visual Basic-Hilfe* befindet. Bei Auswahl dieses Menüpunkts wird die Hilfe (sofern sie installiert wurde) gestartet, deren Ergebnis sich bei den drei Versionen jedoch grundsätzlich unterscheidet:

- **Word 2000:** Das Hilfefenster öffnet sich am rechten Rand und das des VB-Editors wird entsprechend verschmälert. Die Anzeige flimmert und hüpfert bei der Ein- und Ausblendung der Hilfe. Das Hilfefenster darf verschoben und verkleinert werden und, hat man das einige Male getan, wird es schließlich künftig so geöffnet.

Das VBA-Hilfefenster ist dreiteilig: Oben befindet sich eine Symbolleiste, links darunter ein Fenster mit den Registerkarten *Inhalt*, *Antwort-Assistent* sowie *Index*, und rechts wird der Text zu dem im linken Fenster ausgewählten Thema angezeigt. Informationen können in jeder der drei Registerkarten gesucht werden. Bei Bedarf kann der Teil mit den Registerkarten ein- und ausgeblendet werden.

- **Word 2002:** Das Hilfefenster in Word 2002 ist dem von Word 2000 ähnlich, enthält aber zusätzlich eine Symbolschaltfläche, worüber gewählt werden kann, ob es frei stehend oder rechts neben dem VB-Editor zu positionieren ist.

In der Registerkarte *Inhalt* werden die Programmierreferenzen für alle in *Extras/Verweise* aktivierte Anwendungsbibliotheken aufgelistet, sobald durch Drücken von **[F1]** die Hilfe für ein Thema aus dieser Bibliothek aufgerufen wurde.

- **Word 2003:** Statt eines eigenständigen Fensters wird der Aufgabenbereich *Visual Basic-Hilfe* eingeblendet. Im oberen Teil befindet sich ein Textfeld für den Suchbegriff, darunter eine Liste von Programmierreferenzen: *Microsoft Word Visual Basic Referenz*, *Microsoft Visual Basic Documentation*, *Microsoft Office Visual Basic Referenz*. Der Teil mit den Registerkarten fehlt.

Die Liste der Programmierreferenzen ist nicht erweiterbar, egal ob ein aktiver Verweis zu einer Anwendungsbibliothek vorliegt.

Bei Auswahl eines Themas wird ein separates Hilfefenster geöffnet, das entweder am rechten Rand neben dem Aufgabenbereich eingefügt wird (der VB-Editor wird entsprechend schmaler) oder über dem anderen Fenster liegt.

Frage hier eingeben

Zusätzlich gibt es seit Word 2002 oben rechts in der Menüleiste das Feld *Frage hier eingeben*. Der Benutzer gibt einen Begriff ein, drückt die **[↵]**-Taste und findet die Ergebnisse der Suche in einer Liste unterhalb des Feldes. Das ist etwas gewöhnungsbedürftig, doch die Idee ist nicht schlecht, da für die Umsetzung nur wenig Platz auf dem Bildschirm beansprucht wird.

Word 2003 hat dieses Eingabefeld in der Menüleiste beibehalten, jedoch wurde es hier fast überflüssig, da bereits im Hilfefenster integriert ist. Gefundene Themen erscheinen als Links im Aufgabenbereich *Suchergebnisse* statt in einer Liste unter dem Feld.

Nach-  
schlagen

Das Auffinden der gesuchten Informationen innerhalb der VBA-Hilfe ist nicht immer einfach, selbst wenn man über ausgezeichnete Englisch-Kenntnisse verfügt. In den älteren Word-Versionen führte das Blättern im Index manchmal zur gesuchten Information.

Da diese Registerkarte in Word 2003 entfernt wurde, bestehen nur dann realistische Chancen, die benötigten Angaben zu finden, wenn der Name eines Objekts, einer Eigenschaft oder einer Methode bekannt ist. Aus diesem Grund beginnt dieses Buch mit einem Kapitel zum Thema Makrorekorder, denn er kann den Begriff liefern, der als Schlüssel zum Hilfetext dient.

Geben Sie den Begriff in ein Suchfeld ein oder positionieren Sie den Mauszeiger innerhalb des Begriffs im Code-Fenster und drücken Sie **[F1]**. Das Hilfefenster öffnet sich und müsste das Thema zum Begriff anzeigen. Wir schreiben »müsste«, weil es leider vorkommt, dass das Hilfefenster leer bleibt. Passiert das beim Drücken von **[F1]**, ist es möglich, dass der Weg über ein Suchfeld mehr Erfolg hat. Oder auch umgekehrt.

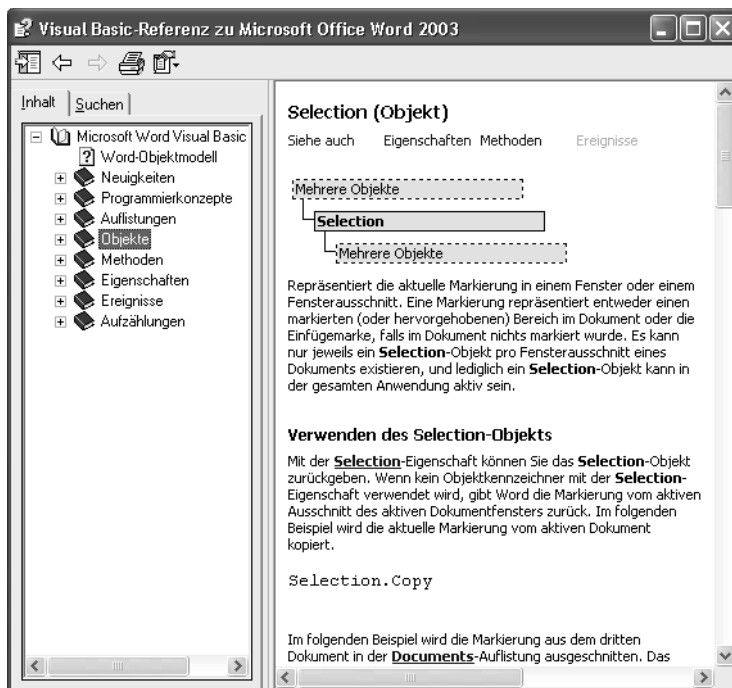
Sie haben aber immer die Möglichkeit, die Hilfedateien direkt zu öffnen. Noch besser: diese verfügen über die Registerkarten *Inhalt* und *Suchen*, welche in Office 2003 nicht mehr angezeigt werden (Abbildung 2.11). Hilfedateien haben die Dateinamenerweiterung \*.*chm*. Sie befinden sich standardmäßig in einem sprachspezifischen Unterordner zum Ordner, in dem die \*.*exe*-Dateien liegen, beispielsweise in *C:\Programme\Microsoft Office\OFFICE11\1031*. In Tabelle 2.1 finden Sie eine Namensliste der Office-Hilfedateien.

**Tabelle 2.1** Die Hilfedateien der Microsoft Office-Anwendungen

Anwendung	Hilfedatei
Microsoft Access	<i>ACMAIN11.CHM</i> (Benutzerschnittstelle) <i>VBAAC10.CHM</i> (Entwicklerreferenz)
Microsoft Graph	<i>GRAPH10.CHM</i> (Benutzerschnittstelle) <i>VBAGR10.CHM</i> (Entwicklerreferenz)
InfoPath	<i>IPMAIN11.CHM</i> (Benutzerschnittstelle) <i>INFMAIN.CHM</i> (Benutzerschnittstelle) <i>INFREF.CHM</i> (Entwicklerreferenz)
OLAP	<i>MCE.CHM</i>
Document Imaging	<i>MSPHELP.CHM</i>
MS Query	<i>MSQRY32.CHM</i>
Microsoft Clip Organizer	<i>MSTORE10.CHM</i>
Microsoft Office Picture Manager	<i>OISMAIN.CHM</i>
Microsoft Outlook Formulare	<i>OLFM10.CHM</i>
Microsoft Outlook	<i>OLMAIN11.CHM</i> (Benutzerschnittstelle) <i>VBAOL11.CHM</i> (Entwicklerreferenz)
Microsoft PowerPoint	<i>PPMAIN10.CHM</i> (Benutzerschnittstelle) <i>VBAPP10.CHM</i> (Entwicklerreferenz)
Microsoft Aktenkoffer-Replikation	<i>RPLBRF35.CHM</i>
Microsoft Office	<i>VBAOF11.CHM</i> (Entwicklerreferenz)
Microsoft Office Publisher	<i>PBMAIN10.CHM</i> (Benutzerschnittstelle) <i>VBAPB10.CHM</i> (Entwicklerreferenz)

**Tabelle 2.1** Die Hilfedateien der Microsoft Office-Anwendungen (Fortsetzung)

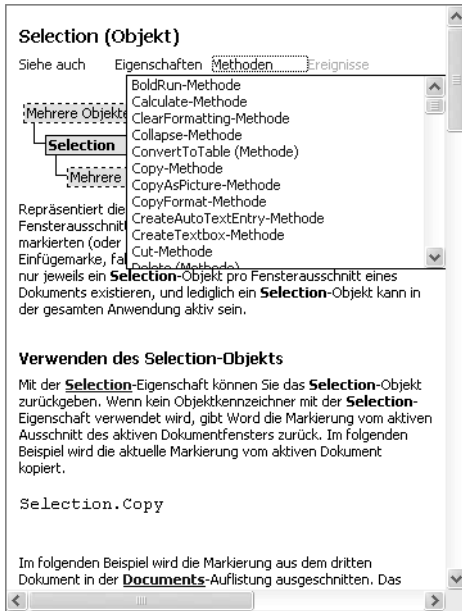
Anwendung	Hilfedatei
Microsoft Office Word	WDMAIN11.CHM (Benutzerschnittstelle) VBAWD10.CHM (Entwicklerreferenz)
Microsoft Office Excel	XLMAIN11.CHM (Benutzerschnittstelle) VBAXL10.CHM (Entwicklerreferenz) XLADDIN.CHM (Entwicklerreferenz) XLMACRO.CHM (Entwicklerreferenz)
Microsoft XML Parser	XMLSDK5.CHM (Entwicklerreferenz)
Hilfedateien für die Office-Programmierung befinden sich standardmäßig unter C:\Programme\Gemeinsame Dateien\Microsoft Shared\VBA\VBA6\1031	
Visual Basic Umgebung (enthält eine Schnittstelle zu den folgenden Bibliotheken)	VBUI6.CHF
Formulare (UserForms)	FM20.CHF
Visual Basic allgemein (Theorie)	VBCN6.CHF
Visual Basic allgemein (Begriffslexikon)	VBENDF98.CHM
Visual Basic allgemein (Debugging)	VBHW6.CHF
Visual Basic allgemein (Objektmodell)	VBRL6.CHF
Visual Basic Editor	VBOB6.CHF

**Abbildg. 2.11** Das Hilfenfenster der Word-Hilfedatei VBAWD10.CHM




Haben Sie das Hilfethema zu einem bestimmten Objekt gefunden, befinden sich darin meist auch Listen zu dessen Eigenschaften und Methoden, wie in Abbildung 2.12 ersichtlich. Das Anwählen eines dieser Einträge bewirkt einen Sprung zum nächsten Hilfethema. Zudem finden Sie weitere nützliche Links im Hilfetext. Obwohl die Hilfeseiten im HTML-Format vorliegen, steht im Kontextmenü eines Hilfenfensters der Befehl »In neuem Fenster öffnen« leider nicht zur Verfügung.

Abbildg. 2.12 Sie finden auf der Hilfeseite nützliche Links zu verwandten Themen



#### HINWEIS

Wenn Sie in der .NET-Umgebung entwickeln, stehen die VBA-Hilfdateien über die üblichen .NET-Hilfeschrittstellen nur dann zur Verfügung, wenn Sie Visual Studio Tools for Office installiert haben. Sonst müssen Sie die Hilfe-Dateien direkt öffnen oder sie über die Anwendungsoberfläche (VB Editor in Word) einsehen.

Die Hilfe-Dateien stehen auch auf der MSDN-Webseite (allerdings in englischer Sprache) auch als Download zur Verfügung: [http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vbawd11/html/WordVBAAWelcome\\_HV01135786.asp](http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vbawd11/html/WordVBAAWelcome_HV01135786.asp).

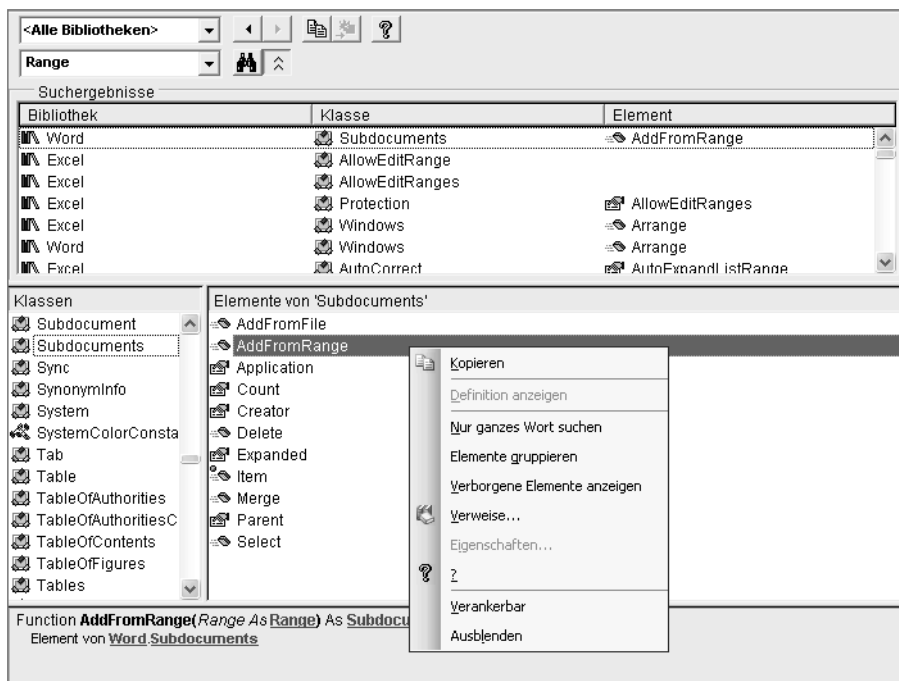
## Wo alle Fäden zusammenlaufen: Der Objektkatalog



Allzu oft kommt es vor, dass wir uns nur an einen Teil des gesuchten Begriffs oder Objektnamens erinnern. Da versagt die Suchfunktion in der Hilfe. Es steht uns aber ein Werkzeug zur Verfügung, das auch mit vagen Erinnerungsbruchstücken etwas anzufangen weiß: der Objektkatalog. Aufgerufen wird er über die nebenstehend abgebildete Symbolschaltfläche oder durch Drücken der Taste **F2**.


Wie in Abbildung 2.13 ersichtlich, bietet der Objektkatalog eine Übersicht der zugehörigen Eigenschaften und Methoden (Elemente) eines Objekts (Klasse). Im oberen Bereich befinden sich rechts einige Symbolschaltflächen für die Bedienung, wichtiger jedoch sind die beiden Dropdown-Listen links daneben. Die obere listet jede geladene Objektbibliothek auf und bietet die Möglichkeit, eine Suche entweder für *Alle Bibliotheken* oder aber nur eine bestimmte, ausgewählte durchzuführen.

**Abbildg. 2.13** Der Objektkatalog bietet einen Zugriff auf die Objekthierarchie aller geladenen Bibliotheken



### HINWEIS

Um weitere Objektbibliotheken zu laden, müssen Sie einen Verweis zu diesen setzen, was über das Kontextmenü oder *Extras/Verweise* erfolgen kann. Zum Thema Verweise erfahren Sie mehr in Kapitel 8.

Den zu suchenden Begriff geben Sie in das untere Feld ein und betätigen dann die -Taste oder klicken auf die Fernglas-Symbolschaltfläche. (Der Dropdown-Teil speichert früher gesuchte Einträge aus der gleichen Sitzung.)

Die Suchergebnisse erscheinen im zweiten Teil des Objektkatalogs. Alle Elemente, in denen die gesuchte Zeichenkette enthalten ist, werden rechts in der dritten Spalte aufgelistet. Das Objekt (Klasse), dem sie angehören, befindet sich in der Spalte links daneben, und die zugehörige Anwendung (Bibliothek) wird in der ersten Spalte angezeigt. Im abgebildeten Beispiel ist sowohl ein Verweis auf die Excel-Objektbibliothek wie auch auf die von Word aktiv, und da das Range-Objekt in beiden Objektmodellen vorkommt, sind gemischte Einträge verzeichnet.

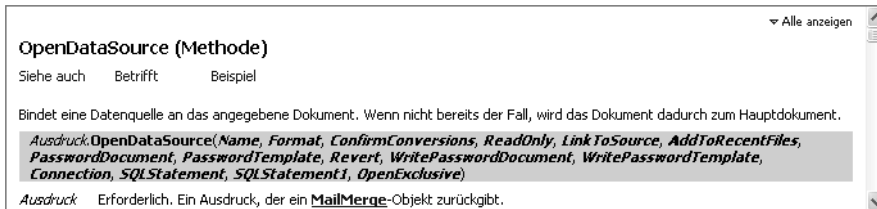
Durch Anklicken eines Eintrags im Fenster *Suchergebnisse* wird auf der linken Seite im darunter liegenden Fenster das Objekt (Klasse) ausgewählt. In der rechten Spalte erscheinen alle zugehörigen

Eigenschaften und Methoden (Elemente). Im untersten Teil finden Sie weitere Angaben zum Element – beispielsweise den Wert, der zurückgegeben wird – oder, wie in Abbildung 2.13, die Prozedursyntax. Die unterstrichenen Begriffe dienen jeweils als Hyperlink, worüber weitere Informationen in der Liste *Elemente* angezeigt werden.

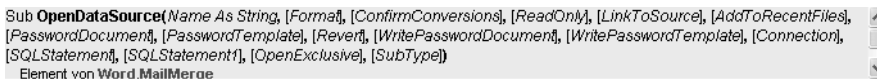
Sie gelangen zur Hilfe für ein markiertes Element durch Drücken der Taste **[F1]** oder über das Kontextmenü.

Ein großer Vorteil des Objektkatalogs besteht darin, dass er, im Gegensatz zur Hilfedokumentation, direkt auf die Definitionen in der Objektbibliothek zugreift. Falls Sie für ein Element Unterschiede zwischen den Angaben des Objektkatalogs und der Hilfedokumentation bemerken, verlassen Sie sich ohne zu zögern auf den Objektkatalog. Das Paradebeispiel dafür finden wir in Word 2002 und 2003. Vergleichen Sie die Abbildung 2.14 mit der Abbildung 2.15. Sehen Sie das *SubType*-Argument in der letzteren? Da die Hilfedokumentation von Menschenhand erstellt wird, unterlief ein Fehler und es wurde vergessen, das Argument zu dokumentieren. Der Objektkatalog aber listet automatisch alle Elemente auf, die er in der Objektbibliothek findet.

**Abbildg. 2.14** Die Methodenunterschrift für die *OpenDataSource*-Methode laut Hilfedokumentation



**Abbildg. 2.15** Die Methodenunterschrift für die *OpenDataSource*-Methode laut Objektkatalog, einschl. *SubType*



Ebenfalls sehr interessant ist der Eintrag *Verborgene Elemente anzeigen* im Kontextmenü. In der Spannung zwischen korrekten COM-Programmierungsregeln und dem Konzept der Rückwärtskompatibilität kommt es gelegentlich vor, dass die Microsoft-Entwickler eine Methode mit einer ganz neuen ersetzen. Die alte wird aber nicht entfernt, sondern umbenannt, wie in Abbildung 2.16 ersichtlich. Falls Ihre Prozeduren in einer neuen Word-Version andere Ergebnisse liefern, kontrollieren Sie, ob eine Methode oder Funktion geändert wurde. Finden Sie eine umbenannte Version, testen Sie sie in Ihrem Code.

### Ein Tipp für Profis

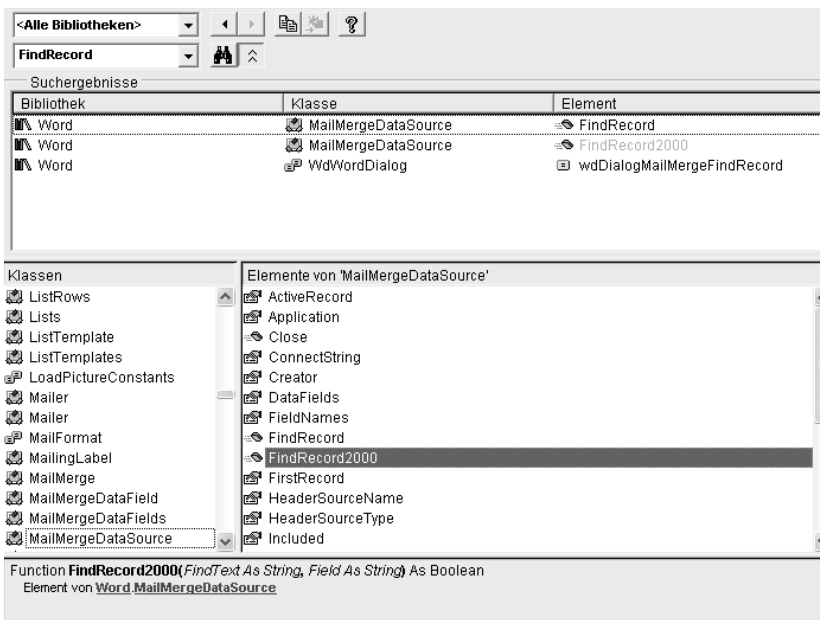
Eigentlich befasst sich dieses Buch nicht mit der Theorie der Programmierung. Aber auch auf dieser Ebene sind Lehren aus der Fehlüberlegung, die hinter FindRecord2000 steht, zu ziehen. Deshalb wird hier etwas ausführlicher beschrieben, was hinter den verborgenen Einträgen steckt und warum sie problematisch sind. ►

Die allgemeinen Regeln für die COM-Programmierung schreiben vor, dass eine Methode, worauf andere Programmierer zugreifen können, nie auf eine Art und Weise geändert werden darf, die die Rückwärtskompatibilität verletzt. Andererseits stehen Anwendungsentwickler unter enormen Druck, die Benutzerschnittstelle möglichst unverändert zu lassen. Gelegentlich führt der Balance-Akt zwischen diesen zwei Geboten zu echten Ungeheuern.

Im Fall der FindRecord-Funktionalität war sie in Word 2000 sowohl in der Benutzer- wie auch in der VBA-Schnittstelle fehlerhaft. Nach einer erfolgreichen Suche konnte ein Seriendruck nicht mehr zusammengeführt werden, man musste zuerst eine *unerfolgreiche* Suche durchführen, um den Seriendruck wieder zu aktivieren. In Version 2002 wurde die Funktionalität für den Benutzer geändert und die alte Methode ersetzt. Diese wurde jedoch in der Objektbibliothek immer noch mitgeführt, verborgen und mit dem Namen FindRecord2000 versehen. Nur stellte es sich leider heraus, dass, obwohl die Suche nach Datensätzen fortan in der Benutzerumgebung funktionierte, diese in VBA vollständig ergebnislos blieb. FindRecord funktionierte also gar nicht, FindRecord2000 dagegen wie früher. Nur hatte der Word-Entwickler keine Ahnung, was passiert war bzw. dass die alte Funktionalität unter diesem Namen vorhanden war.

Strikt nach den Regeln hätten die zuständigen Entwickler für die geänderte Funktionalität eine zusätzliche Methode, neben der existierenden, bereitstellen müssen. Der Vorsatz der Rückwärtskompatibilität wurde falsch interpretiert: Der Methodenname wurde beibehalten, aber die Funktionalität geändert, so dass Lösungen, die um das alte Problem arbeiteten, nicht mehr lauffähig waren. Auch wurde die Änderung nie veröffentlicht, so dass der Entwickler auf die Idee gekommen wäre, FindRecord in seinem Code mit FindRecord2000 zu ersetzen. (Obwohl dies nur eine Notlösung wäre, da er zwei verschiedene Lösungen – eine für Word 2000 und eine für Word 2002/2003 – bereitstellen müsste, was nicht unproblematisch ist.)

Abbildg. 2.16 In Word 2002 gibt es die Methoden *FindRecord* sowie *FindRecord2000*. Letztere ist normalerweise verborgen.



# The Show must go on: Code bearbeiten

Eine Übersicht zur Umgebung des VB-Editors haben Sie nun erhalten. Also ist es an der Zeit, uns dessen eigentlichem Zweck zu widmen: der Bearbeitung von Code. Wie schon erwähnt, ist das Code-Fenster eine Art Texteditor. Sie können darin tippen, suchen, ersetzen, markieren, kopieren, einfügen und löschen. Erkennt der VB-Editor ein Schlüsselwort oder einen Kommentar, nimmt er entsprechende Zeichenformatierungen vor. Das alles ist sehr logisch und bedarf eigentlich keiner weiteren Erklärung.

Nicht sofort offensichtlich ist aber, wie im Code-Fenster navigiert wird. Die Liste oben rechts mit den Prozedurnamen haben wir schon vorgestellt. Wählen Sie dort einen Eintrag aus, und es wird zu dieser Prozedur gesprungen.

Die Code-Menge kann sehr schnell über den Fensterrand hinaus wachsen und es wird entsprechend schwierig, den Überblick zu behalten. Zudem wird Code oft in getrennten Modulen (der Organisation wegen) aufbewahrt. Wenn Ihr Code eine Funktion oder eine Methode im gleichen Projekt aufruft, und Sie gerne zu dieser springen würden, klicken Sie in den Prozedurnamen und drücken Sie **↩** + **F2**.

Um zum ursprünglichen Ort zurückspringen, drücken Sie **Strg** + **↩** + **F2**.

## HINWEIS

Eine Liste aller Tastenkombinationen für den VB-Editor finden Sie in der *Hilfe zur Benutzeroberfläche von Visual Basic*.



Sie können auch selbst Sprungziele mit den vom VB-Editor zur Verfügung gestellten Lesezeichen festlegen. Die Befehle dazu befinden sich im Menü sowie in der Symbolleiste *Bearbeiten*. Die Symbolschaltflächen sind nebenstehend aufgelistet. Die erste fügt ein Lesezeichen für die aktuelle Codezeile ein oder entfernt ein vorhandenes. Die zweite springt zum nächsten, die dritte zum vorherigen Lesezeichen. Und die vierte entfernt alle Lesezeichen aus dem Projekt. Ein Lesezeichen erkennen Sie am türkisfarbenen Punkt im linken Fensterrand, wie in Abbildung 2.17 ersichtlich.

Abbildg. 2.17 Der Punkt im linken Rand ist ein Lesezeichen und standardmäßig türkis

```
Sub TabelleVorbereiten2()
    Dim tbl As Word.Table
    Dim rw As Word.Row

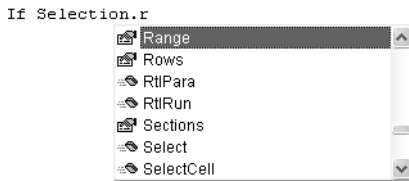
    If Selection.Range.Information(wdWithInTable) Then
        Set tbl = Selection.Tables(1)
        Set rw = tbl.Rows(1)
        rw.Range.Bold = True
        For i = 1 To rw.Cells.Count
            rw.Cells(i).Range.Text = "Spalte " & CStr(i)
        Next
        tbl.Cell(2, 1).Range.Select
        Selection.Collapse
    End If
End Sub
```

In Kapitel 1 haben wir den Makrorekorder und aufgezeichneten Beispiel-Code vorgestellt. Bestimmt erinnern Sie sich an die dort beschriebenen Nachteile des Resultats und die Notwendigkeit, aufgezeichneten Code anpassen zu müssen. Die Prozedur in Abbildung 2.17 entspricht dem Listing aus Kapitel 1. Es ist natürlich einfach zu sagen, man müsse den Code anpassen. Aber woher sollen Sie wissen, *was* zu schreiben ist? Muss man lange Zeilen wie `If Selection.Range.Information(wdWithinTable) Then` auswendig können?

Die Antwort ist »Nein, nicht ganz«. Code zu schreiben, ist ein Zusammenspiel von mehreren Faktoren, und der VB-Editor hilft dabei, mit »IntelliSense«.

Sobald der Name eines Objekts bekannt ist, können Sie in der Hilfe nachschlagen. Dort steht, wie dieser im Code einzusetzen ist (die Syntax) und welche Eigenschaften und Methoden zur Verfügung stehen. Wenn es darum geht, den Code zu schreiben, fängt man mit dem Objektnamen an und gibt unmittelbar danach einen Punkt ein. Der VB-Editor reagiert auf die Eingabe des Punkts mit einer Liste von gültigen Eigenschaften und Methoden, wie in Abbildung 2.18 dargestellt.

**Abbildg. 2.18** Das »IntelliSense« des VB-Editors hilft beim Code-Schreiben



Sie können mit der Bildlaufleiste durch diese Liste blättern oder einfach weitertippen. Die Markierung wird automatisch zum passenden Eintrag springen. Durch Drücken der **[Tab]**- oder **[Enter]**-Taste wird der Vorschlag übernommen.

#### **TIPP**

Für alle »IntelliSense«-Listen gilt: Drücken Sie **[Esc]**, um die Liste ungenutzt zu schließen.



Vergessen Sie nicht, Ihr Projekt regelmäßig zu speichern. Empfehlenswert ist das Abspeichern des Programmcodes vor jeder Testausführung. Wie in einer normalen Anwendung geschieht dies über den Menübefehl *Datei/Speichern*, mit der Tastenkombination **[Strg] + [S]** oder über die nebenstehend gezeigte Symbolschaltfläche.

Bitte beachten Sie, dass damit nur die Word-Datei, die das Makro enthält, gespeichert wird. Haben Sie beispielsweise ein Makro in der *Normal.dot* aufgezeichnet und bearbeitet, wird die *Normal.dot* und nicht das aktuelle Dokument gespeichert. Speichern Sie umgekehrt das aktuelle Dokument, wird der Makro-Code in der *Normal.dot* oder einem anderen zum Dokument gehörenden Projekt nicht gespeichert.

In Kapitel 1 wurde beschrieben, wo Makros gespeichert sind und wie Sie sie über den Word-Menübefehl *Extras/Vorlagen und Add-Ins/Organisieren* verwalten können. Es gibt einen weiteren Weg, Makro-Code auszutauschen sowie Sicherheitskopien zu erstellen: über den Menübefehl *Datei/Datei exportieren* des VB-Editors. Im Gegensatz zu den in Kapitel 1 erwähnten Methoden erstellt die Export-Funktionalität eine reine Textdatei. Standardmodule erhalten die Dateinamenerweiterung *\*.bas*, Klassenmodule *\*.cls*. Für Formulare werden zwei Dateien erstellt: *\*.frm* und *\*.frx* (letztere enthält binären Code, der die OLE-Elemente des Formulars definiert).

Eine solche Datei wird über den Menübefehl *Datei/Datei importieren* in ein VBA-Projekt eingefügt.

**TIPP**

Word speichert VBA-Code in den internen Dokumentstrukturen. Werden diese beschädigt, kann es vorkommen, dass Word den Code nicht mehr korrekt verwalten kann. Dieser Umstand führt zu immer größeren Dateien und kann im schlimmsten Fall zum Dokumentabsturz führen. Wenn Sie vermuten, ein solches Problem liege vor, oder Sie haben viel an dem Code gearbeitet, entfernen Sie alle Code-Module über *Datei/Entfernen von <Modulname>*, speichern die Datei ab und importieren die soeben exportierten Module in das Projekt. Durch die Konvertierung in reinen Text werden unerwünschte Überreste entfernt. (Es steht ein Werkzeug zur Verfügung, das dieses Vorgehen automatisiert. Den VBA Code Cleaner können Sie über die Webseite <http://word.mvps.org/downloads/index.htm> auf Ihren Rechner herunterladen.)

## Zusammenfassung

Dieses Kapitel war eine kurze Übersicht der VBA-Arbeitsumgebung. In den folgenden Kapiteln erfahren Sie mehr über den Code, den Sie hier schreiben und bearbeiten werden.

- Ein erster Abschnitt behandelte den Visual Basic-Editor. Es wurden die einzelnen Fensterbereiche (Seite 54 ff.) und die Optionen des Editors (Seite 61 ff.) vorgestellt.
- Es wurde auf die Online-Hilfe hingewiesen und aufgezeigt, wie diese genutzt werden kann (Seite 66). Als weiteres interessantes Hilfsmittel wurde der Objektkatalog vorgestellt (Seite 69).
- Im letzten Abschnitt wurde das Bearbeiten des Codes behandelt (Seite 73).





## Kapitel 3

# VBA-Grundlagen

**In diesem Kapitel:**

Variablen	78
Konstanten	91
Benutzerdefinierte Typen	93
Nützliche VBA-Funktionen	100
Bedingungen	105
Schleifen	108
Code im VB-Editor debuggen	111
Fehlerbehandlung	118
Dateisystem-Operationen	125
Zusammenfassung	136

In diesem Kapitel versuchen wir Ihnen die Grundlagen zu Visual Basic für Applikationen (VBA) näher zu bringen. Sollten Sie im Umgang mit einer Programmiersprache wenig oder gar keine Erfahrung besitzen, vermitteln Ihnen die folgenden Abschnitte einen kurzen Überblick über die wichtigsten Punkte von VBA:

- Die Deklaration und der Umgang mit Variablen, die verschiedenen Typen von Variablen, deren Gültigkeit innerhalb des Projekts und die Übergabe von Werten an eine Funktion werden im Abschnitt »Variablen« in diesem Kapitel behandelt.
- Der Nutzen von Konstanten, die Deklaration derselben und deren Gültigkeit innerhalb des Projekts werden ebenfalls in diesem Kapitel im Abschnitt »Konstanten« dargelegt.
- Die Deklaration und der Umgang mit eigenen Datentypen, das Erstellen und Anwenden von eigenen Aufzählungen, so genannten Enumerationen, werden im Abschnitt »Benutzerdefinierte Typen« in diesem Kapitel erörtert.
- Die Programmverzweigungen werden im Abschnitt »Bedingungen« und die Programmwiederholungen im Abschnitt »Schleifen« näher betrachtet.
- Im Abschnitt »Code im VB-Editor debuggen« wird aufgezeigt, wie das Verhalten der erstellten Programmsequenzen analysiert wird. Und im Abschnitt »Fehlerbehandlung« wird auf das Behandeln von möglichen Programmfehlern eingegangen.
- Einfache Beispiele mit Operationen am Dateisystem runden dieses Kapitel ab. Die entsprechenden Einsatzmöglichkeiten werden im Abschnitt »Dateisystem-Operationen« behandelt.

**HINWEIS**
**Hinweis zu .NET**

Falls Sie als .NET-Entwickler VBA-Code portieren müssen oder Fragen zu VB-Datentypen haben, achten Sie vor allem auf die Hinweise in diesem Kapitel.

Ferner machen wir auf die MSDN-Themen »Converting Office VBA Help Code Examples to Visual Basic .NET and C#« und »Language Changes in Visual Basic« aufmerksam. Während der Erstellung dieses Buchs waren die beiden Artikel unter folgenden Adressen zu finden:

- [http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dv\\_wrcore/html/wrtchconvertingcodeexamplesfromofficevbahelptovisualbasicnetc.asp](http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dv_wrcore/html/wrtchconvertingcodeexamplesfromofficevbahelptovisualbasicnetc.asp)
- <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vbcn7/html/vaconDifferencesBetweenVB6AndVB7.asp>

## Variablen

Eine Variable bedeutet gemäß Lexikon eine »veränderliche Größe«. Innerhalb eines Computerprogramms werden Variablen zum Zwischenspeichern von einzelnen Werten und Objekten verwendet. Auf diese Weise kann die gleiche Programmsequenz, beispielsweise eine Addition, irgendwelche Werte zusammenzählen. Die entsprechenden Werte für die einzelnen Variablen, in diesem Fall die beiden Summanden, müssen vorab zugewiesen werden.

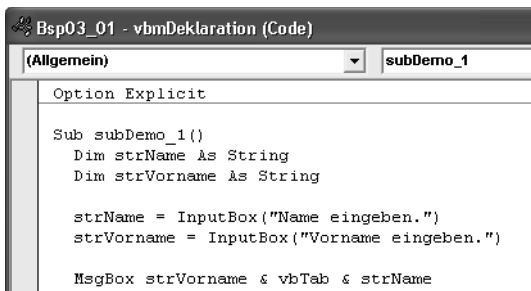
```
intZahlA = 1
intZahlB = 100
intZahlC = intZahlA + intZahlB
```

## Standard-Datentypen

In VBA stehen unterschiedliche Datentypen zur Verfügung. Mit Ausnahme des Datentyps Variant können sie nur Daten einer bestimmten Menge aufnehmen. Die wichtigsten Datentypen sind nachstehend zusammengefasst. Weitere Informationen finden Sie in der Online-Hilfe von VBA zum Thema »Datentypen (Zusammenfassung)«.

**WICHTIG** Für VBA verfolgen die Autoren das Prinzip, die Deklaration sämtlicher Variablen grundsätzlich am Anfang einer Prozedur bzw. Funktion vorzunehmen. So bleibt die Programmsequenz strukturiert und übersichtlich, und beispielsweise während einer Suche nach Programmfehlern sind alle zugewiesenen Datentypen auf einen Blick sichtbar.

Abbildg. 3.1 Deklarieren Sie alle Variablen am Anfang einer Prozedur



In anderen Programmiersprachen (etwa C#) werden die Variablen oft erst deklariert, wenn sie gebraucht werden. In diesen Fällen wird die Gültigkeit durch die Programmstruktur feiner reguliert, als es die klassischen VB-Sprachen tun. Beispielsweise ist dort eine in einer Do-Schleife deklarierte Variable außerhalb der Schleife ungültig.

**String** Variablen vom Datentyp String enthalten Zeichenfolgen, bei denen zwei Arten zu unterscheiden sind: jene mit variabler Länge und solche mit fester Länge. Als Typkennzeichen für String dient das Dollarzeichen (\$). Der Standardwert ist eine leere Zeichenkette ("").

```

Dim strText1 As String
Dim strText2 As String * 50
Dim strText3$

```

**HINWEIS** Da das .NET-Framework Zeichenfolgen mit *fester* Länge allgemein nicht unterstützt, raten wir davon ab, mit solchen Variablen zu arbeiten, wenn der Code möglicherweise einmal in eine .NET-Sprache portiert werden muss.

.NET-Entwickler machen wir auf das mögliche Vorhandensein von Deklarationen für Zeichenfolgen mit fester Länge aufmerksam. Diese werden Sie anpassen müssen.

**Boolean** Variablen vom Datentyp Boolean können nur die beiden logischen Werte True (Wahr) oder False (Falsch) annehmen. Der Standardwert ist False.

```

Dim bStatus As Boolean

```

**HINWEIS** Manche Office-Anwendungen setzen im Hintergrund Ganzzahlwerte für »Wahr« und »Falsch« ein. Dabei steht der Wert 0 (Null) immer für »Falsch«, während für »Wahr« sowohl –1 als auch 1 Verwendung finden. Noch schlimmer, Microsoft könnte den »Wahr«-Wert zwischendurch geändert haben. Größte Aufmerksamkeit ist also bei Code geboten, der boolesche Werte auf »Wahr« testet.

**Integer** Variablen vom Datentyp Integer enthalten nur ganze Zahlen, und zwar im Bereich von –32.768 bis 32.767. Deshalb eignet sich dieser Datentyp in erster Linie für Aufzählungswerte<sup>1</sup>. Als Typkennzeichen für Integer dient das Prozent-Zeichen (%). Der Standardwert ist Null.

```
Dim intWert1 As Integer
Dim intWert2%
```

**Long** Variablen vom Datentyp Long enthalten – wie Integer – ebenfalls nur ganze Zahlen. Jedoch ist der Zahlenbereich hier um ein Mehrfaches größer (–2.147.483.648 bis 2.147.483.647). Insofern ist dieser Datentyp für ganze Zahlen vorzuziehen. Als Typkennzeichen für Long dient das Et-Zeichen (&). Der Standardwert ist Null.

```
Dim lngZahl1 As Long
Dim lngZahl2&
```

**HINWEIS** In der .NET-Umgebung werden für Ganzzahl-Datentypen zwar die gleichen Namen verwendet, jedoch mit anderen Werten. Der klassische VB-Datentyp Integer (16 Bit) entspricht dem .NET-Datentyp Short, Long (32 Bit) entspricht dem .NET-Datentyp Integer, und neu ist in .NET der Zahlenbereich für den Datentyp Long mit 64 Bit.

**Single** Variablen vom Datentyp Single enthalten Gleitkommazahlen mit einfacher Genauigkeit. In den meisten Fällen ist diese Genauigkeit für die Berechnung innerhalb eines Programms ausreichend. Deshalb wird die Verwendung dieses Datentyps für Gleitkommazahlen empfohlen. Als Typkennzeichen für Single dient das Ausrufezeichen (!). Der Standardwert ist Null.

```
Dim sngZahl1 As Single
Dim sngZahl2!
```

**Variant** Variablen vom Datentyp Variant können beliebige Daten enthalten – ausgenommen String-Variablen fixer Länge sowie benutzerdefinierte Datentypen. Variant wird für alle Variablen verwendet, denen nicht explizit ein anderer Datentyp zugewiesen wird. Ein Typkennzeichen für diesen Datentyp existiert nicht. Der Standardwert ist Empty (Leer).

```
Dim varInhalt1 As Variant
Dim varInhalt2
```

<sup>1</sup>. Ein Aufzählungswert besteht aus einer endlichen Menge eindeutiger ganzer Zahlen. Jede dieser Zahlen hat im verwendeten Kontext eine spezielle Bedeutung. Dies erlaubt eine einfache Auswahl aus einer bestimmten Anzahl von Möglichkeiten, z.B. 0 = Sonntag, 1 = Montag usw.

**HINWEIS** Die .NET-Umgebung unterstützt den Datentyp `Variant` nicht. Stattdessen muss für Variablen undefinierten Inhalts der Datentyp `Object` deklariert werden.

**Object** Variablen vom Datentyp `Object` enthalten Speicheradressen und verweisen auf die entsprechenden Objekte der Anwendung. Die Zuweisung zur Variablen erfolgt immer über eine `Set`-Anweisung. Wird eine Variable vom Datentyp `Object` deklariert, so erfolgt die Zuweisung an das entsprechende Objekt erst zur Laufzeit. Eine Bindung des Objekts bereits während des Kompilierungsvorgangs kann erreicht werden, indem die Variable mit dem Namen einer bestimmten Klasse deklariert wird. Der Standardwert ist `Nothing`.

Mehr zum Thema »Bindung zur Laufzeit bzw. zur Kompilierungszeit« erfahren Sie in Kapitel 8.

```
Dim objWB1 As Object
Dim objWB2 As Excel.Workbook
```

Den Variablen vom Datentyp `Object` können die Werte bzw. der Verweis auf das entsprechende Objekt nur mit Hilfe der `Set`-Anweisung zugewiesen werden.

```
Set doc = Documents.Add
```

**HINWEIS** Für die Bezeichnungen der Variablen innerhalb der einzelnen Programmbeispiele wurden spezielle Namenskonventionen eingehalten. Zusätzliche Informationen und Empfehlungen für die Namensgebung von Konstanten finden Sie in Anhang A.

## Gültigkeit bzw. Sichtbarkeit

Wird ein Projekt in verschiedene Prozeduren und Funktionen gegliedert, die sich zusätzlich in unterschiedlichen Programmmodulen befinden, kann durch eine optimale Deklaration der Variablen deren Gültigkeit bzw. deren Sichtbarkeit innerhalb des Projekts beeinflusst werden. VBA kennt drei verschiedene Formen zur Deklaration einer Variablen:

- Prozedur** ■ Auf der Ebene der Prozedur bzw. Funktion. Die Variable kann nur innerhalb der aktuellen Prozedur verwendet werden.
- Modul** ■ Auf der Ebene des Programmmoduls. Die Variable kann innerhalb des ganzen Moduls verwendet werden.
- Projekt** ■ Öffentlich auf der Ebene des Programmmoduls. Die Variable kann innerhalb des gesamten Projekts verwendet werden.

Mit den beiden Schlüsselwörtern `Private` und `Public` wird die Gültigkeit bzw. Sichtbarkeit von Variablen und Konstanten innerhalb des Projekts festgelegt.

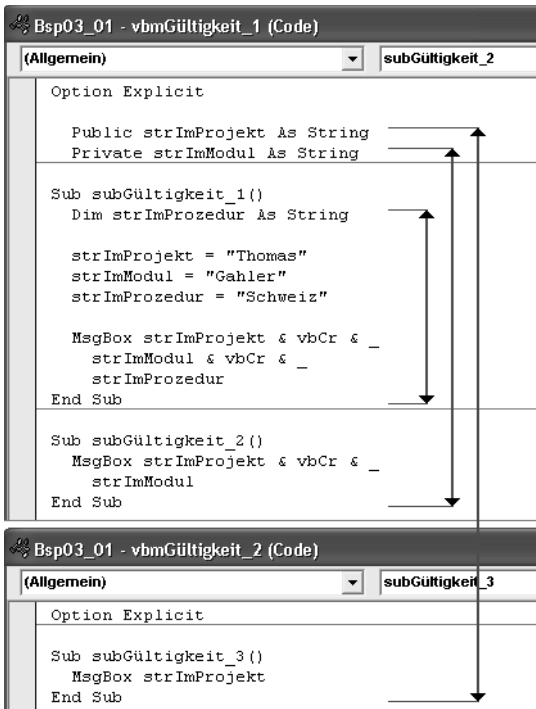
**Private** Wird eine Variable oder Konstante auf Modulebene mit dem Schlüsselwort `Private` deklariert, steht diese dem Programm nur innerhalb des aktuellen Moduls zur Verfügung.

Wird eine Variable oder Konstante auf Modulebene mittels `Dim` deklariert, entspricht diese Zeile einer Deklaration mit dem Schlüsselwort `Private`. Die Gültigkeit ist in diesem Fall ebenfalls auf das aktuelle Modul beschränkt.

**Public** Wird eine Variable oder Konstante auf Modulebene mit dem Schlüsselwort Public deklariert, steht diese innerhalb des ganzen Programms zur Verfügung. Es handelt sich dann um eine öffentliche bzw. globale Variable oder Konstante.

**HINWEIS** Die Schlüsselwörter Private und Public werden auch zur Deklaration von Funktionen und Prozeduren verwendet und wirken sich – wie bei den Variablen und Konstanten – auf deren Gültigkeit bzw. Sichtbarkeit innerhalb des Projekts aus.

Abbildg. 3.2 Übersicht über die Gültigkeit von Variablen innerhalb eines Projekts



Natürlich wäre es am einfachsten, sämtliche Variablen vom Datentyp Variant mit einer Gültigkeit über das *gesamte* Projekt hinweg zu deklarieren. Wir Autoren raten von diesem Vorgehen jedoch dringend ab und empfehlen, stattdessen bei der Deklaration von Variablen die unten genannten Regeln einzuhalten.

Vorteil der *eingeschränkten* Sichtbarkeit von Variablen ist vor allem die Übersichtlichkeit des Programms sowie eine etwas leichtere Suche nach Programmfehlern. Da jede Variable gezielt angelegt werden muss, greifen Sie nicht auf eine Variable zu, die irgendwo innerhalb des Programms angelegt wurde und deren aktueller Inhalt nicht bekannt ist. Auch wird vermieden, dass der Inhalt versehentlich überschrieben und für eine eventuell später geplante Verwendung unbrauchbar wird.

Spielregeln zur Deklaration von Variablen:

- Die Gültigkeit einer Variablen wird auf die jeweils kleinstmögliche Stufe gesetzt. Der Aufbau des Programmcodes wird so gestaltet, dass die Variablen in erster Linie auf Prozedurebene deklariert werden können.

Die Übergabe von Werten an eine Prozedur mittels Argumenten ist gegenüber einer Deklaration mit größerer Gültigkeit vorzuziehen. Zusätzliche Informationen zum Thema »Übergabe von Argumenten an eine Prozedur« finden Sie im Abschnitt »Weiterreichen von Variablen an Prozeduren«.

- Jeder neu deklarierten Variablen wird grundsätzlich ein Datentyp explizit zugewiesen. Der Datentyp `Variant` wird nur in Ausnahmefällen verwendet.
- Sämtliche Variablen werden am Anfang einer Prozedur deklariert.

**HINWEIS**

Wenn Sie versuchen, eine Variable außerhalb ihres Gültigkeitsbereichs zu verwenden, verweigert der Compiler seine Arbeit mit dem Hinweis »Variable nicht definiert«, sofern in der ersten Zeile des Moduls die Deklaration `Option Explicit` gesetzt wurde. Mehr zum Thema »Option Explicit« erfahren Sie in Kapitel 2.

Lebens-  
dauer von  
Variablen

Eine auf Prozedurebene deklarierte Variable »stirbt«, sobald die entsprechende Prozedur abgearbeitet wurde. Der reservierte Speicher wird automatisch freigegeben. Erfolgt ein weiterer Aufruf für die gleiche Prozedur, ist der alte Inhalt der Variablen *nicht* mehr vorhanden, die Variable wird im Speicher neu angelegt.

Freigabe  
von  
Object-  
Variablen

Bei Variablen vom Typ `Object` ist es nicht nur sinnvoll, sondern in vielen Fällen sogar unabdingbar, dass sie nach ihrer Verwendung wieder freigegeben werden. Durch die Verwendung des Schlüsselworts `Nothing` wird die Verbindung zum betreffenden Objekt aufgehoben. Diese Freigabe ist nach einer abgeschlossenen Steuerung einer anderen Applikation zwingend. Mehr zum Thema »Steuerung von anderen Programmen« erfahren Sie in Kapitel 9 und 10.

```
Set doc = Nothing
```

Beachten Sie bei der Freigabe von Objekten, dass diese in umgekehrter Reihenfolge zu deren Zuweisung erfolgt. Auf diese Weise verhindern Sie mögliche Fehler bei der Programmausführung.

**Listing 3.1**

Variablen vom Typ `Object` werden in umgekehrter Reihenfolge freigegeben

```
Sub Demo_Object()
    Dim doc As Word.Document
    Dim tbl As Word.Table
    Dim cel As Word.Cell
    'Dokument instanzieren
    Set doc = Documents.Add(
        Template:=ThisDocument.FullName)
    'Tabelle instanzieren
    Set tbl = doc.Tables(1)
    'Zelle instanzieren
    Set cel = tbl.Cell(1, 1)
    cel.Range.Text = Now
    'Alle Objekte freigeben
    Set cel = Nothing
    Set tbl = Nothing
    Set doc = Nothing
End Sub
```

## Umwandlung von Datentypen

Zur Umwandlung einer Variablen von einem bestimmten Datentyp in einen anderen stellt Ihnen VBA zahlreiche Umwandlungsfunktionen zur Verfügung, von denen die wichtigsten in Tabelle 3.1 aufgeführt sind. Weitere Informationen finden Sie in der Online-Hilfe von VBA zum Thema »Typ-Umwandlungsfunktionen«.

Tabelle 3.1 Die wichtigsten *Typ*-Umwandlungsfunktionen im Überblick

Funktion	Rückgabewert
CStr()	Zeichenkette vom Datentyp <b>String</b>
CBool()	Logischer Wert vom Datentyp <b>Boolean</b>
CInt()	Ganzzahl vom Datentyp <b>Integer</b>
CLng()	Ganzzahl vom Datentyp <b>Long</b>
CSng()	Gleitkommazahl vom Datentyp <b>Single</b>
CVar()	Wert vom Datentyp <b>Variant</b>

Mit Hilfe der Umwandlungsfunktionen können Sie Ihren Programmcode zusätzlich dokumentieren, indem Sie anzeigen, dass das Ergebnis einer Operation einen bestimmten, vom Standarddatentyp abweichenden Datentyp haben soll:

```
strText = CStr(intZahl)
```

In den meisten Fällen ist das Einbinden der Umwandlungsfunktionen aber gar nicht nötig, denn eine Besonderheit von VBA ist das automatische Umwandeln eines Datentyps in einen anderen. Daher liegt die Betonung im vorangegangenen Absatz auf »Dokumentieren des Programmcodes«.

```
strText = intZahl
```

Obwohl die Verwendung dieses Automatismus verlockend erscheint, raten wir Autoren davon ab. Nutzen Sie zur Umwandlung von Datentypen stattdessen die entsprechenden Umwandlungsfunktionen gemäß Tabelle 3.1.

Die Gründe sind nahe liegend: Der Programmcode bleibt übersichtlicher, da die Verwendung der Funktionen eine indirekte Dokumentation des Programms darstellt. Das Resultat einer automatischen Umwandlung liefert nicht in allen Konstellationen den gewünschten Wert zurück. Dies wiederum kann zu Programmfehlern führen, deren Ursache nicht einfach zu finden ist.

---

**HINWEIS** Auch Visual Basic .NET unterstützt den Umwandlungsautomatismus, vorausgesetzt, Option **Strict** ist nicht eingeschaltet.

Das gilt jedoch nicht für C#. Falls Sie VBA-Code auf C# portieren, müssen Sie alle Datentypen explizit mit den entsprechenden .NET-Funktionen umwandeln.

---



### Verknüpfen von Zeichenketten

strText1 &  
strText2

Um zwei Zeichenketten miteinander zu verknüpfen, verwenden Sie grundsätzlich das Et-Zeichen (&). Dieser Operator weist den Compiler an, eine Verknüpfung und nicht etwa eine »Addition« der Zeichenketten auszuführen.

Enthalten die zu verknüpfenden Variablen effektive Zeichenketten (Text), erfolgt die Verknüpfung in beiden Fällen problemlos:

```
strWortA = "Voll"
strWortB = "Mond"
strWortC = strWortA & strWortB      '"VollMond"'
strWortC = strWortA + strWortB      '"VollMond"'
```

Enthalten die zu verknüpfenden Variablen Zahlenwerte statt »echter« Zeichenketten oder ist eine beiden Variablen als Zahl deklariert, erfolgt die Verknüpfung der beiden Variablen unterschiedlich und unabhängig vom Datentyp:

```
strWortA = "12"
intZahlA = 34
strWortC = strWortA & intZahlA      '"1234"'
strWortC = strWortA + intZahlA      '"46"'
```

#### PROFITIPP

Damit sich möglichst wenig Fehler in das Programm einschleichen können, empfehlen wir, grundsätzlich folgende Regeln einzuhalten:

- Jede Umwandlung eines Datentyps erfolgt durch die zugehörige Umwandlungsfunktion.
- String-Variablen werden immer unter Verwendung des Et-Zeichens (&) verknüpft.

## Weiterreichen von Variablen an Prozeduren

Variablen sollten, wann immer möglich, auf Prozedurebene deklariert werden, damit das Programm strukturiert aufgebaut werden kann. Um die Werte dieser Variablen auch in anderen Prozeduren oder Funktionen verwenden zu können, müssen sie als Argumente an die betreffende Funktion übergeben werden.

Diese Argumente können bei der Deklaration der Prozedur auf zwei verschiedene Arten definiert werden:

- |       |  |
|-------|--|
| ByVal | ■ Ein Wert wird als Wert (ByVal) an die Prozedur übergeben. Dies bedeutet, dass innerhalb der Prozedur eine Kopie der Variablen zur Bearbeitung zur Verfügung steht. Der Ursprungswert der eigentlichen Variablen kann durch die Prozedur nicht verändert werden.                          |
| ByRef | ■ Ein Wert wird als Referenz (ByRef) an die Prozedur übergeben. Dies bedeutet, dass die Adresse der Variablen an die Prozedur übergeben wird. Innerhalb der Prozedur steht die Variable zur Bearbeitung zur Verfügung. Der Ursprungswert der eigentlichen Variablen kann verändert werden. |

Ist in der Deklaration der Prozedur nichts angegeben, werden die Argumente als Referenz an die Prozedur übergeben.

**HINWEIS** In der .NET-Umgebung verhält es sich umgekehrt: Standardmäßig werden Argumente als Werte (ByVal) an Prozeduren übergeben. Auch hier ist beim Portieren von VBA-Code nach .NET Vorsicht geboten.

### Argument als Wert übergeben – ByVal

Das folgende Beispiel einer Dreiecksflächenberechnung<sup>1</sup> soll die Übergabe eines Arguments als Wert, also ByVal, verdeutlichen. In Listing 3.2 wird mit der Prozedur subFlächeDreieck im ersten Schritt der Wert der Variablen sngHöhe halbiert, anschließend werden im zweiten Schritt die beiden Strecken multipliziert. Das Resultat ergibt den Flächeninhalt des Dreiecks.

Zur Kontrolle der effektiven Werte enthält die Programmsequenz zwei zusätzliche Bildschirmmeldungen (MsgBox). Sie verdeutlichen, dass in der Hauptprozedur der Wert der Variablen sngHöhe gleich geblieben ist, obwohl diese Variable innerhalb der Prozedur subFlächeDreieck im ersten Berechnungsschritt geändert wurde.

**Listing 3.2** Bei Verwendung von *ByVal* wird die Variable innerhalb der Hauptprozedur nicht verändert

```
Sub Demo_ByVal1()
    Dim sngSeite As Single
    Dim sngHöhe As Single

    sngSeite = 2.5
    sngHöhe = 4
    subFlächeDreieck sngSeite, sngHöhe

    MsgBox sngHöhe, , "Kontrolle A"           'ist 4
End Sub

Public Sub subFlächeDreieck( _
    ByVal sngSeite As Single, _
    ByVal sngHöhe As Single)

    Dim sngFläche As Single

    sngHöhe = sngHöhe / 2
    sngFläche = sngHöhe * sngSeite

    MsgBox "Fläche " & CStr(sngFläche)       'ist 5
    MsgBox sngHöhe, , "Kontrolle B"         'ist 2
End Sub
```

Um den Prozeduraufruf zur Berechnung der Dreiecksfläche so einfach wie möglich zu halten, könnten die Variablen auch auf Modulebene deklariert werden. In Listing 3.3 ist der Beispielcode entsprechend angepasst: Die Deklaration der Variablen erfolgt auf Modulebene.

Auf eine Übergabe der Argumente an die Prozedur subFlächeDreieck kann verzichtet werden, da die Variablen bereits »bekannt« sind. Die eigentliche Berechnung der Fläche erfolgt wieder in zwei Schritten.

<sup>1</sup> Formel zur Berechnung der Dreiecksfläche: Seite \* zugehörige Höhe / 2. Die Reihenfolge der drei Faktoren kann geändert werden und hat auf das Resultat keinen Einfluss.

Zur Kontrolle der effektiven Werte sind auch hier zwei Bildschirmmeldungen (MsgBox) eingebaut. So ist erkennbar, dass sich diesmal der Wert der Variablen `sngHöhe` durch die Division innerhalb der Prozedur `subFlächeDreieck` ebenfalls in der Hauptprozedur geändert hat.

Eine einfache Änderung der Berechnungsformel bewirkt zwar, dass die Werte der beiden Variablen innerhalb der Hauptprozedur nicht mehr geändert werden, die Problematik bleibt aber weiterhin bestehen:

```
sngFläche = sngHöhe * sngSeite
sngFläche = sngFläche / 2
```

Das genannte Beispiel zeigt eindrucksvoll, wie leicht sich ein Programmfehler einschleichen kann. Die Suche nach fehlerhaften Programmzeilen gestaltet sich dann möglicherweise sehr aufwändig, zumal der Fehler teilweise nur unter bestimmten Umständen auftritt.

**Listing 3.3** Ein schlechtes Beispiel mit auf Modulebene deklarierten Variablen

```
Option Explicit
Dim sngSeite As Single
Dim sngHöhe As Single

Sub Demo_ByVal2()
    sngSeite = 2.5
    sngHöhe = 4
    subFlächeDreieck

    MsgBox sngHöhe, , "Kontrolle A" 'ist 2
End Sub

Public Sub subFlächeDreieck()
    Dim sngFläche As Single

    sngHöhe = sngHöhe / 2
    sngFläche = sngHöhe * sngSeite

    MsgBox "Fläche " & CStr(sngFläche) 'ist 5
    MsgBox sngHöhe, , "Kontrolle B" 'ist 2
End Sub
```

### Argument als Referenz übergeben – ByRef

Im nächsten Beispiel können Sie nachvollziehen, wie ein Argument als Referenz, also `ByRef`, übergeben wird.

Mit Hilfe der Prozedur `subGesamtKosten` in Listing 3.4 werden fiktiv Gesamtkosten für den Einkauf berechnet. Zunächst wird vom Einzelpreis der feststehende Rabatt abgerechnet. Anschließend erfolgt die Berechnung der Einzelposition durch Multiplikation der beiden Variablen `sngMenge` und `sngBetrag`. Im dritten Schritt wird diese Einzelposition zum Gesamtbetrag addiert.

Beachten Sie, dass nur das Argument `sngTotal` als Referenz an die Prozedur übergeben wird. So ist sichergestellt, dass nur die Variable `sngGesamtKosten` in der Hauptprozedur geändert werden kann. Würden alle drei Werte als Referenz übergeben, würde die Berechnung des Rabatts im ersten Arbeitsschritt eine Änderung der Preise in der Hauptprozedur auslösen.

**Listing 3.4** Bei Verwendung von *ByRef* wird die Variable innerhalb der Hauptprozedur geändert

```

Sub Demo_ByRef()
    Dim sngGesamtKosten As Single

    subGesamtKosten 2, 15.5, sngGesamtKosten
    subGesamtKosten 21, 5, sngGesamtKosten
    subGesamtKosten 2.5, 10, sngGesamtKosten
    MsgBox sngGesamtKosten
End Sub

Public Sub subGesamtKosten( _
    ByVal sngMenge As Single, _
    ByVal sngPreis As Single, _
    ByRef sngTotal As Single) _

    Dim sngEinzelPosition As Single
    Dim sngRabatt As Single

    sngRabatt = 0.9

    sngPreis = sngPreis * sngRabatt
    sngEinzelPosition = sngMenge * sngPreis
    sngTotal = sngTotal + sngEinzelPosition

    MsgBox "EinzelPosition " & CStr(sngEinzelPosition) & _
        vbCrLf & "Total " & CStr(sngTotal)
End Sub

```

Im ersten Abschnitt dieses Kapitels haben wir Ihnen empfohlen, sämtliche Variablen zusammen mit einem Datentyp zu deklarieren. Ferner möchten wir Ihnen jetzt nahe legen, bei jeder Deklaration von Argumenten anzugeben, auf welche Art diese an die Prozedur übergeben werden.

Der Programmcode bleibt so übersichtlicher, weil die detaillierte Deklaration der Argumente eine indirekte Dokumentation des Programms darstellt. Auch können sich weniger Programmfehler einschleichen, da Sie bewusst die eine oder andere Art der Übergabe festgelegt haben.

## Variablen in Datenfeldern ablegen

Array

Zusammengehörende Variablen können in einem Datenfeld (*Array*) abgelegt und verwaltet und dadurch auch ihre Anzahl verringert werden. Abgesehen vom Wegfall des Deklarationsaufwands, es muss nur eine einzige Variable statt einer großen Anzahl Variablen deklariert werden, wird das Programm auch flexibler, da die benötigte Menge an Variablen bei Bedarf festgelegt werden kann.

Als Beispiel dient die Empfängeradresse eines Briefs, bei der die Anzahl der Adresszeilen unterschiedlich sein kann. Jetzt können, wie in Listing 3.5 ersichtlich, Variablen für Anrede, Vorname, Nachname, Straße, Postleitzahl, Ort usw. deklariert werden. Soll der Brief ins Ausland versendet werden oder handelt es sich bei der Anschrift um eine Firmenadresse, fehlen entsprechende Variablen. Keine Probleme entstehen, wenn wie in Listing 3.6 ein Datenfeld für die einzelnen Adresszeilen erstellt wird.

Listing 3.5 Zuweisen der Briefadresse an einzelne definierte Variablen

```

Sub Demo_OhneDatenfeld()
    Dim strAnrede As String
    Dim strVornamenNamen As String
    Dim strStrasse As String
    Dim strPostleitzahlOrt As String
    Dim doc As Word.Document

    'Adresse abfragen
    strAnrede = InputBox("Anrede eingeben")
    strVornamenNamen = InputBox("Vorname und Name eingeben")
    strStrasse = InputBox("Strasse eingeben")
    strPostleitzahlOrt = InputBox("Postleitzahl und Ort eingeben")

    'Brief erstellen
    Set doc = Application.Documents.Add
    doc.Range.Text = strAnrede & vbVerticalTab & _
        strVornamenNamen & vbVerticalTab & _
        strStrasse & vbVerticalTab & _
        strPostleitzahlOrt
End Sub

```

Um die Programmzeilen in Listing 3.6 richtig verstehen zu können, folgen nähere Informationen zu den Datenfeldern:

- Die Standarduntergrenze für ein Array ist Null. Es besteht aber die Möglichkeit, diesen Wert mit der Anweisung `Option Base` zu ändern und auf Eins (1) zu setzen.

**WICHTIG**

Wir raten davon ab, `Option Base` zu verwenden. Da diese Anweisung auf Modullebene deklariert werden muss, wirkt sie sich entsprechend auf alle im gleichen Modul deklarierten Datenfelder aus. Möchten Sie die Untergrenze einzelner Datenfelder bewusst auf einen bestimmten Wert setzen, kann dies bei der Deklaration der Variablen erfolgen. Vom .NET-Framework wird die Anweisung `Option Base` nicht unterstützt.

- Ein Datenfeld kann eine oder mehrere Dimensionen haben. Grundsätzlich besteht sogar die Möglichkeit, verschachtelte Datenfelder zu erzeugen, also ein einzelnes Feld, welches wiederum ein Datenfeld enthält. Die Verwaltung von Datenfeldern mit mehr als drei Dimensionen oder verschachtelter Konstrukte ist jedoch sehr komplex und kommt daher in der Praxis eher selten vor.

```

Dim strVariable1(5) As String
Dim strVariable2(5, 3) As String

```

- Ein Datenfeld kann entweder in der Deklarationszeile mit einer statischen Größe vorbelegt werden

```

Dim strAdresse(5) As String

```

oder die Zuweisung der Größe mit Hilfe von `ReDim` dynamisch während der Laufzeit des Programms erhalten.

```

Dim strAdresse() As String
ReDim Preserve strAdresse(intZähler)

```

ReDim,  
ReDim  
Preserve

**HINWEIS**

Mit der ReDim-Anweisung kann ein Datenfeld neu »dimensioniert«, also die Obergrenze neu gesetzt werden. Dies ist jedoch nur möglich, wenn keine statische Größe deklariert wurde. Die ReDim-Anweisung kann nur auf die letzte Dimension des Datenfelds angewendet werden. Wird die Anweisung ohne den Zusatz Preserve verwendet, wird das Datenfeld neu initialisiert und die gespeicherten Werte werden verworfen.

**Listing 3.6** Zuweisen der Briefadresse an ein dynamisches Datenfeld

```
Sub Demo_MitDatenfeld()
    Dim strAdressZeile As String
    Dim strAdresse() As String
    Dim intZähler As Integer
    Dim doc As Word.Document

    'Adresse abfragen
    Do
        strAdressZeile = InputBox("Adresszeile " & CStr(intZähler + 1) & " eingeben")

        If Len(strAdressZeile) <> 0 Then
            ReDim Preserve strAdresse(intZähler)
            strAdresse(intZähler) = strAdressZeile
            intZähler = intZähler + 1
        Else
            Exit Do
        End If
    Loop While Len(strAdressZeile) <> 0

    'Brief erstellen
    Set doc = Application.Documents.Add
    For intZähler = 0 To intZähler - 1
        doc.Range.InsertAfter strAdresse(intZähler) & vbVerticalTab
    Next intZähler
End Sub
```

### Größe eines Datenfelds ermitteln

LBound  
UBound

Die aktuelle Größe eines Datenfelds kann während der Laufzeit dynamisch ermittelt werden. Dazu stellt VBA die beiden Funktionen LBound und UBound zur Verfügung.

Die Schleife zum Erstellen der Briefadresse aus Listing 3.6 hätte unter Verwendung dieser Funktionen folgendermaßen ausgesehen:

```
For intZähler = LBound(strAdresse) To UBound(strAdresse)
```

### Datenfelder sortieren

Leider bietet VBA keine Funktion, um ein Datenfeld zu sortieren. Entweder muss eine eigene Funktion entwickelt oder im Internet nach entsprechenden Sortieralgorithmen (Bubblesort usw.) gesucht werden. Alternativ kann die Funktion SortArray des WordBasic-Objekts eingesetzt werden. Nähere Informationen zu WordBasic enthält Kapitel 6.



Die aufgeführten Codesequenzen finden Sie in der Beispieldatei *Bsp03\_01.doc* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap03*.

# Konstanten

Unter einer Konstante versteht man gemäß Lexikon eine »Größe, deren Wert sich nicht verändert«. Insofern stellt sie das Gegenteil einer Variablen dar – womit das Thema eigentlich schon abgehandelt wäre.

Einer Konstanten können die gleichen Datentypen zugewiesen werden, und es gelten die gleichen Regeln zur Gültigkeit und Lebensdauer wie bei Variablen. Wir Autoren empfehlen hier ebenfalls, die Deklaration sämtlicher Konstanten grundsätzlich am Anfang einer Prozedur bzw. Funktion vorzunehmen, also analog den Variablen.

Abschließend bleibt aber die Frage offen, was denn den Einsatz einer Konstanten rechtfertigt oder wofür eine Konstante im Programm überhaupt verwendet werden kann.

## Verwendungszweck von Konstanten

Konstanten werden in einer Programmsequenz dann eingesetzt, wenn ein gleicher Wert mehrmals verwendet wird (beispielsweise ein fixer Umrechnungsfaktor, der Dateiname einer Konfigurationsdatei usw.). Die Verwendung einer Konstanten bietet folgende Vorteile:

- Der effektive Wert ist nur an einer Stelle innerhalb des Projekts eingetragen. Bei einer eventuellen Änderung des Wertes muss die Anpassung lediglich an dieser Stelle vorgenommen werden.
- Das Einschleichen von Programmfehlern wird reduziert, denn anstelle einer immer gleichen Anweisung innerhalb des Projekts wird eine Konstante verwendet. Konstanten werden von IntelliSense unterstützt und vom Compiler als solche erkannt. Schreibfehler sind somit ganz ausgeschlossen.
- Mit Hilfe der Konstanten können Sie Ihren Programmcode zusätzlich dokumentieren. Die Programmsequenz bleibt übersichtlicher, die Anweisungen sind verständlicher.

Aus diesen Gründen ist es sinnvoll, wenn vom Einsatz von Konstanten regelmäßig Gebrauch gemacht wird. Der Nachteil des zusätzlichen Deklarationsaufwands wird durch die Vorteile wettgemacht.

Die Anwendung von Konstanten soll Ihnen anhand einer einfachen Umrechnung des englischen Längenmaßes Inch (Zoll) in Meter aufgezeigt werden.

```
sngLängeA = sngLängeA * 0.0254
```

Mit dieser Anweisung lässt sich auf den ersten Blick nicht feststellen, welche Bedeutung die Zahl »0.0254« innerhalb des Projekts hat und aus welchem Grund diese Berechnung durchgeführt wird.

```
sngLängeA = sngLängeA * sngINCH_METER
```

Wird die Zahl »0.0254« jedoch durch eine Konstante ersetzt, kann der Grund der Berechnung anhand des aussagekräftigen Namens der Variablen abgeleitet werden.

### Listing 3.7 Verwenden von Konstanten in Umrechnungsfunktionen

```
Option Explicit
Public Const sngINCH_METER As Single = 0.0254
```

**Listing 3.7** Verwenden von Konstanten in Umrechnungsfunktionen (Fortsetzung)

```
Sub Demo_1()
    Dim sngLängeA As Single

    sngLängeA = 10

    MsgBox CStr(sngLängeA) & " Inch = " & vbCrLf & _
        CStr(fktInchInMeter(sngLängeA)) & " Meter"
End Sub

Function fktInchInMeter(
    ByVal sngInch As Single) _
    As Single

    fktInchInMeter = sngInch * sngINCH_METER
End Function
```

Wenn Sie das Listing 3.7 genauer studieren, werden Sie feststellen, dass die Deklaration der Konstanten sngINCH\_METER für das ganze Projekt sichtbar eingetragen wurde.

```
Public Const sngINCH_METER As Single = 0.0254
```

Dies widerspricht ja den Empfehlungen des Autorenteam, Konstanten (und auch Variablen) möglichst immer auf Prozedurebene zu deklarieren. Grundsätzlich haben Sie natürlich Recht, doch im vorliegenden Fall stellt sich die Frage, ob der definierte Umrechnungsfaktor wirklich nur der betreffenden Funktion zur Verfügung stehen muss. Bereits eine einfache Änderung am Programm zeigt den Grund für eine globale Konstante auf:

```
MsgBox CStr(sngLängeA) & " Inch = " & vbCrLf & _
    CStr(fktInchInMeter(sngLängeA)) & " Meter" & vbCrLf & _
    CStr(sngINCH_METER) & " Umrechnungsfaktor"
```

---

**HINWEIS** Für die Bezeichnungen der Konstanten innerhalb der einzelnen Programmbeispiele wurden spezielle Namenskonventionen eingehalten. Zusätzliche Informationen und Empfehlungen für die Namensgebung von Variablen finden Sie in Anhang A.

---

Wird eine Konstante neu angelegt, sollten Sie sich insbesondere Gedanken zu ihrer Gültigkeit und Sichtbarkeit machen, damit sie beim Programmieren des Projekts auch wirklich zur Verfügung steht.

Die Erfahrungen aus Listing 3.7 zeigen, dass die passende Deklaration oft erst auf den zweiten Blick zu erkennen ist.




---

Die aufgeführten Codesequenzen finden Sie in der Beispieldatei *Bsp03\_02.doc* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap03*.

---



# Benutzerdefinierte Typen

Neben Variablen und Konstanten kennt VBA noch zwei weitere Arten, um Daten innerhalb des Programms zu speichern bzw. eine Programmsequenz übersichtlicher zu gestalten:

- den benutzerdefinierten Datentyp
- die Aufzählung

## Type-Anweisung

Type  
End Type

Beim benutzerdefinierten Datentyp handelt es sich eigentlich um ein Konstrukt aus einem oder mehreren Elementen. Dabei werden Variablen, welche miteinander in einem direkten Zusammenhang stehen, zusammengefasst. Jedes Element eines benutzerdefinierten Datentyps wird entweder als Standarddatentyp (String, Integer usw.) deklariert oder es wird ein benutzerdefinierter Datentyp zugewiesen:

```
Public Type Person
    strName As String
    strVorname As String
    strLand As String
    dateGeburtstag As Date
End Type
```

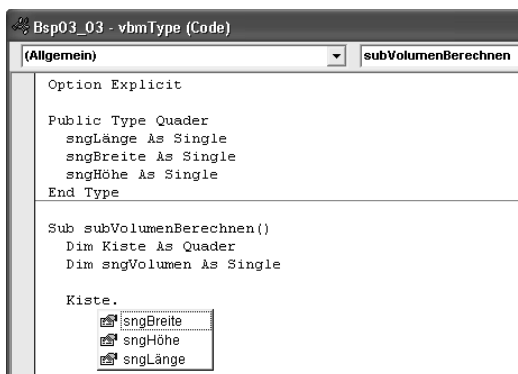
Benutzerdefinierte Typen können grundsätzlich nur auf Modulebene deklariert werden. Es besteht jedoch die Möglichkeit, die Gültigkeit für den neuen Typ auf das entsprechende Modul zu beschränken, und zwar durch Verwendung des Schlüsselworts `Private` innerhalb der Deklarationszeile.

Der benutzerdefinierte Datentyp steht innerhalb der Programmsequenz erst dann zur Verfügung, wenn der neue Typ einer Variablen zugewiesen wird:

```
Dim Mitarbeiter As Person
```

In Hinsicht auf Gültigkeit, Sichtbarkeit, Deklaration usw. dieser Variablen gelten die gleichen Regeln wie bei einer Variablen eines Standarddatentyps.

**Abbildg. 3.3** Deklarieren Sie den benutzerdefinierten Datentyp und weisen Sie diesen einer Variablen zu



Für die Verwendung von benutzerdefinierten Datentypen im eigenen Projekt sprechen die unten aufgeführten Gründe:

- Der Programmcode bleibt übersichtlicher, da die detaillierte Deklaration der Elemente sowie die Anwendung des Typs eine indirekte Dokumentation des Programms darstellen.
- Die Deklaration der Elemente erfolgt an zentraler Stelle. Werden verschiedene Variablen des gleichen Typs angelegt, ist sichergestellt, dass alle Variablen dieses Typs über die gleichen Elemente verfügen.
- Die zusammengehörenden Elemente können auf einen Blick als solche erkannt werden. Dies wäre bei einer Verwendung von einzelnen Variablen nicht sichergestellt.
- Bei der Verwendung einer benutzerdefinierten Variablen wird der Entwickler von der Entwicklungsumgebung durch IntelliSense unterstützt.

Es ist also durchaus sinnvoll, regelmäßig vom Einsatz benutzerdefinierter Datentypen Gebrauch zu machen. Der Nachteil des zusätzlichen Deklarationsaufwands wird durch die Vorteile wettgemacht.

#### HINWEIS

In der .NET-Umgebung wird die Deklaration »Type« durch den Begriff Structure ersetzt.

#### Ein Beispiel

Das nachfolgende Beispiel soll Ihnen die Vorteile des benutzerdefinierten Datentyps näher bringen. Es sollen drei Werte (Name, Vorname und Abteilung) von einer frei definierten Menge an Mitarbeitern erfasst werden.

In Listing 3.8 werden die erfassten Daten in einem Array abgelegt. Die Verwendung des Arrays ist für unsere Bedürfnisse sicher geeignet, doch sind damit auch drei einschneidende Nachteile verbunden:

- Das Array kann nur Daten des deklarierten Datentyps aufnehmen – in unserem Fall also nur Daten vom Typ String.
- Auf den ersten Blick ist nicht ersichtlich, in welchem Bereich des Arrays die einzelnen Werte (Name, Vorname und Abteilung) für die Mitarbeiter abgelegt werden.
- Damit eine dynamische Anzahl von Mitarbeitern erfasst werden kann, muss ReDim Preserve verwendet werden. Eine Vergrößerung des Arrays kann nur in der letzten Dimension des Arrays erfolgen. Dies hat zur Folge, dass in einem Element nicht die Daten eines einzelnen Mitarbeiters, sondern die Werte einer Kategorie abgelegt sind.

Abbildg. 3.4

In jedem Element des Arrays sind die Daten einer Kategorie abgelegt

Ausdruck	Wert	Typ
sngMitarbeiter		String(0 to 2, 0 to 2)
sngMitarbeiter(0)		String(0 to 2)
sngMitarbeiter(0,0)	"Gahler"	String
sngMitarbeiter(0,1)	"Müller"	String
sngMitarbeiter(0,2)	"Kellenberger"	String
sngMitarbeiter(1)		String(0 to 2)
sngMitarbeiter(1,0)	"Thomas"	String
sngMitarbeiter(1,1)	"Karin"	String
sngMitarbeiter(1,2)	"Sonja"	String
sngMitarbeiter(2)		String(0 to 2)

Listing 3.8 Verwalten einer dynamischen Anzahl von Mitarbeitern, ein schlechtes Codebeispiel

```

Sub Type_Schlecht_A()
    Dim sngMitarbeiter() As String
    Dim intZähler As Integer

    Do While vbYes = MsgBox("Mitarbeiter erfassen?", vbYesNo)
        ReDim Preserve sngMitarbeiter(2, intZähler)

        sngMitarbeiter(0, intZähler) = InputBox("Name eingeben")
        sngMitarbeiter(1, intZähler) = InputBox("Vorname eingeben")
        sngMitarbeiter(2, intZähler) = InputBox("Abteilung eingeben")

        intZähler = intZähler + 1
    Loop
End Sub

```

In Listing 3.9 wurde versucht, die vorgängig genannten Probleme zu beseitigen. Dieser Erfolg ist jedoch sehr bescheiden ausgefallen. Damit auf den ersten Blick erkennbar ist, in welchem Bereich die Werte der Mitarbeiter hinterlegt sind, wurden der Programmsequenz drei Konstanten hinzugefügt. So konnten aussagekräftige Bezeichner für die einzelnen Felder eingeführt werden.

Listing 3.9 Verwalten einer dynamischen Anzahl von Mitarbeitern, ein mäßiges Codebeispiel

```

Sub Type_Mässig()
    Const intMA_NAME As Integer = 0
    Const intMA_VORNAME As Integer = 1
    Const intMA_ABTEILUNG As Integer = 2

    Dim sngMitarbeiter() As String
    Dim intZähler As Integer

    Do While vbYes = MsgBox("Mitarbeiter erfassen?", vbYesNo)
        ReDim Preserve sngMitarbeiter(intMA_NAME To intMA_ABTEILUNG, intZähler)

        sngMitarbeiter(intMA_NAME, intZähler) = InputBox("Name eingeben")
        sngMitarbeiter(intMA_VORNAME, intZähler) = InputBox("Vorname eingeben")
        sngMitarbeiter(intMA_ABTEILUNG, intZähler) = InputBox("Abteilung eingeben")

        intZähler = intZähler + 1
    Loop
End Sub

```

In Listing 3.10 wurde auf das mehrdimensionale Array verzichtet. Stattdessen wurde ein benutzerdefinierter Typ angelegt. Die Daten werden in einem eindimensionalen Array abgelegt, als Datentyp wurde der neu angelegte benutzerdefinierte Datentyp zugewiesen.

Auf diese Weise konnten die Probleme, die uns das mehrdimensionale Array beschert hat, umgangen werden:

- Durch die Verwendung des benutzerdefinierten Datentyps können unterschiedliche Datentypen im Array abgelegt werden.
- Auf den ersten Blick und ohne Dokumentation ist erkennbar, welche Werte des Mitarbeiters in welcher Variablen abgelegt werden.

- Alle Daten des Mitarbeiters sind im gleichen Element des Arrays abgespeichert.

**Listing 3.10** Verwalten einer dynamischen Anzahl von Mitarbeitern, ein gelungenes Codebeispiel

```
Option Explicit

Type Angestellter
    strName As String
    strVorname As String
    strAbteilung As String
End Type

Sub Type_Gelungen()
    Dim Mitarbeiter() As Angestellter
    Dim intZähler As Integer

    Do While vbYes = MsgBox("Mitarbeiter erfassen?", vbYesNo)
        ReDim Preserve Mitarbeiter(intZähler)

        Mitarbeiter(intZähler).strName = InputBox("Name eingeben")
        Mitarbeiter(intZähler).strVorname = InputBox("Vorname eingeben")
        Mitarbeiter(intZähler).strAbteilung = InputBox("Abteilung eingeben")

        intZähler = intZähler + 1
    Loop
End Sub
```

## Enum-Anweisung

Enum,  
End  
Enum

Bei den Aufzählungsvariablen handelt es sich um konstante Werte für eine Variable. Die gültigen Werte für die einzelne Variable entsprechen einer klar definierten Menge. Jedem einzelnen Wert ist eine Bedeutung zugewiesen. Jedes Element der Aufzählung ist vom Datentyp Long.

```
Public Enum EWocheTag
    eMontag = 1
    eDienstag = 2
    eMittwoch = 3
    eDonnerstag = 4
    eFreitag = 5
    eSamstag = 6
    eSonntag = 7
End Enum
```

### HINWEIS

Ein manuelles Zuweisen von Zahlenwerten an die einzelnen Elemente der Aufzählung ist nicht zwingend notwendig. Werden keine spezifischen Werte zugewiesen, werden die Elemente automatisch in der erfassten Reihenfolge durchnummeriert.

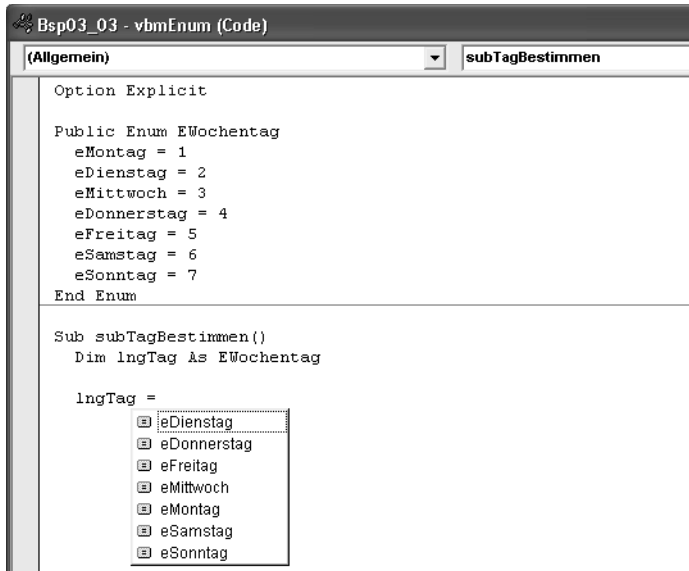
Aufzählungsvariablen können grundsätzlich nur auf Modulebene deklariert werden. Es besteht jedoch die Möglichkeit, die Gültigkeit der Aufzählung auf das entsprechende Modul zu beschränken. Dies erfolgt durch die Verwendung des Schlüsselworts `Private` innerhalb der Deklarationszeile.

Die Aufzählungsvariable steht innerhalb der Programmsequenz erst dann zur Verfügung, wenn diese einer Variablen zugewiesen wird:

```
Dim lngWochentag As Ewochentag
```

Hinsichtlich der Gültigkeit, Sichtbarkeit, Deklaration usw. dieser Variablen gelten die gleichen Regeln wie bei einer Variablen eines Standarddatentyps.

Abbildg. 3.5 Deklarieren Sie die Aufzählungsvariable und weisen Sie diese einer Variablen zu



Für die Verwendung von Aufzählungsvariablen im eigenen Projekt sprechen verschiedene Vorteile, die hier kurz aufgeführt werden:

- Der Programmcode bleibt übersichtlicher, da die detaillierte Deklaration der Elemente und die Anwendung der Aufzählung eine indirekte Dokumentation des Programms darstellt.
- Die Deklaration der Elemente erfolgt an zentraler Stelle. Werden verschiedene Variablen mit der gleichen Aufzählung angelegt, ist sichergestellt, dass alle Variablen über die gleichen Elemente verfügen.
- Bei der Verwendung einer Aufzählungsvariablen wird der Entwickler von der Entwicklungsumgebung durch IntelliSense unterstützt.

Aus den genannten Gründen sollten Sie vom Einsatz der Aufzählungen regelmäßigen Gebrauch machen. Der Nachteil des zusätzlichen Deklarationsaufwands wird durch die Vorteile aufgewogen.

**WICHTIG** Die Verwendung von Aufzählungsvariablen schützt Sie nicht vor Fehlern bei der Zuweisung von »ungültigen« Werten an eine als Aufzählung deklarierte Variable:

```
Dim lngTag As Ewochentag
lngTag = 232
```

Es wird weder vom Compiler noch während der Laufzeit des Programms ein Fehler entdeckt, wenn der Variablen ein Wert außerhalb der Aufzählung zugewiesen wird. Der Grund für dieses Verhalten liegt im eigentlichen Typ der Aufzählungsvariablen, dieser ist fix als Datentyp Long deklariert.

### Ein Beispiel

Das nachfolgende Beispiel wird Ihnen die Vorteile der Aufzählungsvariablen etwas näher bringen. Anhand einer numerischen Eingabe soll darin die Himmelsrichtung bestimmt werden.

In Listing 3.11 wird die Eingabe innerhalb einer Select Case-Anweisung ausgewertet. Zur Unterscheidung der Werte werden die Zahlenwerte verwendet, doch sind damit auch Nachteile verbunden.

Auf den ersten Blick ist nicht ersichtlich, welcher gültige Wert welcher Himmelsrichtung zugewiesen wurde (sofern man die Einfachheit des Beispiels bewusst ignoriert).

**Listing 3.11** Auswerten der eingegebenen Himmelsrichtung, ein schlechtes Codebeispiel

```
Sub Enum_Schlecht()
    Dim lngHimmelsrichtung As Long
    Dim strHimmelsrichtung As String

    lngHimmelsrichtung = InputBox("Wert von 1 - 4 eingeben.")

    Select Case lngHimmelsrichtung
        Case 1
            strHimmelsrichtung = "Norden"
        Case 2
            strHimmelsrichtung = "Osten"
        Case 3
            strHimmelsrichtung = "Süden"
        Case 4
            strHimmelsrichtung = "Westen"
        Case Else
            strHimmelsrichtung = "Falsche Zahl angegeben."
    End Select

    MsgBox CStr(lngHimmelsrichtung) & vbTab & strHimmelsrichtung
End Sub
```

In Listing 3.12 wurde auf die Verwendung der Zahlenwerte verzichtet. Stattdessen wurde eine Aufzählungsvariable angelegt.

Auf diese Weise ist auf den ersten Blick erkennbar, welchem Eintrag innerhalb der Select Case-Anweisung welche Himmelsrichtung zugewiesen wurde.

**Listing 3.12** Auswerten der eingegebenen Himmelsrichtung, ein gelungenes Codebeispiel

```
Option Explicit

Enum EHimmelrichtung
    eNorden = 1
    eOsten = 2
    eSüden = 3
    eWesten = 4
```

**Listing 3.12** Auswerten der eingegebenen Himmelsrichtung, ein gelungenes Codebeispiel (Fortsetzung)

```

End Enum

Sub Enum_Gelungen()
    Dim lngHimmelsrichtung As EHimmelrichtung
    Dim strHimmelsrichtung As String

    lngHimmelsrichtung = InputBox("Wert von 1 - 4 eingeben.")

    Select Case lngHimmelsrichtung
        Case eNorden
            strHimmelsrichtung = "Norden"
        Case eOsten
            strHimmelsrichtung = "Osten"
        Case eSüden
            strHimmelsrichtung = "Süden"
        Case eWesten
            strHimmelsrichtung = "Westen"
        Case Else
            strHimmelsrichtung = "Falsche Zahl angegeben."
    End Select

    MsgBox CStr(lngHimmelsrichtung) & vbCrLf & strHimmelsrichtung
End Sub

```

Da eine Aufzählungsvariable nicht zwingend der Reihe nach durchnummeriert sein muss, kann das Einsatzgebiet erweitert werden. So können die Variablen zur Benennung von Werten (beispielsweise Wochentage, Farbwerte usw.) verwendet werden. Die hinterlegten Einheiten können aber auch als Umrechnungsfaktoren Verwendung finden, wie in Listing 3.13 dargestellt.

**Listing 3.13** Verwenden Sie Aufzählungsvariablen als Umrechnungsfaktoren

```

Option Explicit

Public Const sngMILE_METER As Single = 1609

Public Enum EMeterFaktor
    eMeterKM = -3
    eMeterM = 1
    eMeterCM = 2
    eMeterMM = 3
End Enum

Sub Enum_Umrechnen()
    Dim sngMeilen As Single

    sngMeilen = InputBox("Meilen eingeben")

    MsgBox "km " & fktMeilenMeter(sngMeilen, eMeterKM)
    MsgBox "cm " & fktMeilenMeter(sngMeilen, eMeterCM)
End Sub

Private Function fktMeilenMeter _
    (ByVal sngStrecke As Single, _
    ByVal lngFaktor As EMeterFaktor) _

```

**Listing 3.13** Verwenden Sie Aufzählungsvariablen als Umrechnungsfaktoren (Fortsetzung)

```
As Single

fktMeilenMeter = sngStrecke * sngMILE_METER * 10 ^ lngFaktor
End Function
```



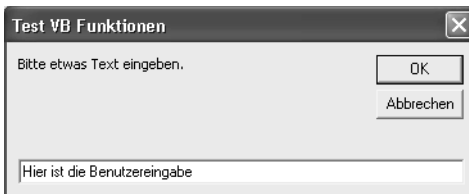
Die aufgeführten Codesequenzen finden Sie in der Beispieldatei *Bsp03\_03.doc* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap03*.

## Nützliche VBA-Funktionen

Mit dem Namen »Visual Basic für Applikationen« wäre es nur logisch, in der Office-Programmiersprache auch etwas vom »echten« Visual Basic zu finden, und so ist es. Neben der Funktionalität des Anwendungsobjektmodells stehen dem Entwickler eine Vielzahl allgemein nützlicher Funktionen und Methoden zur Verfügung. Diese sind alle in den Hilfedateien zu Visual Basic aufgelistet. Hier werden wir einige der am häufigsten verwendeten beschreiben.

**InputBox** Windows-Anwendungen sind »interaktiv«: Der Benutzer wird in den Ablauf mit einbezogen. Das bedeutet, er wird gelegentlich zu einer Eingabe aufgefordert. Für viele Aufgaben lohnt es sich, eine UserForm aufzubauen, wie in Kapitel 14 beschrieben. Handelt es sich jedoch nur um sehr kurze oder einfache Eingaben, ist es weniger aufwändig, eine InputBox einzublenden, wie in Abbildung 3.6 abgebildet.

**Abbildg. 3.6** Eine InputBox für kurze, einfache Benutzereingaben



Die Benutzereingabe wird als Zeichenkette zurückgegeben, die entweder direkt weiterverwendet oder in einer Variablen gespeichert wird. Die Funktion hat die folgende Syntax, wobei »prompt« Aufforderung bedeutet. Eine Beschreibung aller Argumente finden Sie im Hilfeintrag zur Funktion.

```
InputBox(prompt[, title] [, default] [, xpos] [, ypos] [, helpfile, context])
```

### TIPP

Wenn Sie ein Argument in eckigen Klammern ([]) sehen, ist das Argument optional. Optionale Argumente werden immer am Schluss, hinter den erforderlichen, stehen.

In Listing 3.14 ist ein einfaches Beispiel dargestellt. In der Prozedur `InputBoxAnzeigen` werden zwei Variablen als Zeichenketten (String) deklariert. Der Aufforderungstext wird `strAufforderung` zuge-



wiesen, den Wert für strAntwort gibt die InputBox in Abbildung 3.6 zurück. Der Text, den der Benutzer in das Eingabefeld einträgt, wird in einer MsgBox angezeigt, wie in Abbildung 3.7 dargestellt.

**Listing 3.14** Zusammenwirken der Funktionen *InputBox* und *MsgBox*

```
Private Const strMSGITITEL As String = "Test VB Funktionen"

Sub InputBoxAnzeigen()
    Dim strAufforderung As String
    Dim strAntwort As String

    strAufforderung = "Bitte etwas Text eingeben."
    strAntwort = InputBox(strAufforderung, strMSGITITEL)
    MsgBox strAntwort
End Sub
```

**Abbildg. 3.7** Eine *MsgBox* übermittelt dem Benutzer eine kurze Nachricht.



**MsgBox** Die Funktion *MsgBox* hat folgende Syntax:

```
MsgBox(prompt[, buttons] [, title] [, helpfile, context])
```

Auch hier ist prompt das einzige erforderliche Argument. Das Argument buttons ist aber sehr interessant und hilfreich. Damit lassen sich nicht nur verschiedene Schaltflächen anzeigen, um das Feedback des Benutzers auszuwerten. Sie können ihm auch den Wichtigkeitsgrad der Meldung mitteilen. In Tabelle 3.2 sind die möglichen Werte aufgelistet. Diese können miteinander addiert werden, um eine beliebige Kombination von Schaltflächen und Symbolen anzuzeigen, wie das Listing 3.15 und die Abbildung 3.8 demonstrieren.

**Tabelle 3.2** Werte für das Argument *buttons* der Funktion *MsgBox*

Konstante	Wert	Beschreibung
vbOKOnly	0	Nur die Schaltfläche <i>OK</i> anzeigen.
vbOKCancel	1	Schaltflächen <i>OK</i> und <i>Abbrechen</i> anzeigen.
vbAbortRetryIgnore	2	Schaltflächen <i>Abbruch</i> , <i>Wiederholen</i> und <i>Ignorieren</i> anzeigen.
vbYesNoCancel	3	Schaltflächen <i>Ja</i> , <i>Nein</i> und <i>Abbrechen</i> anzeigen.
vbYesNo	4	Schaltflächen <i>Ja</i> und <i>Nein</i> anzeigen.
vbRetryCancel	5	Schaltflächen <i>Wiederholen</i> und <i>Abbrechen</i> anzeigen.
vbCritical	16	 Meldung mit Stopp-Symbol anzeigen.

**Tabelle 3.2** Werte für das Argument *buttons* der Funktion *MsgBox* (Fortsetzung)

Konstante	Wert	Beschreibung
vbQuestion	32	 Meldung mit Fragezeichen-Symbol anzeigen.
vbExclamation	48	 Meldung mit Ausrufezeichen-Symbol anzeigen.
vbInformation	64	 Meldung mit Info-Symbol anzeigen.
vbDefaultButton1	0	Erste Schaltfläche ist Standardschaltfläche.
vbDefaultButton2	256	Zweite Schaltfläche ist Standardschaltfläche.
vbDefaultButton3	512	Dritte Schaltfläche ist Standardschaltfläche.
vbDefaultButton4	768	Vierte Schaltfläche ist Standardschaltfläche.
vbApplicationModal	0	An die Anwendung gebunden. Der Benutzer muss auf das Meldungsfeld reagieren, bevor er seine Arbeit mit der aktuellen Anwendung fortsetzen kann.
vbSystemModal	4096	An das System gebunden. Alle Anwendungen werden unterbrochen, bis der Benutzer auf das Meldungsfeld reagiert.
vbMsgBoxHelpButton	16384	Fügt die Schaltfläche <i>Hilfe</i> der MsgBox hinzu.
vbMsgBoxSetForeground	65536	Setzt das MsgBox-Fenster in den Vordergrund.
vbMsgBoxRight	524288	Der Text wird rechts ausgerichtet.
vbMsgBoxRtlReading	1048576	Legt fest, dass Text auf hebräischen und arabischen Systemen von rechts nach links auszurichten ist.

**Abbildg. 3.8** Eine *MsgBox* aussagekräftig mit verschiedenen Kombinationen von Schaltflächen und Symbolen gestalten

**Listing 3.15** Die *MsgBox* wird mit Schaltflächen, Symbolen und einem Titel ausgestattet

```

Sub TestVBFunktionen2()
    Dim strAufforderung As String
    Dim strAntwort As String

    strAufforderung = "Bitte etwas Text eingeben."
    strAntwort = InputBox (strAufforderung, strMSGTITEL)
    MsgBox strAntwort, vbYesNo + vbQuestion, strMSGTITEL
End Sub

```

Ihnen ist vielleicht aufgefallen, dass das *Schließen*-Symbol oben rechts in Abbildung 3.8, nicht aktiv ist. Dies geschieht, weil vom Benutzer eine klare »Ja«- oder »Nein«-Antwort erwartet wird. Nun gut,

der Benutzer kann auf *Ja* oder *Nein* klicken, aber woher weiß der Code, was gewählt wurde? Die Auswahl wird in einer Variablen festgehalten.

```
lWert = MsgBox(strAntwort, vbYesNo + vbQuestion, strMSGTITEL)
```

**TIPP**

Bitte beachten Sie: Wenn der Rückgabewert in einer Funktion festgehalten wird, müssen alle Argumente in einem Klammernpaar eingeschlossen sein. Vergleichen Sie die Codezeilen, die die MsgBox-Funktion aufrufen, mit und ohne voranstehende Variable für den Rückgabewert.

Die Werte, die eine MsgBox zurückgibt, finden Sie in Tabelle 3.3 aufgelistet. Wie Sie diese Information auswerten, ist im Abschnitt »Bedingungen« in diesem Kapitel beschrieben.

**Tabelle 3.3** Die Rückgabewerte für eine *MsgBox*

Konstante	Wert	Beschreibung
vbOK	1	OK
vbCancel	2	Abbrechen
vbAbort	3	Abbruch
vbRetry	4	Wiederholen
vbIgnore	5	Ignorieren
vbYes	6	Ja
vbNo	7	Nein

Left,  
Right,  
Mid

Bislang wurde immer mit der ganzen Zeichenkette gearbeitet. Es kommt aber vor, dass nur ein Bruchteil davon benötigt wird. Visual Basic stellt einige Funktionen zur Verfügung, mit denen wir Buchstaben von links (Left), von rechts (Right) oder von einer beliebigen Stelle (Mid) auslesen können. Die Syntax der drei Funktionen lautet:

```
Left(string, length)
Right(string, length)
Mid(string, start[, length])
```

Wie diese Funktionen verwendet werden, veranschaulicht Listing 3.16. Das Resultat sehen Sie in Abbildung 3.9.

**Listing 3.16** Die Wirkungsweise der Zeichenketten-Funktionen *Left*, *Right* und *Mid*

```
Sub TestVBZeichenkettenFunktionen1()
    Dim strAufforderung As String
    Dim strAntwort As String
    Dim lWert As Long

    strAufforderung = "Bitte etwas Text eingeben."
    strAntwort = InputBox (strAufforderung)
    MsgBox "Die vollständige Zeichenkette: " & strAntwort & vbCr & _
```

**Listing 3.16** Die Wirkungsweise der Zeichenketten-Funktionen *Left*, *Right* und *Mid*

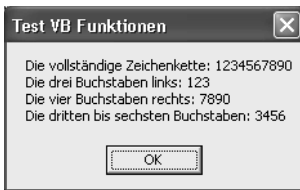
```

"Die drei Buchstaben links: " & Left(strAntwort, 3)
& vbCr & "Die vier Buchstaben rechts: " & Right(strAntwort, 4)
& vbCr & "Die dritten bis sechsten Buchstaben: " & Mid(strAntwort, 3, 4) _
, , strMSGITEL
End Sub

```

**TIPP**

In Listing 3.16 sehen Sie auch, wie lange Codezeilen mit einem Unterstrich umbrochen werden. Ebenfalls ersichtlich ist, wie mit vbCR neue Zeilen in eine Zeichenkette eingebaut werden. Statt vbCR können Sie auch Chr\$(13) verwenden.

**Abbildg. 3.9** Das Resultat von Listing 3.16


Instr,  
InstrRev,  
Replace

Es ist manchmal hilfreich zu wissen, ob ein Zeichen in einer Zeichenkette überhaupt vorkommt, und wenn ja, an welcher Stelle. Dafür stellt Visual Basic die Funktionen Instr und InstrRev bereit, die die Position des gesuchten Zeichens zurückgeben, wie in Abbildung 3.10 und Listing 3.17 zu sehen. Die Syntax der zwei Funktionen lautet:

```

Instr([Start, ]Zeichenfolge1, Zeichenfolge2[, Vergleich])
InstrRev(stringcheck, stringmatch [, start[, compare]])

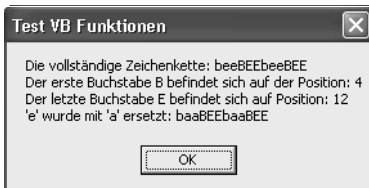
```

Zudem ist es möglich, mit der Funktion Replace alle Vorkommnisse einer Zeichenkette innerhalb einer anderen zu ersetzen. Die Syntax der Replace-Funktion:

```

Replace(expression, find, replace[, start[, count[, compare]])

```

**Abbildg. 3.10** Die Wirkung der Funktionen *Instr*, *InstrRev*, und *Replace*

**Listing 3.17** Die Funktionen *Instr*, *InstrRev* und *Replace*

```

Sub TestVBZeichenkettenFunktionen2()

Dim strAufforderung As String
Dim strAntwort As String

```

Listing 3.17 Die Funktionen *Instr*, *InstrRev* und *Replace* (Fortsetzung)

```

Dim lWert As Long

strAufforderung = "Bitte etwas Text eingeben."
strAntwort = InputBox(strAufforderung)
MsgBox "Die vollständige Zeichenkette: " & strAntwort & vbCrLf & _
    "Der erste Buchstabe B befindet sich auf der Position: " & Instr(strAntwort, "B") & _
    vbCrLf & "Der letzte Buchstabe E befindet sich auf Position: " & _
    InstrRev(strAntwort, "E") & vbCrLf & "'e' wurde mit 'a' ersetzt: " & _
    Replace(strAntwort, "e", "a"), , strMSGTITEL

End Sub

```

Date,  
Format

In diesem Abschnitt sollen noch zwei weitere Funktionen vorgestellt werden: *Date* und *Format*. Die Funktion *Date* gibt das aktuelle Datum in dem unter Windows definierten Kurzformat zurück.

Natürlich wäre es schön, wenn das Datum auch anders angezeigt werden könnte. Dafür stellt Visual Basic die Funktion *Format* bereit. Das Ergebnis und den dahinter stehenden Code sehen Sie in Abbildung 3.11 bzw. in Listing 3.18. Diese Funktion wird auch für die Formatierung von Zahlen verwendet. Die Syntax lautet:

```
Format(Ausdruck[, Format[, firstdayofweek[, firstweekofyear]])
```

Abbildg. 3.11 Das aktuelle Datum wurde mit der Funktion *Format* nach dem Muster *t-mmm-jjji* formatiert.

Das Hilfethema zur Funktion enthält nähere Angaben zu den gültigen Symbolen, mit denen das *Format* bestimmt werden kann. Folgen Sie auch den Links unter *Siehe auch*.

Listing 3.18 Ein Datum mit der Funktion *Format* festlegen

```

Sub TestFormatFunktion()
    Dim strDatum As String

    MsgBox "Das heutige Datum: " & Date & vbCrLf & _
        "Und formatiert: " & Format(Date, "d-MMM-yyyy"), , strMSGTITEL

End Sub

```

## Bedingungen

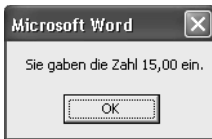
Nur selten läuft eine Aufgabe reibungslos von A bis Z durch – ohne Wenn und Aber. Meistens müssen Entscheidungen getroffen werden, und der Code muss entsprechend dieses oder jenes ausführen, je nachdem, welcher Zustand im Moment herrscht. VBA stellt grundsätzlich zwei Konstrukte

zur Verfügung, die eine Abzweigung des Codes in unterschiedlichen Bahnen ermöglichen: If...Then...Else (Wenn...dann...sonst) und Select Case (den Fall auswählen).

If...Then...Else In seiner Grundform testet If...Then eine Bedingung, und falls sie wahr ergibt, werden die festgelegten Handlungen ausgeführt. Wird Else mit einbezogen, werden die darunter gelisteten Befehle ausgeführt, wenn die Bedingung falsch ist. Ein If-Block endet immer mit End If. Die Codezeilen innerhalb des Blocks werden allgemein eingerückt, um den Code lesbarer zu gestalten.

IsNumeric Das Listing 3.19 zeigt ein Beispiel. Eine InputBox fordert den Benutzer auf, eine Zahl einzugeben. Die Eingabe wird in die Zeichenkette strAntwort zurückgegeben. In der If-Codezeile wird getestet, ob die Eingabe numerisch ist (IsNumeric). Falls ja, wird die Nachricht wie in Abbildung 3.12 angezeigt, sonst passiert nichts. Beachten Sie, dass trotz des festgelegten Zahlenformats »0.00« (also mit Punkt) als Dezimaltrennzeichen ein Komma verwendet wird. Die Format-Funktion wandelt das angegebene Format um, so dass das Resultat mit den Systemeinstellungen übereinstimmt.

Abbildg. 3.12 Die eingegebene Zahl wird mit der Funktion Format formatiert



Listing 3.19 Die Bedingung, ob die Benutzereingabe numerisch ist, wird geprüft

```
Sub TestIsNumericFunktion()
    Dim strAntwort As String

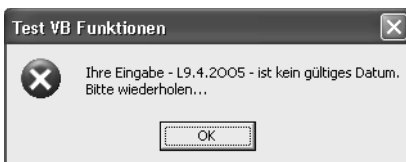
    strAntwort = InputBox("Eine Zahl eingeben:")
    If IsNumeric(strAntwort) Then
        MsgBox "Sie gaben die Zahl " &
            Format(strAntwort, "0.00") & " ein."
    End If
End Sub
```

IsDate, UCase Hat der Benutzer irrtümlich eine ungültige Angabe gemacht, wird er seinen Bildschirm etwas konsterniert betrachten, wenn die erwartete Meldung nicht erscheint. Deshalb wurde das Beispiel in Listing 3.20 um einen Else-Block erweitert. Ergibt die Auswertung der If-Codezeile den Wert »Falsch«, springt die Ausführung hierher und zeigt die Fehlermeldung aus Abbildung 3.13 an.

#### HINWEIS

Einen kniffligen Fehler stellt die Eingabe von Buchstaben statt Zahlen dar. Anfangs, als die Menschen von der Schreibmaschine auf den Rechner umstiegen, kam er häufiger vor. Aber auch heute ist er nicht auszuschließen, wenn Schwierigkeiten mit Zahlen auftreten. Die Funktion UCase zeigt klar, dass ein »l« (kleines »L«) statt einer »1« (Eins) eingegeben wurde. Weniger offensichtlich ist, dass ein großes »O« statt einer »0« (Null) eingetippt wurde.

Abbildg. 3.13 Buchstaben wurden versehentlich statt Zahlen für die Datumsangabe eingegeben



Zwei neue Visual Basic-Funktionen wurden in Listing 3.20 eingesetzt: IsDate und UCase. IsDate prüft, ob der Ausdruck ein gültiges Datum ist. Dabei spielt es keine Rolle, in welchem Format es eingegeben wurde. Das Kurz- oder Langformat von Windows ist genauso gültig wie 1-Sep-2005 oder 2006/01/01. Wichtig zu wissen ist, dass die Funktion ein gültiges Datum aktiv forciert. 8.24.2005 wird ebenso akzeptiert wie 24.8.2005; IsDate versucht aus allen möglichen Kombinationen des angegebenen Werts ein gültiges Datum zu machen.

Die Funktion UCase wandelt alle Buchstaben einer Zeichenkette in Großbuchstaben um. Wie Ihnen in Abbildung 3.10 vielleicht aufgefallen ist, unterscheiden die VB-Funktionen zwischen Groß- und Kleinschreibung.

**Listing 3.20** Entspricht die Eingabe einem Datum, wird sie in einem bestimmten Format angezeigt, ansonsten erscheint eine Meldung, dass es sich um kein gültiges Datum handelt

```
Sub TestIsDateFunktion()
    Dim strAntwort As String

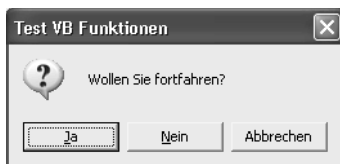
    strAntwort = InputBox("Ein Datum eingeben:")
    If IsDate(strAntwort) Then
        MsgBox "Sie haben das Datum " & _
            Format(strAntwort, "d. mmmm yyyy") & " eingegeben."
    Else
        MsgBox "Ihre Eingabe - " & strAntwort & " - ist kein gültiges Datum." -
            & vbCrLf & "Bitte wiederholen...", vbOKOnly + vbCritical, strMSGTITEL
    End If
End Sub
```

**Select Case** Nicht immer müssen nur zwei Zustände verglichen und ausgewertet werden; es können auch mehrere sein. Nehmen wir als Beispiel die MsgBox in Abbildung 3.14. Der Benutzer hat drei Schaltflächen zur Auswahl. Es wäre durchaus möglich, den Rückgabewert mit If wie folgt auszuwerten:

```
If lAuswahl = vbYes Then
    MsgBox "Sie haben »Ja« gesagt."
ElseIf lAuswahl = vbNo Then
    MsgBox "Sie haben »Nein« gesagt."
ElseIf lAuswahl = vbCancel Then
    MsgBox "Sie haben abgebrochen."
End If
```

Für mehrere Auswertungen ist diese Methode nach Meinung der Autoren weniger übersichtlich als Select Case in Listing 3.21. Zudem arbeitet sie meist langsamer.

**Abbildg. 3.14** Eine MsgBox, die drei verschiedene Werte zurückgeben kann



Select Case fängt mit Select Case [Wert] an. Dieser Zeile folgen so viele Case-Auswertungen wie nötig. Wenn [Wert] der Angabe neben Case entspricht, werden die nachfolgenden Zeilen bis zum

nächsten Case ausgeführt. Danach springt die Ausführung zu End Select. Befinden sich keine Codezeilen zwischen zwei Case-Zeilen, springt die Ausführung einfach weiter zu End Select.

**TIPP**

Es ist ratsam, als letztes Element einer Select Case-Auswertung den Fall Case Else – [Wert] entspricht keinem der Case-Werte – einzubauen, sonst erscheint eine Fehlermeldung, und die Prozedur wird abgebrochen. Mit Case Else können Sie diesen Zustand abfangen und entsprechend handeln.

**Listing 3.21** Die Rückgabewerte der *MsgBox* mit *Select Case* auswerten

```
Sub TestSelectCase()
    Dim lAuswahl As Long

    lAuswahl = MsgBox("Wollen Sie fortfahren?", vbYesNoCancel + vbQuestion, strMSGTITEL)
    Select Case lAuswahl
        Case vbYes
            MsgBox "Sie wollen fortfahren.", , strMSGTITEL
        Case vbNo
            MsgBox "Sie wollen nicht fortfahren.", , strMSGTITEL
        Case vbCancel
            MsgBox "Sie haben abgebrochen.", , strMSGTITEL
        Case Else
            MsgBox "Ein unerwarteter Wert kam zurück: " & CStr(lAuswahl), , strMSGTITEL
    End Select
End Sub
```

## Schleifen

Schleifen werden gebraucht, um Handlungen mehrmals auszuführen. Eine Schleife wird entweder ausgeführt, bis eine festgelegte Bedingung erfüllt ist, oder eine bestimmte Anzahl Male.

Do...While  
Do...Until  
Do...Loop

Im ersten Fall wird eine Do-Schleife eingesetzt. Visual Basic unterstützt vier Variationen: Do While [Bedingung]...Loop, Do Until [Bedingung]...Loop, Do...Loop While [Bedingung] und Do...Loop Until [Bedingung]. Vier Möglichkeiten können etwas verwirrend sein, aber:

- Steht die Bedingung am Anfang, wird die Schleife ein erstes Mal nur ausgeführt, falls die Bedingung zutrifft.
- Steht die Bedingung am Ende, wird die Schleife immer mindestens einmal ausgeführt, egal, ob die Bedingung zutrifft.
- While bedeutet, der geprüfte Wert wird sich nur einmal ändern, und sobald dies passiert, wird die Schleife beendet.
- Until bedeutet, der geprüfte Wert könnte sich mehrmals ändern, abgebrochen wird erst, wenn er einen bestimmten Wert erreicht hat.

**TIPP**

Es besteht bei Schleifen immer die Gefahr, in eine Endlosschleife zu geraten. Falls dies beim Testen vorkommt, können Sie mit `[Strg]+[Untbr]` die Ausführung unterbrechen. Um sicherzustellen, dass der Benutzer das Problem nie erlebt, können Sie einen Zähler in die Schleife einbauen. Erreicht er einen Grenzwert, wird der Testwert auf den Ausstiegswert gesetzt. Natürlich kann in diesem Fall eine entsprechende Meldung eingeblendet und der Code abgezweigt werden.



Das Listing 3.22 veranschaulicht die vier Variationen. Da eine Variable (lAntwort) des Datentyps Long standardmäßig den Wert 0 (Null) hat, und die Rückgabewerte einer MsgBox-Funktion von 1 bis 7 (inklusive) sind (vbNo hat den Wert 7), wird in der Prozedur TestDoWhileLoop die Frage nie eingeblendet. In der Prozedur TestDoUntilLoop wird eine Schleife so lange durchlaufen, bis der Wert von lAntwort gleich vbYes (6) ist. Hier erscheint die Nachricht, bis der Benutzer auf »Ja« klickt.

Das Verhalten der Prozedur TestDoLoopWhile, im Gegensatz zu TestDoWhileLoop, wird immer mindestens einmal ausgeführt, weil der Vergleich erst am Schluss der Schleife stattfindet. In diesem Fall wiederholt sich die Schleife, solange der Benutzer »Nein« antwortet.

Auch TestDoLoopUntil wird mindestens einmal ausgeführt, weil der Test erst am Ende der Schleife steht. Diese wird ausgeführt, bis der Benutzer »Ja« anklickt.

In diesem Beispiel ist es egal, mit Ausnahme der ersten Prozedur, welche Variation Sie wählen. Je nach Aufgabe kann die Reihenfolge jedoch kritisch sein.

**Listing 3.22** Die *Do...Loop*-Variationen

```
Sub TestDoWhileLoop()
    Dim strAufforderung As String
    Dim lAntwort As Long

    strAufforderung = "Wollen Sie fortfahren?"
    Do While lAntwort = vbNo
        lAntwort = MsgBox(strAufforderung, vbYesNo + vbQuestion, strMSGTITEL)
    Loop
End Sub

Sub TestDoUntilLoop()
    Dim strAufforderung As String
    Dim lAntwort As Long

    strAufforderung = "Wollen Sie fortfahren?"
    Do Until lAntwort = vbYes
        lAntwort = MsgBox(strAufforderung, vbYesNo + vbQuestion, strMSGTITEL)
    Loop
End Sub

Sub TestDoLoopWhile()
    Dim strAufforderung As String
    Dim lAntwort As Long

    strAufforderung = "Wollen Sie fortfahren?"
    Do
        lAntwort = MsgBox(strAufforderung, vbYesNo + vbQuestion, strMSGTITEL)
    Loop While lAntwort = vbNo
End Sub

Sub TestDoLoopUntil()
    Dim strAufforderung As String
    Dim lAntwort As Long

    strAufforderung = "Wollen Sie fortfahren?"
    Do
        lAntwort = MsgBox(strAufforderung, vbYesNo + vbQuestion, strMSGTITEL)
    Loop Until lAntwort = vbYes
End Sub
```

**TIPP**

Sie können eine Do-Schleife mit der Anweisung `Exit Do` vorzeitig beenden.

For...Next

Muss eine Schleife eine bestimmte Anzahl Male ausgeführt werden, bedienen wir uns meistens der Anweisung `For...Next`. Die Syntax dafür ist

```
For Zähler = Anfang To Ende [Step Schritt]
```

Zähler ist ein Testwert, der eine Ganzzahl sein muss. Am Anfang wird er dem Wert `Anfang` zugewiesen, und die Schleife wird so lange wiederholt, bis Zähler den Wert von `Ende` erreicht oder überschritten hat. Das Listing 3.23 veranschaulicht dies.

**TIPP**

Am Ende einer `For...Next`-Schleife dürfen Sie die Zähler-Variable nach der `Next`-Anweisung schreiben. Dies hat auf den Code-Ablauf keine Einwirkung, hilft aber, ihn zu dokumentieren, wenn mehrere verschachtelte Schleifen vorhanden sind (Sie sehen sofort, welche `Next`-Zeile zu welcher `For`-Schleife gehört)

**Listing 3.23**

Mit `For...Next` wird eine Schleife eine bestimmte Anzahl Male durchlaufen

```
Sub TestForNext()
    Dim lZaehler As Long
    Dim lAnfang As Long
    Dim lEnde As Long

    lAnfang = 1
    lEnde = 5
    For lZaehler = lAnfang To lEnde
        MsgBox "Die Schleife wurde " & lZaehler & " mal ausgeführt.", _
            vbOKOnly + vbInformation, strMSGTITEL
    Next lZaehler
End Sub
```

Standardmäßig wird Zähler bei jeder Durchführung der Schleife automatisch um den Faktor Eins erhöht. Mit der Anweisung `Step` kann ein anderer, auch negativer, Faktor bestimmt werden. Ein Beispiel dafür zeigt der Code in Listing 3.24.

Es werden wahlweise Absätze aus der Paragraphs-Auflistung gelöscht. Wenn Sie dies mit der `For Each...Next`-Schleife täten, würde der Code in einem großen Dokument immer langsamer werden, da Word in der Auflistung immer wieder von vorn beginnen würde. Ein ähnliches Problem würde mit einer gewöhnlichen `For...Next`-Schleife auftreten. Arbeitet sich der Code jedoch vom letzten zum ersten Element durch, beansprucht die Neuindizierung der Auflistung weniger Zeit.

**HINWEIS**

Die Anweisung `For Each...Next` wird in Kapitel 5 behandelt.

**Listing 3.24**

Dieses Beispiel löscht alle Absätze mit der Formatvorlage »Standard«

```
Sub TestForNextStep()
    Dim para As Word.Paragraph
    Dim doc As Word.Document
    Dim lZaehler As Long
    Dim lAnfang As Long
    Dim lEnde As Long
```

**Listing 3.24** Dieses Beispiel löscht alle Absätze mit der Formatvorlage »Standard« (*Fortsetzung*)

```

Set doc = ActiveDocument
lAnfang = doc.Paragraphs.Count
lEnde = 1
Wird der Inhalt einer Auflistung geändert (hauptsächlich Objekte gelöscht),
'ist es u. U. wichtig, dass Sie sich von hinten nach vorn durch die Elemente arbeiten.
For lZaehler = lAnfang To lEnde Step -1
    Set para = doc.Paragraphs(lZaehler)
    If para.Style = "Standard" Then
        para.Range.Delete
    End If
Next lZaehler
End Sub

```

**TIPP**

Sie können eine For...Next-Schleife mit der Anweisung `Exit For` vorzeitig beenden.



Die Beispieldatei für die Abschnitte »Nützliche VBA-Funktionen«, »Bedingungen« und »Schleifen« finden Sie in der Beispieldatei *Bsp03\_04.doc* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap03*.

## Code im VB-Editor debuggen

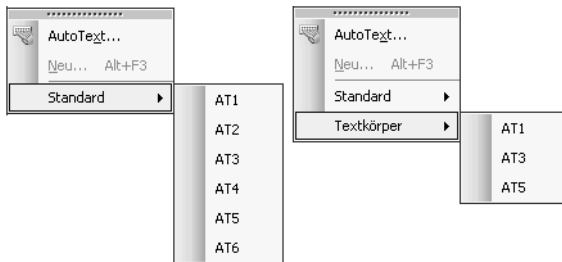
Vorausgesetzt, Sie haben `Option Explicit` eingeschaltet **und** verfügen über viel Erfahrung sowie eine übergroße Portion Glück, wird Ihr Code bei jedem ersten Mal ohne Fehler laufen. Ist Ihnen jedoch so viel Glück zuteil, wären Sie gut beraten, besser in einem Spielcasino anstatt vorm Bildschirm zu sitzen ...

Es ist allerdings eher die Ausnahme, dass Anwendungen beim ersten Versuch fehlerfrei laufen. Des VB-Editors IntelliSense, zusammen mit `Option Explicit`, verringern Tippfehler, können aber die Logik des Code-Aufbaus nicht nachprüfen. Dafür braucht es den menschlichen Verstand.

Der VB-Editor enthält etliche Werkzeuge, die bei der Fehlersuche hilfreich sind. Zuerst muss das momentane Resultat ausgewertet werden, um die mögliche Ursache herauszufinden. Bricht die Ausführung mit einer Fehlermeldung ab, erhalten wir erstens eine Information in der Fehlermeldung sowie häufig Gelegenheit, auf die Schaltfläche *Debug* zu klicken, um zur problematischen Codezeile zu springen.

Läuft die Anwendung ohne Fehler durch, liefert jedoch ein unerwartetes Ergebnis, bleibt uns nur, den Code Abschnitt für Abschnitt, ja sogar Zeile für Zeile, durcharbeiten, bis die Fehllogik (Bug) aufgespürt wurde. Die Abbildung 3.15 stellt ein solches Problem dar, das an die Diskussion über die For...Next-Anweisung im Abschnitt »Schleifen« in diesem Kapitel anschließt.

**Abbildg. 3.15** In einer *For Each*-Schleife wurde nur jeder zweite Eintrag bearbeitet



Die sechs AutoText-Einträge links in Abbildung 3.15 sind mit der Formatvorlage »Standard« formatiert und befinden sich demzufolge in der AutoText-Kategorie gleichen Namens. Es wurde entschieden, sie in die Kategorie »Textkörper« zu verschieben und diese Aufgabe programmatisch zu erledigen.

Nichts einfacher, denkt der VBA-Entwickler, setzt sich an die Tastatur und gibt die Prozedur in Listing 3.25 ein, die auch einwandfrei läuft. Pflichtbewusst schaut der Entwickler schnell im Menü (Abbildung 3.15) nach, in der Erwartung, alle Einträge würden sich in der zweiten Liste, rechts, befinden. Etwas konsterniert stellt er fest, dass nur jeder zweite verschoben wurde.

**Listing 3.25** Die Kategorie eines AutoTextEintrags wird geändert, indem dem Eintrag eine andere Formatvorlage zugewiesen wird

```
' Diese Prozedur soll jeden AutoText-Eintrag aus der Kategorie »Standard« in das Dokument
' einfügen, ihm jeweils die Formatvorlage »Textkörper« zuweisen und ihn anschließend
' wieder unter gleichem Namen als AutoText speichern.
Sub ForEachAutoText()
    Dim AT As Word.AutoTextEntry
    Dim tmpl As Word.Template
    Dim rng As Word.Range

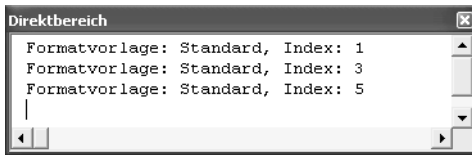
    Set tmpl = Documents("DebugTest.dot").AttachedTemplate
    Set rng = ActiveDocument.Range
    rng.Collapse wdCollapseEnd
    For Each AT In tmpl.AutoTextEntries
        If AT.StyleName = "Standard" Then
            Set rng = AT.Insert(Where:=rng, RichText:=True)
            rng.Style = ActiveDocument.Styles(wdStyleBodyText)
            tmpl.AutoTextEntries.Add Name:=AT.Name, Range:=rng
            rng.Delete
        End If
    Next
End Sub
```

Was tun? Da gibt es nur eines: Die Werte der Objekte und Variablen überprüfen, bis die Ursache gefunden ist.

Eine Übersicht der Werte erhalten Sie, wenn Sie an wichtigen Codestellen MsgBox-Anweisungen einbauen oder Debug.Print-Anweisungen benutzen. Statt den Ablauf ständig zu unterbrechen, schreibt Debug.Print die Meldungen in den Direktbereich, wie in Abbildung 3.16 ersichtlich.

```
If AT.StyleName = "Standard" Then
    Debug.Print "Formatvorlage: " & AT.StyleName & ", Index: " & AT.Index
```

Abbildg. 3.16 Die Anweisung *Debug.Print* gibt eine Meldung in den Direktbereich aus, statt sie auf dem Bildschirm anzuzeigen



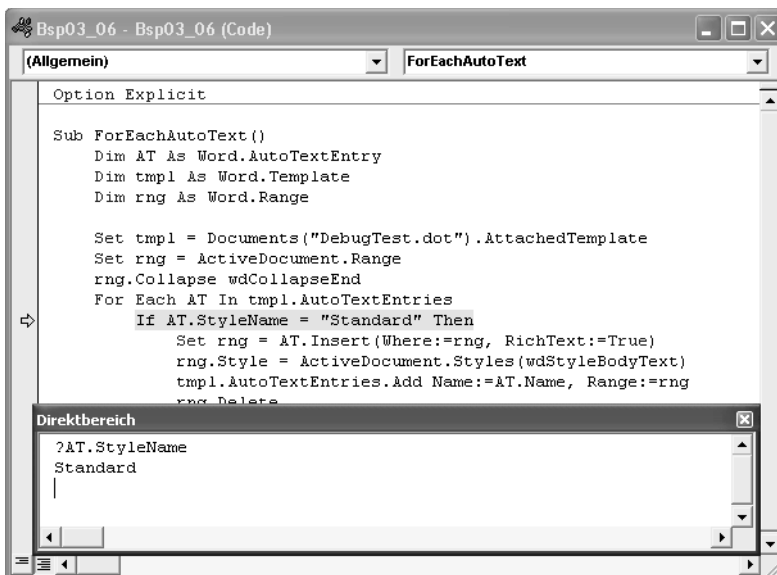
Unter Umständen ist es hilfreicher, die Werte bei der Ausführung jeder Codezeile zu kontrollieren. Dies bedeutet, die Prozedur muss Schritt für Schritt ausgeführt werden.

Der Befehl für die schrittweise Ausführung einer Prozedur befindet sich im Menü *Debuggen/Einzel-schritt*. Es wäre nun recht mühsam, für jede Codezeile das Menü zu öffnen und den Eintrag auszuwählen. Deshalb sollten Sie sich die Taste **[F8]** einprägen. Beim ersten Tastendruck wird die erste Codezeile (Sub *ForEachAutoText*) gelb hervorgehoben. Bei jedem weiteren wird die markierte Zeile ausgeführt und die nächste ausführbare hervorgehoben. (Die Variablendeklarationen werden nicht hervorgehoben; diese werden ausgewertet, bevor die Prozedur anfängt.)


Aber wie prüfen wir nun die Werte? Dafür gibt es mehrere Möglichkeiten:

- Den Mauszeiger über dem Eintrag ruhen lassen, bis die Information angezeigt wird.
- Im Direktbereich (wird mit **[Strg] + [G]** eingeblendet) den Ausdruck, mit einem Fragezeichen vorangestellt, eingeben, dann die **[↵]**-Taste drücken (beispielsweise `?AT.StyleName`). Der Wert erscheint in der nächsten Zeile des Fensters (Abbildung 3.17).
- Die Werte dem Überwachungsfenster zuweisen, wie in Abbildung 3.18 ersichtlich.

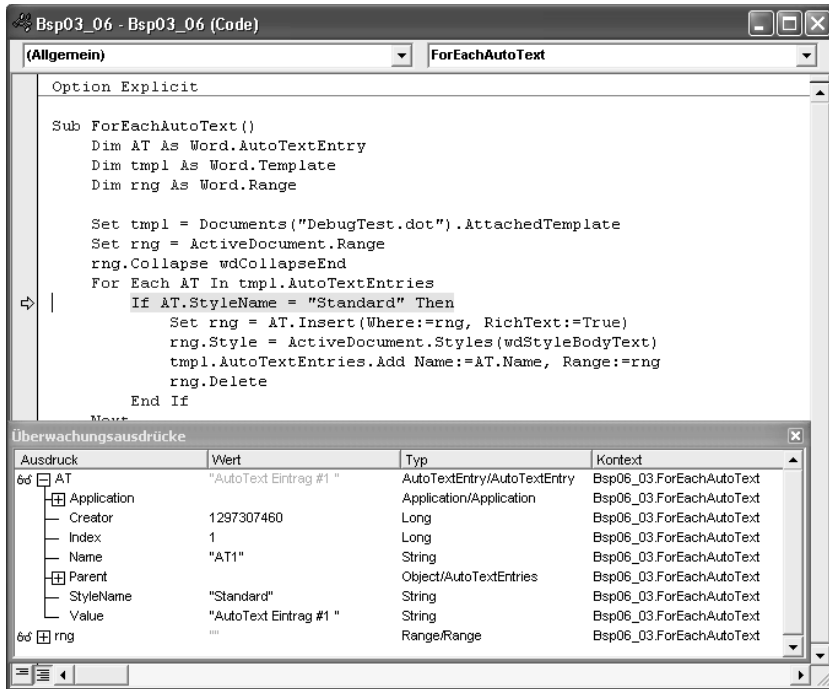
Abbildg. 3.17 Wert im Direktbereich prüfen. Die Hervorhebung und der Pfeil im Code-Fenster weisen auf die nächste Anweisung hin.



**TIPP**


Während der schrittweisen Ausführung dürfen Sie den Variablen auch andere Werte im Direktfenster zuweisen. Beispiel: lEnde = 7 (aus Listing 3.24) und anschließend die -Taste drücken.

Abbildg. 3.18 Im Überwachungsbereich können mehrere Werte gleichzeitig bei jedem Schritt geprüft werden

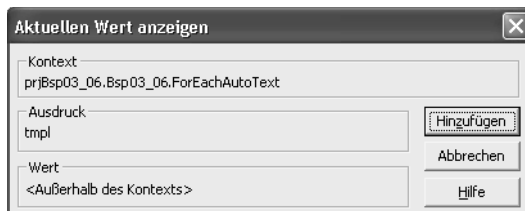


## Der Überwachungsbereich

Auch die Befehle für den Überwachungsbereich befinden sich im Menü *Debuggen*, und für die wichtigsten stehen Tastaturzuweisungen bereit:

- Um der Liste einen Wert hinzuzufügen, markieren Sie den Ausdruck im Code-Fenster und betätigen  + **F9**. Das Dialogfeld *Aktuellen Wert anzeigen* (siehe Abbildung 3.19) wird eingeblendet; klicken Sie auf *Hinzufügen*. (Ein Ausdruck darf vor sowie während der Code-Ausführung hinzugefügt werden.)

Abbildg. 3.19 Dem Überwachungsbereich einen Ausdruck hinzufügen



- Um einen Ausdruck aus der Liste zu entfernen, klicken Sie rechts darauf und wählen *Überwachung entfernen*.
- Die Spalte neben dem Ausdruck zeigt den Wert der standardmäßigen Eigenschaft an. Um weitere Informationen zu sehen, klicken Sie auf das Zeichen »+« links daneben.
- Beachten Sie, wie sich die Werte während des Ablaufs der Prozedur ändern.

Um auf unser Beispiel zurückzukommen, merkt der Entwickler, dass bei jeder Ausführung der Schleife der Index-Wert des AT-Objekts (AutoText-Eintrag) um zwei, statt eins, erhöht wird. Daraus schließt er, dass das Ersetzen des existierenden Eintrags durch den geänderten die Auflistung durcheinander bringt. (Weil sich der Inhalt der Auflistung ändert, wird jeder Eintrag, der einem bearbeiteten folgt, übersprungen, da dieser den Indexwert des bearbeiteten bekommen hat.)

Als nächstes versucht er es mit einer For...Next-Schleife wie in Listing 3.26. Auch diese Prozedur läuft ohne Fehlermeldung, und voller Zuversicht schaut der Entwickler im Menü *Einfügen/AutoText* nach. Dort findet sich nur der Eintrag *Textkörper*; die Kategorie »Standard« ist verschwunden. Schaut er jedoch unter *Textkörper* genau hin, fällt auf, dass nur fünf der sechs Einträge übernommen wurden. Der sechste Eintrag ist zwar in *Einfügen/AutoText/AutoText* noch vorhanden, aber irgendetwas ist schief gegangen (vermutlich ein Problem in der Word-Anwendung).

**Listing 3.26** Mit einer For...Next- anstatt einer For Each...Next-Schleife eine Auflistung bearbeiten

```
Sub ForNext()
    Dim AT As Word.AutoTextEntry
    Dim tmpl As Word.Template
    Dim rng As Word.Range
    Dim lZaehler As Long
    Dim lAnfang As Long
    Dim lEnde As Long

    Set tmpl = Documents("DebugTest.dot").AttachedTemplate
    Set rng = ActiveDocument.Range
    rng.Collapse wdCollapseEnd
    lAnfang = 1
    lEnde = tmpl.AutoTextEntries.Count
    For lZaehler = lAnfang To lEnde
        Set AT = tmpl.AutoTextEntries(lZaehler)
        If AT.StyleName = "Standard" Then
            Set rng = AT.Insert(Where:=rng, RichText:=True)
            rng.Style = ActiveDocument.Styles(wdStyleBodyText)
            tmpl.AutoTextEntries.Add Name:=AT.Name, Range:=rng
            rng.Delete
        End If
    Next
End Sub
```

Da jetzt klar ist, dass das Problem in der Schleife vorkommt, spart es Zeit, wenn der Code bis zu diesem Punkt ununterbrochen ausgeführt wird und erst auf der For-Zeile anhält. Um einen Haltepunkt zu setzen, klicken Sie in die Codezeile und betätigen dann die Taste **F9** (die Codezeile wird dunkelrot hervorgehoben). Drücken Sie **F5**, um die Ausführung zu starten; wenn sie beim Haltepunkt anhält, drücken Sie **F8** so lange, bis die Problemstelle gefunden wird.

Unser Entwickler setzt einen Haltepunkt, durchläuft die Zeilen innerhalb der Schleife und kommt dem Fehler doch nicht auf der Schliche. Er sieht aber, dass der letzte Eintrag korrekt mit der Format-

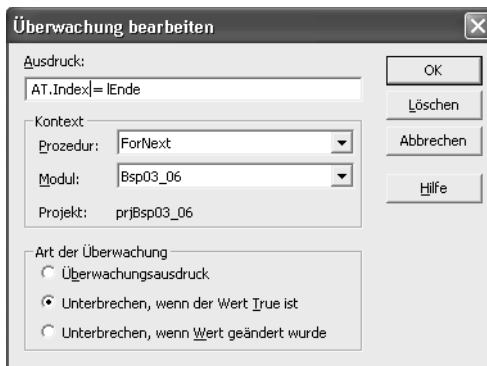
vorlage »Textkörper« formatiert wird, was eine Abfrage im Direktfenster bestätigt. Also probiert er, ob es etwas bringt, wenn der letzte AutoText-Eintrag ein zweites Mal erstellt wird.

Diesmal soll der Code nur anhalten, wenn der letzte AutoText-Eintrag bearbeitet wird. Mit einem Doppelklick wird der Ausdruck (AT) markiert; aus dem Kontextmenü wählen Sie den Eintrag *Überwachung hinzufügen*, und das Dialogfeld in Abbildung 3.20 wird eingeblendet. Im Feld oben können Sie den Ausdruck eingeben, den es zu testen gilt. Im unteren Bereich aktivieren Sie die Option *Unterbrechen, wenn der Wert True ist*. In diesem Beispiel testen wir, ob der Indexwert des AutoText-Eintrags gleich dem Endwert der Schleife ist.

### TIPP

Bitte beachten Sie, dass Sie auch testen und unterbrechen können, wenn sich ein Wert während der Ausführung ändert.

Abbildg. 3.20 Die Ausführung des Codes wird nur unterbrochen, wenn der eingegebene Ausdruck wahr wird



Unser Entwickler entfernt alle Haltepunkte (**Strg** + **⇧** + **F9**) und drückt dann **F5**. Die Ausführung hält in der Zeile mit `If AT.StyleName an`, und im Überwachungsbereich stellt er fest, dass der AutoText-Indexwert »6« ist. Mit **F8** geht er schrittweise vor, bis die Zeile `rng.Delete` gelb hervorgehoben wird; der AutoText-Eintrag wurde hinzugefügt. Nun möchte er die vorhergehende Zeile wiederholen.

Sie können jederzeit bei der schrittweisen Ausführung die Ausführungsstelle verschieben, entweder zurück oder weiter nach vorn. Führen Sie den Mauszeiger über den gelben Pfeil, ziehen Sie diesen dann mit gedrückter linker Maustaste nach oben bzw. nach unten. Oder klicken Sie in die Codezeile, wo die Weiterführung anfangen soll, und drücken dann **Strg** + **F9** (Menübefehl *Debuggen/Nächste Anweisung festlegen*), um den Ausführungspunkt zu versetzen.

Nachdem der Entwickler den gelben Pfeil eine Codezeile nach oben verschoben hat, drückt er nochmals **F5**; die Prozedur wird zu Ende ausgeführt. Er kontrolliert im Menü, ob dieses Mal alle sechs Einträge darin erscheinen, und stellt erleichtert fest, dass dies der Fall ist. Er ändert den Code wie in Listing 3.27, testet nochmals und fügt ihn dann seiner Code-Bibliothek hinzu. (Die C#-Version finden Sie in Listing 3.28.)



Listing 3.27 Alle AutoText-Einträge einer bestimmten Kategorie einer anderen Kategorie zuweisen.

```

'Da eine AutoText-Kategorie durch den Namen der Formatvorlage bestimmt wird, die dem Text
'bei Erstellung des Eintrags zugewiesen war, muss der Eintrag in ein Dokument eingefügt,
'neu formatiert und der Auflistung wieder hinzugefügt werden
Sub AutoTextKategorieAendern()
    Dim strKategorieAlt As String
    Dim strKategorieNeu As String
    Dim AT As Word.AutoTextEntry
    Dim tmp1 As Word.Template
    Dim rng As Word.Range
    Dim lZaehler As Long
    Dim lAnfang As Long
    Dim lEnde As Long

    strKategorieAlt = "Standard"
    strKategorieNeu = "Textkörper"
    Set tmp1 = Documents("DebugTest.dot").AttachedTemplate
    Set rng = ActiveDocument.Range
    rng.Collapse wdCollapseEnd
    lAnfang = tmp1.AutoTextEntries.Count
    lEnde = 1
    For lZaehler = lAnfang To lEnde Step -1
        Set AT = tmp1.AutoTextEntries(lZaehler)
        If AT.StyleName = strKategorieAlt Then
            Set rng = AT.Insert(Where:=rng, RichText:=True)
            rng.Style = ActiveDocument.Styles(strKategorieNeu)
            tmp1.AutoTextEntries.Add Name:=AT.Name, Range:=rng
            If lZaehler = lEnde Then
                tmp1.AutoTextEntries.Add Name:=AT.Name, Range:=rng
            End If
            rng.Delete
        End If
    Next
End Sub

```

**HINWEIS**

Mehr Informationen zu den in der Prozedur AutoTextKategorieAendern verwendeten Word-Objekten finden Sie in Kapitel 6.

Listing 3.28 Die C#-Version

```

private void AutoTextKategorieAendern_CS()
{
    try
    {
        wd.Application wdApp = (wd.Application)
            wdMarshal.GetActiveObject("Word.Application");

        object objTrue = (object) true;
        object objMissing = System.Reflection.Missing.Value;
        object objFileName = Application.StartupPath + "\\DebugTest.dot";
        string kategorieAlt = "Standard";
        string kategorieNeu = "Textkörper";
        wd.Document doc = wdApp.Documents.Open(ref objFileName, ref objMissing,
            ref objMissing, ref objMissing, ref objMissing, ref objMissing,
            ref objMissing, ref objMissing, ref objMissing, ref objTrue, ref objMissing,

```

**Listing 3.28** Die C#-Version (Fortsetzung)

```

        ref objMissing, ref objMissing, ref objMissing, ref objMissing);
    wd.Template tmpl = (wd.Template) doc.get_AttachedTemplate();
    wd.Range rng = doc.Content;
    object objCollapseEnd = (object) wd.WdCollapseDirection.wdCollapseEnd;
    rng.Collapse(ref objCollapseEnd);
    int lAnfang = tmpl.AutoTextEntries.Count;
    int lEnde = 1;
    for (int lZaehler = lAnfang; lZaehler >= lEnde; lZaehler --)
    {
        object objATentry = (object) lZaehler;
        wd.AutoTextEntry AT = tmpl.AutoTextEntries.get_Item(ref objATentry);
        if(AT.StyleName == kategorieAlt)
        {
            wd.Range rngAT = AT.Insert(rng, ref objTrue);
            object objStyleName = (object) kategorieNeu;
            rngAT.set_Style(ref objStyleName);
            tmpl.AutoTextEntries.Add(AT.Name, rngAT);
            if (lZaehler == lEnde)
            {
                tmpl.AutoTextEntries.Add(AT.Name, rngAT);
            }
            rngAT.Delete(ref objMissing, ref objMissing);
        }
    }
    //Das Word-Fenster anzeigen
    wdApp.Activate();
    wdMarshal.ReleaseComObject(wdApp);
    wdApp = null;
    MessageBox.Show("Fertig!");
}
catch (System.Runtime.InteropServices.COMException ex)
{
    MessageBox.Show(ex.Message);
}
}

```



Die Beispieldatei *Bsp03\_06.doc* mit den Listings zu diesem Abschnitt sowie die Datei *Debug-Test.dot* mit dem Beispiel der AutoText-Einträge finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap03*.

## Fehlerbehandlung

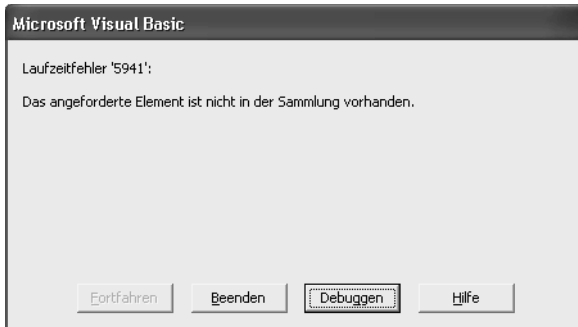
Während der Entwicklung haben Sie Ihre Anwendung sorgfältig getestet. Sie überlegten immer wieder, welche Fehlhandlungen und Problemsituationen vorkommen könnten. Und Sie haben durch den Einsatz von If- und Select-Anweisungen mögliche Fehlerursachen abgefangen, um Abstürze zu vermeiden. Ihre Anwendung scheint für den Gebrauch bereitzustehen.

Nun ist es mal so, dass Anwender alles »Unmögliche« tun, was eigentlich gar nicht vorgesehen und vom »Entwickler-Verstand« nicht voraussehbar war. Deshalb braucht eine robuste Anwendung unbedingt eine Fehlerbehandlung. Der Umfang dieses Buches ist zu begrenzt, um sich mit der The-

orie der Fehlerbehandlung eingehend zu befassen. Ein Kapitel über die VBA-Funktionalität wäre jedoch ohne eine kurze Zusammenfassung derselben unvollständig.

Ist in einer Anwendung keine Fehlerbehandlung aktiviert, zeigt die VBA-Umgebung beim Auftreten eines Fehlers eine Fehlermeldung an. Der Benutzer wird mit einer Laufzeit-Fehlermeldung, ähnlich wie in Abbildung 3.21, konfrontiert, wobei die Schaltfläche *Debuggen* unter Umständen gesperrt sein könnte. Im Prinzip bleibt ihm nichts anderes übrig, als auf *Beenden* zu klicken – die Anwendung ist abgestürzt.

**Abbildg. 3.21** Ein Laufzeitfehler bedeutet, dass der Code unter den herrschenden Umständen nicht erfolgreich ausgeführt werden konnte



An einem solch abrupten Ende einer Anwendung gibt es einiges auszusetzen:

- Der Benutzer kann die vorgenommene Aufgabe nicht zu Ende führen; er hat Zeit verloren und ist frustriert.
- Zudem wird er ob der für ihn kryptischen Fehlermeldung verwirrt und verunsichert.
- Die Fehlermeldung bietet zu wenig Informationen darüber, was wo passiert ist, als dass der Entwickler dem Problem nachgehen könnte.
- Unter Umständen gehen sogar wichtige Daten verloren.

Die Vorteile einer Fehlerbehandlung entsprechen ungefähr den aufgezählten Problemen:

- Die Anwendung wird, wenn überhaupt, nicht abrupt beendet.
- Muss dem Benutzer etwas gemeldet werden, kann der Inhalt informativ und hilfreich sein.
- Die Meldung bietet für den Entwickler wichtige Informationen, die er zur Problembehebung gebrauchen kann.
- Die Anwendung hat Gelegenheit, Daten zu speichern und Vorgänge rückgängig zu machen.

Eine Fehlerbehandlung lässt sich folgendermaßen in eine Prozedur integrieren:

1. Geben Sie nach der Variablendeklaration `On Error GoTo [Zeilenmarke]` in eine Codezeile ein, wobei der Begriff `[Zeilenmarke]` ein beliebiges Wort sein darf.
2. Vor der Codezeile `End Sub` geben Sie die Zeilenmarke, gefolgt von einem Doppelpunkt ein, beispielsweise: `Fehlerbehandlung:.` Das Wort muss genau dem Begriff in der `On Error GoTo`-Anweisung entsprechen. Die Zeilenmarke muss linksbündig sein; ein Einzug ist nicht erlaubt.
3. Nach der Zeilenmarke folgt der Code, der die Fehler behandelt.

Fehler-  
behand-  
lung  
einschal-  
ten

In Listing 3.29 sehen Sie ein vereinfachtes Beispiel. Einem Array (Datenfeld) werden vier Elemente zugewiesen. Danach wird der Benutzer aufgefordert, Anfangs- und Endzahlen einzugeben. Die Schleife wird, auf diesen Angaben basierend, ausgeführt und die Elemente des Arrays werden in den Direktbereich ausgegeben.

Die Eingabe einer negativen Zahl, einer Zahl größer als drei oder gar keiner Zahl resultiert in einem Laufzeitfehler. Dieses Mal erscheint statt der Microsoft-Fehlermeldung jedoch eine MsgBox mit einem (hoffentlich) hilfreichen Text.

**Listing 3.29** Die Grundrisse der Fehlerbehandlung: *On Error GoTo*-Anweisung, mit der Zeilenmarke *Fehlerbehandlung*

```
Sub FehlerBehandlung1()
    Dim lAnfang As Long
    Dim lZaehler As Long
    Dim lEnde As Long
    Dim aTest As Variant

    On Error GoTo Fehlerbehandlung
    aTest = Array("zero", "eins", "zwei", "drei")
    lAnfang = CLng(InputBox("Bitte die Anfangszahl eingeben"))
    lEnde = CLng(InputBox("Bitte die Endzahl eingeben"))
    For lZaehler = lAnfang To lEnde
        Debug.Print aTest(lZaehler)
    Next

Fehlerbehandlung:
    MsgBox "Fehler in der Prozedur FehlerBehandlung1" & vbCrLf _
        & Err.Description & vbCrLf & "Fehlernummer: " & Err.Number, _
        vbCritical + vbOKOnly
End Sub
```

Diese minimale Fehlerbehandlung hat noch einige Nachteile. Erstens erscheint die Fehlermeldung auch dann, wenn der Code eigentlich einwandfrei läuft. Zweitens wird die Anwendung immer noch abgebrochen.

Um nach erfolgreicher Ausführung die Prozedur zu beenden, wird vor der Zeilenmarke eine *Exit Sub*- bzw. *Exit Function*-Anweisung benötigt.

Um an einen gewissen Punkt der Prozedur zurückzukehren, wird wieder eine Zeilenmarke verwendet. Die Anweisung hierfür lautet *Resume [Zeilenmarke]*. Ein Beispiel sehen Sie in Listing 3.30.

#### **HINWEIS**

Soll die Ausführung mit der gleichen Codezeile wieder aufgenommen werden, die den Fehler verursacht hat, benutzen Sie die Anweisung *Resume* ohne Zeilenmarke.

Soll die Ausführung mit der Codezeile wieder aufgenommen werden, die an jene anschließt, die den Fehler verursacht hat, benutzen Sie die Anweisung *Resume Next* ohne Zeilenmarke.

**Listing 3.30** Läuft die Anwendung ohne Laufzeitfehler durch, wird die Prozedur in der Codezeile *Exit Sub* beendet. Kommt ein Fehler vor, wird die Ausführung zurück an die Zeilenmarke *Start* versetzt.

```
Sub FehlerBehandlung2()
    Dim lAnfang As Long
    Dim lZaehler As Long
    Dim lEnde As Long
```

**Listing 3.30** Lläuft die Anwendung ohne Laufzeitfehler durch, wird die Prozedur in der Codezeile *Exit Sub* beendet. Kommt ein Fehler vor, wird die Ausführung zurück an die Zeilenmarke *Start* versetzt. (Fortsetzung)

```

Dim aTest As Variant

On Error GoTo Fehlerbehandlung
aTest = Array("zero", "eins", "zwei", "drei")
Start:
lAnfang = CLng(InputBox("Bitte die Anfangszahl eingeben"))
lEnde = CLng(InputBox("Bitte die Endzahl eingeben"))
For lZaehler = lAnfang To lEnde
    Debug.Print aTest(lZaehler)
Next

Exit Sub

Fehlerbehandlung:
MsgBox "Fehler in der Prozedur FehlerBehandlung1" & vbCrLf _
    & Err.Description & vbCrLf & "Fehlernummer: " & Err.Number, _
    vbCritical + vbOKOnly
Resume Start
End Sub

```

Die Meldung der MsgBox ist noch verbesserungswürdig. Zudem müssen wir zwischen verschiedenen Fehlern unterscheiden können. Unser Beispiel enthält zwei verschiedene Arten von Fehlern: *Index außerhalb des gültigen Bereichs* (mit der Fehlernummer 9) sowie *Typen unverträglich* (mit der Fehlernummer 13). Der erste bedeutet, dass eine Indexzahl außerhalb des Arraybereichs (weniger als Null oder größer als drei) liegt; der zweite, dass der eingegebene Wert keine Zahl ist.

Die Prozedur in Listing 3.31 trägt diesen Umständen Rechnung. Im Fehlerbehandlungsabschnitt wird in einer Anweisung mit *Select Case* die Fehlernummer geprüft und entsprechend abgezweigt. Kommt ein unerwarteter Fehler vor, kommt *Case Else* mit der bisherigen Meldung zum Zug. In diesem Fall wird die Anwendung abgebrochen, indem zur neuen Zeilenmarke, *EndPunkt*, gesprungen wird.

#### WICHTIG

Diese letzte Zeilenmarke ist wichtig, wenn Ihre Anwendung immer bestimmte Handlungen ausführen soll, bevor sie beendet wird. Müssen beispielsweise Daten gespeichert oder offene Dateien geschlossen bzw. freigestellt werden, folgt der Code dafür hinter dieser Zeilenmarke.

**Listing 3.31** In diesem Beispiel werden mit einer *Select Case*-Anweisung in der Fehlerbehandlung die verschiedenen Fehler, die in der Prozedur vorkommen können, differenziert behandelt

```

Sub FehlerBehandlung3()
    Dim lAnfang As Long
    Dim lZaehler As Long
    Dim lEnde As Long
    Dim aTest As Variant

    On Error GoTo Fehlerbehandlung
    aTest = Array("zero", "eins", "zwei", "drei")
    Start:
    lAnfang = CLng(InputBox("Bitte die Anfangszahl eingeben"))
    lEnde = CLng(InputBox("Bitte die Endzahl eingeben"))

```

**Listing 3.31** In diesem Beispiel werden mit einer *Select Case*-Anweisung in der Fehlerbehandlung die verschiedenen Fehler, die in der Prozedur vorkommen können, differenziert behandelt (*Fortsetzung*)

```

For lZaehler = lAnfang To lEnde
    Debug.Print aTest(lZaehler)
Next

EndPunkt:
Exit Sub

Fehlerbehandlung:
Select Case Err.Number
    Case 9, 13
        MsgBox "Sie müssen eine Zahl zwischen 0 und 3 eingeben.", _
            vbInformation + vbOKOnly
        Resume Start
    Case Else
        MsgBox "Fehler in der Prozedur FehlerBehandlung1" & vbCrLf _
            & Err.Description & vbCrLf & "Fehlernummer: " & Err.Number, _
            vbCritical + vbOKOnly
        Resume EndPunkt
End Select
End Sub

```

In der VB-Sprache wird die Fehlerbehandlung auch benötigt, um Zustände und Werte zu testen, weil nicht alle Methoden und Anweisungen Rückgabewerte liefern. Ein gutes Beispiel hierfür ist die Automatisierung einer anderen Anwendung, wie in den Kapiteln 8, 9 und 10 beschrieben. Soll, wo vorhanden, eine laufende Instanz genutzt werden, wird die *GetObject*-Anweisung eingesetzt. Es kann aber nicht gewährleistet werden, dass die Anwendung tatsächlich schon läuft, und *GetObject* verursacht einen Fehler, wenn dies der Fall ist.

In einem solchen Fall wird die Fehlerfunktionalität mit der Anweisung *On Error Resume Next* für kurze Zeit ausgesetzt, wie Listing 3.32 veranschaulicht. Damit wird ein eventueller Fehler von der VB-Umgebung ignoriert und der Code weiter ausgeführt. Dieser Zustand ist natürlich sehr gefährlich; die Fehlerfunktionalität muss baldmöglichst – nach Prüfung der Eigenschaft *Err.Number* – wieder eingeschaltet werden, was mit einer gewöhnlichen *On Error GoTo*-Anweisung erfolgt. Enthält die Prozedur sonst keine Fehlerbehandlung, wird *On Error GoTo 0* eingesetzt, um die standardmäßige Fehlerfunktionalität einzuschalten.

#### WICHTIG

Es gibt Leute, die am Anfang einer Prozedur *On Error Resume Next* eingeben und die Fehlerfunktionalität gänzlich ausschalten, ohne sie wieder zu aktivieren. Sie haben das Gefühl, die Anwendung laufe damit besser, weil man mit Fehlermeldungen nicht ständig stört. Ja, natürlich. Aber wenn ein unerwartetes Ergebnis vorliegt, haben sie keine Ahnung, warum, und können den Fehler nicht finden. Fallen Sie nicht darauf herein, nur weil es schneller und einfacher aussieht! Lassen Sie die Fehlerfunktionalität eingeschaltet.

**Listing 3.32** *On Error Resume Next* schaltet die VB-Fehlerfunktionalität vorübergehend aus. Es ist wichtig, diese mit *On Error GoTo* baldmöglichst wieder einzuschalten.

```

Public xlApp as Object
Sub TestGetObject()
    On Error Resume Next
    Set xlApp = GetObject(, "Excel.Application")

```

**Listing 3.32** *On Error Resume Next* schaltet die VB-Fehlerfunktionalität vorübergehend aus. Es ist wichtig, diese mit *On Error GoTo* baldmöglichst wieder einzuschalten. (Fortsetzung)

```
If Err.Number = 429 Then
    Set xlApp = CreateObject("Excel.Application")
ElseIf Err.Number <> 0 Then
    MsgBox "Ein Problem ist aufgetreten. Excel konnte nicht gestartet werden." _
        & vbCr & Err.Description & vbCr & "Fehlernummer: " & Err.Number
End If
On Error GoTo 0
xlApp.Visible = True
End Sub
```

**Err-Objekt** Die VB-Umgebung enthält das Objekt Err, das wir schon in einigen Listings gesehen haben. Es hat mehrere Eigenschaften, von denen *Number* (die Fehlernummer) und *Description* (eine Beschreibung des Problems) die meist gebrauchten sind.

Treten keine Fehler während des Programmablaufs auf, hat *Err.Number* den Wert 0. Sonst beträgt sie einen von der Anwendung vorgegebenen Wert, wie die vorangehenden Beispiele zeigen.

Es gibt auch eine Methode für das Objekt *Err.Raise*. Sie ermöglicht »entwicklerdefinierte« Fehler (Fehler, die von Visual Basic wie Anwendungsfehler behandelt werden). Die Syntax:

```
Err.Raise (number, [source], [description], [helpfile], [helpcontext])
```

Detaillierte Informationen enthält die VBA-Hilfe zum Thema.

Das Prinzip wird veranschaulicht in Listing 3.33. Nach den Eingabeaufforderungen wird in einer *If*-Anweisung kontrolliert, ob der Anfangswert größer ist als der Endwert. Ist das der Fall, wird ein Fehler veranlasst (*Err.Raise*). Dabei werden Fehlernummer, -quelle sowie -beschreibung spezifiziert. Der Fehlerbehandlung wurde eine *Case*-Anweisung hinzugefügt, die diesen Fehler behandelt. Die Abbildung 3.22 zeigt die Fehlermeldung an.

Die Fehlernummer wurde als konstanter Wert am Anfang der Prozedur deklariert. Es ist ratsam, den VB-Konstantwert *vbObjectError* bei der Festlegung von Fehlernummern zu benutzen. So werden Konflikte zwischen den VB- und Anwendungsfehlernummern vermieden.

**Listing 3.33** Beispiel für die Verwendung des *Err*-Objekts

```
Sub FehlerBehandlung4()
    Dim lAnfang As Long
    Dim lZaehler As Long
    Dim lEnde As Long
    Dim aTest As Variant
    Const lEINGABEFEHLER As Long = vbObjectError + 100

    On Error GoTo Fehlerbehandlung
    aTest = Array("zero", "eins", "zwei", "drei")

Start:
    lAnfang = CLng(InputBox("Bitte die Anfangszahl eingeben"))
    lEnde = CLng(InputBox("Bitte die Endzahl eingeben"))
    If lAnfang > lEnde Then
        Err.Raise lEINGABEFEHLER, "Fehlerbehandlung4", _
            "Der Anfangswert ist größer als der Endwert."
```

**Listing 3.33** Beispiel für die Verwendung des *Err*-Objekts (Fortsetzung)

```

End If
For lZaehler = lAnfang To lEnde
    Debug.Print aTest(lZaehler)
Next

EndPunkt:
Exit Sub

Fehlerbehandlung:
Select Case Err.Number
    Case 9, 13
        MsgBox "Sie müssen eine Zahl zwischen 0 und 3 eingeben.", vbInformation + vbOKOnly
        Resume Start
    Case lEINGABEFEHLER
        MsgBox "Der Anfangswert muss unter dem Endwert liegen."
        & vbCr & Err.Description & vbCr & "Fehlernummer: " & Err.Number
        & vbCr & "in der Prozedur: " & Err.Source, vbCritical + vbOKOnly
        Resume Start
    Case Else
        MsgBox "Fehler in der Prozedur FehlerBehandlung1" & vbCr
        & Err.Description & vbCr & "Fehlernummer: " & Err.Number, vbCritical + vbOKOnly
        Resume EndPunkt
End Select
End Sub

```

**Abbildg. 3.22** Ein vom Entwickler definierter Fehler wird mit *Err.Raise* erzeugt

**HINWEIS**

Die Autoren würden die in diesen Beispielen gezeigte Fehlerbehandlung selten so einsetzen. Benutzereingaben werden stattdessen in Schleifen mit If-Anweisungen getestet und die Eingabeaufforderung so lange wiederholt, bis eine korrekt Eingabe erfolgt. Die dargestellten Beispiele veranschaulichen anhand von Konzepten, die bisher im Buch vorgestellt wurden, lediglich die Grundprinzipien der Fehlerbehandlung in der VB-Umgebung.

Noch eine letzte Bemerkung zur Fehlerhandlung in der VB-Umgebung. Wenn die Anwendung aus mehreren Prozeduren besteht, die einander aufrufen, werden unbehandelte Fehler (wenn On Error GoTo in der Prozedur nicht vorhanden ist) von der aufgerufenen Prozedur an die rufende Prozedur zurückgereicht. Enthält diese Prozedur eine Fehlerbehandlung, werden die zurückgereichten Fehler hier behandelt. Und so weiter, bis die oberste Ebene erreicht wird. Steht hier auch keine Fehlerbehandlung zur Verfügung, wird die Visual Basic-Fehlerbehandlung tätig, und es erfolgt eine Laufzeitfehlermeldung (wie in Abbildung 3.21).





Die Beispieldatei *Bsp03\_07.doc* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap03*.

# Dateisystem-Operationen

Der Zugriff auf das Dateisystem wird bei vielen Makros verwendet, in denen es oft darum geht, den Status einer Datei zu ermitteln. So will der Programmierer sicherstellen, dass eine Datei oder ein Verzeichnis vorhanden ist, bevor dieses bearbeitet wird. Es muss die Größe oder das Speicherdatum einer Datei ermittelt werden. Oder es werden überzählige Dateien von der Festplatte entfernt. Dies sind ein paar Beispiele, welche einen Zugriff auf das Dateisystem erfordern.

Anhand der nachstehenden Beispiele erhalten Sie Einblick in die vielen Möglichkeiten im Zusammenhang mit dem Dateisystem. Wir Autoren sind uns jedoch bewusst, dass die aufgezeigten Programmbeispiele nicht alle Bereiche des Themas abdecken.

## Alle Dateien eines Verzeichnisses auflisten

**Dir** Um in einem bestimmten Ordner alle Dateien bzw. alle Dateien mit der gleichen Dateinamenerweiterung aufzulisten, steht die *Dir*-Funktion zur Verfügung.

Die Funktion gibt den ersten Dateinamen zurück, der im angegebenen Verzeichnis mit dem angegebenen Suchmuster übereinstimmt. Damit alle Dateien ermittelt werden, muss die Funktion erneut aufgerufen werden. Für die folgenden Aufrufe darf jedoch kein Argument an die Funktion übergeben werden. Bei jedem erneuten Aufruf wird der nächste Dateiname zurückgeliefert. Wird keine weitere Datei mehr gefunden, so wird eine leere Zeichenkette ("" ) zurückgeliefert. Die *Dir*-Funktion unterstützt eine Suche mittels Platzhalterzeichen.

Im Listing 3.34 wird dieser Umstand genutzt, um eine Liste aller vorhandenen Dokumente im aktuellen Verzeichnis auszugeben.

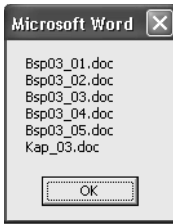
**Listing 3.34** Auflisten aller Dateinamen im aktuellen Verzeichnis

```
Sub AlleDateienAuflisten()
    Dim strDateiname As String
    Dim strDateiliste As String

    strDateiname = Dir$("*.*doc")
    Do While Not strDateiname = ""
        strDateiliste = strDateiliste & strDateiname & vbCrLf
        strDateiname = Dir
    Loop

    MsgBox strDateiliste
End Sub
```

Abbildg. 3.23 Alle vorhandenen Dokumente werden aufgelistet



In einem ersten Schritt wird die `Dir`-Funktion initialisiert. Diese Programmzeile liefert bereits einen ersten Treffer, also den Namen der ersten Dateien im aktuellen Verzeichnis zurück.

```
strDateiname = Dir$("*.*")
```

Jetzt wird eine `Do While...Loop`-Schleife so lange durchlaufen, bis ein wiederholter Aufruf der Funktion eine leere Zeichenkette zurückliefert.

```
Do While Not strDateiname = ""
```

Innerhalb der Schleife werden zwei Programmschritte ausgeführt: Der zuletzt gefundene Dateiname wird an die Liste der bereits ermittelten Dateinamen angehängt. Anschließend wird der Name der nächsten Datei abgefragt. Dieser Aufruf der Funktion erfolgt jetzt ohne die Angabe eines Arguments:

```
strDateiliste = strDateiliste & strDateiname & vbCrLf  
strDateiname = Dir
```

### Platzhalter für das Dateisystem

\*  
?

Für den Zugriff auf das Dateisystem werden vom Betriebssystem Microsoft Windows zwei Platzhalter unterstützt. Diese Platzhalter erlauben es, eine Gruppe von Dateien innerhalb des Dateisystems gleichzeitig anzusprechen.

Das Fragezeichen (?) ist der Platzhalter für ein *einzelnes* Zeichen. So können beispielsweise die drei Dateien *DateiA.doc*, *DateiB.doc* und *DateiC.doc* unter Verwendung des Platzhalters gleichzeitig gelöscht werden:

```
Kill "Datei?.doc"
```

Der Stern (\*) ist Platzhalter für eine *beliebige Menge* von Zeichen. So können beispielsweise alle Dokumente mit der Dateinamenerweiterung *.doc* und alle Dokumentvorlagen mit der Dateinamenerweiterung *.dot* gleichzeitig gelöscht werden:

```
Kill "*.do?"
```

## Verzeichnisname mit Backslash ergänzen

Eine Datei innerhalb des Dateisystems wird durch die Angabe des Dateinamens und des Ordners, in dem die betreffende Datei gespeichert ist, eindeutig bestimmt. Die einzelnen Unterordner sowie der Dateiname werden durch die Verwendung des Backslash (\) voneinander getrennt.

Ein gültiger Name eines Ordners endet nie mit einem Backslash (beispielsweise *C:\Programme*). Diese Regel hat jedoch eine Ausnahme. Das Wurzelverzeichnis zu jedem Laufwerk endet mit einem Backslash (beispielsweise *C:\*).

Müssen für den Zugriff auf das Dateisystem zwei Variablen miteinander verknüpft werden (die eine Variable enthält den Ordnernamen, die andere den Dateinamen), muss sichergestellt sein, dass das Resultat dieser Verknüpfung ein gültiger Dateiname ist:

```
strDateiname = strPfad & strDateiname
```

Anhand der vorstehenden Codezeile kann nicht sichergestellt werden, dass in jedem Fall ein gültiger Dateiname erzeugt wird, da die Variable *strPfad* nicht zwingend mit einem Backslash enden muss.

In Listing 3.35 wird mittels einer eigenen Funktion diesem Umstand Rechnung getragen. Die Funktion hängt bei Bedarf den fehlenden Backslash an den Ordnernamen an.

**Listing 3.35** Beim Verknüpfen mit Ordnernamen wird geprüft, ob der Pfad mit einem Backslash endet

```
Sub VerzeichnisnameMitBackslashErgänzen()
    Dim strPfad As String
    Dim strDateiname As String

    strPfad = "C:\Programme\Microsoft Office\Office10"
    strDateiname = "Winword.exe"

    strDateiname = fktPfadInklBackslash(strPfad) & strDateiname

    MsgBox strDateiname
End Sub

Public Function fktPfadInklBackslash( _
    ByVal strPfad As String) _
    As String
    'Die Funktion kontrolliert, ob der übergebene Pfad
    'als letztes Zeichen einen Backslash aufweist.
    'Falls nicht wird dieser angehängt.
    If Not (Right$(strPfad, 1) = Application.PathSeparator) Then
        strPfad = strPfad & Application.PathSeparator
    End If

    fktPfadInklBackslash = strPfad
End Function
```

Zusätzlich zur eigentlichen Verknüpfung der beiden Variablen wird geprüft, ob die Variable *strPfad* bereits mit einem Backslash endet. Ist dies nicht der Fall, wird das benötigte Trennzeichen durch die Funktion *fktPfadInklBackslash* eingefügt:

```
strDateiname = fktPfadInklBackslash(strPfad) & strDateiname
```

Die Arbeitsweise der Funktion `fktPfadInklBackslash` ist schnell erklärt. Am Argument `strPfad` wird das erste Zeichen von rechts eingelesen und mit dem gültigen Trennzeichen verglichen. Handelt es sich dabei nicht um das gesuchte Trennzeichen, wird die Variable um das entsprechende Zeichen, also einen Backslash, erweitert und dem Rückgabewert zugewiesen.

Abbildg. 3.24 Der fehlende Backslash in der Variable `strPfad` wurde angehängt



## Prüfen, ob eine bestimmte Datei vorhanden ist

Bevor mit einer bestimmten Datei gearbeitet werden kann, sollte geprüft werden, ob die betreffende Datei auf dem Dateisystem überhaupt vorhanden ist. Dies kann basierend auf dem Wissen aus dem Abschnitt »Alle Dateien eines Verzeichnisses auflisten« in diesem Kapitel mit der `Dir`-Funktion ermittelt werden.

```
If Not Dir("C:\Temp\Test.doc") = "" Then
```

Der Aufruf der `Dir`-Funktion, ohne Verwendung eines Platzhalterzeichens, liefert den Namen der gesuchten Datei zurück, sofern diese gefunden wird. Ansonsten wird eine leere Zeichenkette zurückgeliefert.

**WICHTIG** Wir raten dringend davon ab, die `Dir`-Funktion für diese Aufgabe zu verwenden. Der Grund dazu liegt in der erneuten Initialisierung der betreffenden Funktion durch die Angabe eines Dateinamens.

Zur Veranschaulichung kann das Listing 3.34 verwendet werden. Würden innerhalb der Schleife nicht nur die gefundenen Dateinamen an eine Variable angehängt, sondern ein zusätzlicher Aufruf in eine andere Funktion erfolgen, bestünde die Gefahr, dass in einer untergeordneten Funktion die `Dir`-Funktion neu initialisiert wird und somit nicht alle Dateien des Ausgangsverzeichnisses bearbeitet werden.

**GetAttr** In Listing 3.36 wird ohne die Verwendung der `Dir`-Funktion geprüft, ob sich eine bestimmte Datei auf dem Dateisystem befindet. Um dies zu erreichen, wird nicht die Existenz der Datei im Dateisystem, sondern es werden deren Dateiattribute mittels `GetAttr` ermittelt.

Listing 3.36 Anhand der Dateiattribute prüfen, ob eine Datei überhaupt vorhanden ist

```
Sub PrüfenObDateiVorhandenIst()
    MsgBox fktExistiertDatei("C:\BOOTLOG.TXT")
    MsgBox fktExistiertDatei("C:\Temp\Test.doc")
    MsgBox fktExistiertDatei("C:\Programme")
End Sub

Public Function fktExistiertDatei( _
    ByVal strDateiname As String) _
```

Listing 3.36 Anhand der Dateiattribute prüfen, ob eine Datei überhaupt vorhanden ist (Fortsetzung)

```

As Boolean
'Die Funktion versucht, die Dateiattribute der gesuchten Datei zu ermitteln.
'Der ermittelte Wert darf nicht mit jenem für einen Ordner übereinstimmen.
'Der Fehler, falls die Datei nicht vorhanden ist, wird mittels »On Error« übersprungen.
Const intATTR_NOTFILE = vbDirectory + vbVolume

On Error Resume Next
fktExistiertDatei = CBool((GetAttr(strDateiname) And intATTR_NOTFILE) = 0)
End Function

```

Wie aus Tabelle 3.4 ersichtlich, sind den einzelnen Attributen Zahlenwerte zugewiesen. Der Rückgabewert wird bitweise mit der Konstanten `intATTR_NOTFILE` verglichen. Die Konstante enthält die Werte aller *Nicht*-Dateiattribute.

Der bitweise Vergleich mit einer *Datei* liefert einen Wert gleich Null. Das Ergebnis dieser bitweisen Überprüfung wird mit Null verglichen. Dieser Vergleich gibt den Wert »Wahr« zurück.

Der bitweise Vergleich mit einem *Ordner* liefert hingegen einen Wert ungleich Null. Das Ergebnis dieser bitweisen Überprüfung wird ebenfalls mit Null verglichen. Dieser Vergleich gibt den Wert »Falsch« zurück, da die Zahl eben nicht Null entspricht.

Ist die gesuchte Datei nicht vorhanden, wird ein Fehler ausgelöst. Die vorgängige Anweisung `On Error Resume Next` bewirkt, dass die betreffende Zeile nicht ausgewertet wird, das Programm arbeitet weiter. Schlussendlich bleibt ein leerer Vergleich mit Null, der wiederum einen Fehler auslöst. Die Typ-Umwandlungsfunktion `CBool` gibt in diesem Fall automatisch den Wert »Falsch« zurück.

Tabelle 3.4 Rückgabewerte der *GetAttr*-Funktion

Konstante	Wert	Beschreibung
<code>vbNormal</code>	0	Normal
<code>vbReadOnly</code>	1	Schreibgeschützt
<code>vbHidden</code>	2	Versteckt
<code>vbSystem</code>	4	Systemdatei (beim Macintosh nicht verfügbar)
<code>vbVolume</code>	8	Laufwerksbezeichnung
<code>vbDirectory</code>	16	Verzeichnis, Ordner
<code>vbArchive</code>	32	Datei wurde seit dem letzten Sichern geändert (beim Macintosh nicht verfügbar).
<code>vbAlias</code>	64	Angegebener Dateiname ist ein Alias (nur beim Macintosh verfügbar).

## Prüfen, ob ein bestimmter Ordner vorhanden ist

Bevor auf einen bestimmten Ordner zugegriffen wird, sollte geprüft werden, ob das betreffende Verzeichnis auf dem Dateisystem überhaupt vorhanden ist. Dies könnte basierend auf dem Wissen aus dem Abschnitt »Alle Dateien eines Verzeichnisses auflisten« in diesem Kapitel wiederum mit der `Dir`-Funktion ermittelt werden:

```
If Not Dir("C:\Temp", vbDirectory) = "" Then
```

Doch wie bereits im Abschnitt »Prüfen, ob eine bestimmte Datei vorhanden ist« weiter vorne in diesem Kapitel erläutert, kann die Verwendung der Dir-Funktion zu Problemen führen. Um diese zu umgehen, wird der Programmcode aus Listing 3.36 verwendet und leicht angepasst.

In Listing 3.37 wird ebenfalls ohne die Verwendung von Dir-Funktion geprüft, ob sich ein bestimmter Ordner auf dem Dateisystem befindet. Hierfür wird nicht die Existenz des Verzeichnisses im Dateisystem, sondern es werden dessen Dateiattribute mittels GetAttr ermittelt.

**Listing 3.37** Anhand der Dateiattribute prüfen, ob ein Ordner überhaupt vorhanden ist

```
Sub PrüfenObOrdnerVorhandenIst()
    MsgBox fktExistiertOrdner("C:\Programme")
    MsgBox fktExistiertOrdner("C:\")
    MsgBox fktExistiertOrdner("C:\Temp\Test.doc")
End Sub

Public Function fktExistiertOrdner( _
    ByVal strOrdnername As String) _
    As Boolean
    'Die Funktion versucht, die Dateiattribute des gesuchten Ordners zu ermitteln. Der Fehler,
    'falls der Ordner nicht vorhanden ist, wird mittels »On Error« übersprungen.
    On Error Resume Next
    If Right$(strOrdnername, 1) = Application.PathSeparator Then
        strOrdnername = Left$(strOrdnername, Len(strOrdnername) - 1)
    End If
    fktExistiertOrdner = CBool((GetAttr(strOrdnername) And vbDirectory))
End Function
```

Wie aus Tabelle 3.4 ersichtlich, sind den einzelnen Attributen Zahlenwerte zugewiesen. Der Rückgabewert wird bitweise mit der Konstanten vbDirectory verglichen.

Der bitweise Vergleich mit einem **Ordner** liefert den Wert 16. Die Typ-Umwandlungsfunktion CBool gibt für jeden Wert ungleich Null den Wert »Wahr« zurück.

Der bitweise Vergleich mit einer **Datei** liefert hingegen einen Wert gleich Null. Die Typ-Umwandlungsfunktion CBool gibt für jeden Wert gleich Null den Wert »Falsch« zurück.

Ist der gesuchte Ordner nicht vorhanden, wird ein Fehler ausgelöst. Die vorgängige Anweisung On Error Resume Next bewirkt, dass die betreffende Zeile nicht ausgewertet wird, das Programm arbeitet weiter. Die Typ-Umwandlungsfunktion CBool gibt in diesem Fall automatisch den Wert »Falsch« zurück.

Am Anfang der Funktion wird sichergestellt, dass der übergebene Ordnername nicht mit einem Backslash endet. Trotzdem kann überprüft werden, ob der Wurzelordner (beispielsweise C:\) vorhanden ist. Die Prüfung erfolgt in diesem Fall nicht auf den Wurzelordner (C:\), sondern auf den aktuellen Ordner (C:). Da zu jedem aktuellen Ordner eines bestimmten Laufwerks immer ein Wurzelordner vorhanden ist, reicht die Überprüfung des aktuellen Ordners aus.

## Prüfen, ob eine Datei von jemanden im Zugriff ist

Bevor mit einer bestimmten Datei gearbeitet werden kann, muss sichergestellt werden, dass diese Datei vorhanden ist. Sollen an der betreffenden Datei Änderungen vorgenommen werden, sollte zusätzlich geprüft werden, ob die Datei nicht bereits von einem anderen Anwender bearbeitet wird.

Vom Betriebssystem werden Dateien automatisch für einen weiteren Zugriff gesperrt, wenn eine Datei im Änderungsmodus geöffnet ist. Es wird dabei unterschieden, ob die Datei mehrmals oder nur einmal, also exklusiv, geöffnet werden kann. Dieser Umstand wird in Listing 3.38 genutzt, um die entsprechende Prüfung durchzuführen.

**Listing 3.38** Prüfen, ob ein exklusiver Zugriff auf eine Datei möglich ist

```
Sub PrüfenObDateiImZugriffIst()
    MsgBox fktIstDateiBereitsGeöffnet("C:\BOOTLOG.TXT")
    MsgBox fktIstDateiBereitsGeöffnet(ThisDocument.FullName)
End Sub

Function fktIstDateiBereitsGeöffnet( _
    ByVal strDateiname As String) _
    As Boolean
    'Die Funktion versucht, eine Datei exklusiv zu öffnen. Wird die Datei bereits von
    'einem anderen Prozess verwendet, schlägt dieser Versuch fehl.
    On Error Resume Next
    Open strDateiname For Binary Access Read Lock Read As #1
    Close #1
    fktIstDateiBereitsGeöffnet = CBool(Err.Number)
End Function
```

Die Funktion versucht die Datei exklusiv im Änderungsmodus zu öffnen. Ist die Datei bereits geöffnet, schlägt dieser Versuch fehl. Die vorab aufgerufene Anweisung `On Error Resume Next` bewirkt, dass die betreffende Zeile nicht ausgewertet wird, das Programm arbeitet weiter. Die Typ-Umwandlungsfunktion `CBool` wertet die aktuelle Fehlernummer aus. Für jeden Wert ungleich Null wird der Wert »Wahr« zurückgegeben.

## Datei löschen

**Kill** Um eine Datei auf dem Datenträger zu löschen, steht die `Kill`-Anweisung zur Verfügung. Doch wie bereits aufgezeigt, sollte zuerst geprüft werden, ob der Versuch, die Datei zu löschen, Erfolg haben könnte.

### HINWEIS

Es zeugt von schlechtem Programmierstil, wenn mögliche Fehlerquellen innerhalb des Programms nicht bearbeitet werden und deshalb die Anweisung `On Error Resume Next` großzügig im Programmcode eingesetzt wird.

In Listing 3.39 wird versucht, vor dem Aufruf der `Kill`-Anweisung auf mögliche Fehler (Datei nicht vorhanden, Datei bereits geöffnet, berücksichtigen der Dateiattribute) zu reagieren. Dies wird mit den bereits erstellten Funktionen aus Listing 3.36 und Listing 3.38 bewerkstelligt.

**Listing 3.39** Datei auf dem Datenträger löschen und mögliche Fehler bearbeiten

```

'*** Achtung diese Funktion löscht ohne Rückfrage
'*** Dateien von der Festplatte, sofern diese
'*** tatsächlich vorhanden sind.
Sub DateiLöschen()
    MsgBox fktDateiLöschen("C:\Temp\TestA.doc")
    MsgBox fktDateiLöschen("C:\Temp\TestB.doc")
End Sub

Function fktDateiLöschen( _
    ByVal strDateiname As String) _
    As Boolean

    Const intATTR NODELETE = vbReadOnly + vbHidden
    Dim bFlag As Boolean

    'Ist die Datei vorhanden
    bFlag = fktExistiertDatei(strDateiname)

    'Ist die Datei bereits geöffnet
    If bFlag Then
        bFlag = Not fktIstDateiBereitsGeöffnet(strDateiname)
    End If

    'Ist die Datei weder schreibgeschützt noch versteckt
    If bFlag Then
        bFlag = CBool((GetAttr(strDateiname) And intATTR_NODELETE) = 0)
    End If

    'Datei löschen
    If bFlag Then
        Kill strDateiname
    End If

    'Konnte die Datei wirklich entfernt werden.
    bFlag = Not fktExistiertDatei(strDateiname)
    End If
End If
fktDateiLöschen = bFlag
End Function

```

Die Funktion liefert einen logischen Wert zurück, der Auskunft gibt, ob die übergebene Datei auf dem Dateisystem tatsächlich gelöscht werden konnte.

**HINWEIS**

Es wäre durchaus möglich, die Funktion so anzupassen, dass ein aussagekräftiger Fehlercode für jeden abgefangenen Fehler zurückgegeben wird.

Die ersten beiden Prüfungen, um das Auftreten eines Laufzeitfehlers zu verhindern, basieren auf den erstellten Funktionen. Die Auswertung der Dateiattribute ist ebenfalls in Listing 3.37 integriert und detailliert beschrieben.



## Letztes Speicherdatum einer Datei ermitteln

FileDate-  
Time

Um das letzte Speicherdatum einer Datei bzw. eines Ordners auf dem Datenträger zu ermitteln, steht die `FileDateTime`-Funktion zur Verfügung. Auch in diesem Fall ist es sinnvoll, zuerst das Umfeld dahingehend zu überprüfen, ob der Versuch, das Datum des Verzeichniseintrags zu ermitteln, Erfolg haben könnte.

### TIPP

Wir empfehlen grundsätzlich, eine Sammlung von einzelnen Funktionen und Prozeduren mit genau definiertem Funktionsumfang (beispielsweise eine Datei löschen, das Datum einer Datei ermitteln usw.) anzulegen. So wird vermieden, dass Sie sich bei jedem Aufruf des Original-VBA-Befehls auch noch um die möglichen Fehlerfälle kümmern müssen. Dies ist nicht mehr nötig, da dies zentral innerhalb der einzelnen Funktion aus der Sammlung bereits erfolgt ist.

In Listing 3.40 wird sichergestellt, dass ein gültiger Verzeichniseintrag ohne abschließenden Backslash vorhanden ist. In einem zweiten Schritt wird geprüft, ob es sich um einen gültigen Verzeichniseintrag handelt.

**Listing 3.40** Ermitteln des Speicherdatums eines Verzeichniseintrags

```
Sub DateiDatumUndZeitErmittleIn()
    MsgBox fktDateiDatumUndZeitErmittleIn("C:\Temp\", "dd/ mmmm yyyy")
    MsgBox fktDateiDatumUndZeitErmittleIn("C:\Temp\Test.doc")
End Sub

Public Function fktDateiDatumUndZeitErmittleIn( _
    ByVal strDateiname As String, _
    Optional ByVal strFormat As String = "dd/mm/yyyy hh:nn:ss") _
    As String
    'Die Funktion ermittelt das Systemdatum einer Datei oder
    'eines Verzeichnisses oder gibt eine leere Zeichenkette
    'zurück, wenn 'strDateiname' nicht vorhanden ist.

    'Pfad hat am Ende kein Backslash
    If Right$(strDateiname, 1) = "\" Then
        strDateiname = Left$(strDateiname, Len(strDateiname) - 1)
    End If

    'Ist die Datei/Verzeichnis vorhanden
    If fktExistiertDatei(strDateiname) Or fktExistiertOrdner(strDateiname) Then
        fktDateiDatumUndZeitErmittleIn = Format$(FileDateTime(strDateiname), strFormat)
    End If
End Function
```

### HINWEIS

Das aktuelle Programmbeispiel baut auf die beiden bereits vorgestellten Funktionen aus Listing 3.36 und Listing 3.37 auf. Damit das Beispiel ausgeführt werden kann, müssen diese Funktionen im Zugriff sein.

Als Besonderheit kann das gewünschte Ausgabeformat der Funktion `fktDateiDatumUndZeitErmittleIn` als zusätzlicher Parameter übergeben werden. Wird kein Wert eingetragen, wird der definierte Standardwert übernommen. Dies wird durch die Angabe des Schlüsselworts `Optional` und der zusätzlichen Zuweisung eines Werts innerhalb der Deklarationszeile erreicht:

```
Optional ByVal CFormat As String = "dd/mm/yyyy hh:nn:ss")
```

Abbildg. 3.25 Unterschiedlicher Rückgabewert der Funktion in Abhängigkeit zum Parameter *strFormat*



## Größe einer Datei ermitteln

FileLen In Listing 3.41 wird zuerst geprüft, ob die gesuchte Datei auf dem Datenträger vorhanden ist. Ist dies der Fall, wird die Größe der Datei mittels der FileLen-Funktion ermittelt und in die entsprechende Maßeinheit umgerechnet. Im abschließenden Programmschritt wird das Ausgabeformat bestimmt.

Listing 3.41 Ermitteln der Größe einer Datei und Umrechnen des Resultats in die gewünschte Maßeinheit

```
Option Explicit

Enum eDateigrösse
    eDG_Byte = 1
    eDG_kByte = 1024
    eDG_MByte = 1048576
End Enum

Sub DateiGrösseErmitteln()
    MsgBox fktDateiGrösseErmitteln("C:\Temp\TestA.doc", eDG_Byte)
    MsgBox fktDateiGrösseErmitteln("C:\Temp\TestA.doc", eDG_MByte)
End Sub

Public Function fktDateiGrösseErmitteln( _
    ByVal strDateiname As String, _
    ByVal lngDG As eDateigrösse, _
    Optional ByVal strFormat As String = "###,###,###,##0") _
    As String
    'Die Funktion ermittelt die Dateigröße einer Datei oder
    'gibt den Wert -1 zurück, wenn 'strDateiname' nicht
    'vorhanden ist.

    Dim lngGrösse As Long

    If fktExistiertDatei(strDateiname) Then
        lngGrösse = FileLen(strDateiname)
        'Umrechnen in gewünschte Dateigröße
        lngGrösse = (lngGrösse - 1) \ lngDG + 1
    Else
        lngGrösse = -1
    End If

    fktDateiGrösseErmitteln = Format$(CStr(lngGrösse), strFormat)
End Function
```

**HINWEIS** Das aktuelle Programmbeispiel baut auf die beiden bereits vorgestellten Funktionen aus Listing 3.36 auf. Damit das Beispiel ausgeführt werden kann, muss diese Funktion im Zugriff sein.

Das gewünschte Ausgabeformat der Funktion `fktDateiGrösseErmitteln` kann als zusätzlicher Parameter übergeben werden. Wird kein Wert eingetragen, wird der definierte Standardwert übernommen. Dies wird durch die Angabe des Schlüsselworts `Optional` und der zusätzlichen Zuweisung eines Werts innerhalb der Deklarationszeile erreicht:

```
Optional ByVal strFormat As String = "###,###,###,##0"
```

Als weitere Besonderheit der Funktion `fktDateiGrösseErmitteln` kann die gewünschte Maßeinheit für den Rückgabewert übergeben werden. Dazu wurde auf Modulebene eine entsprechende Aufzählung deklariert. Die zugewiesenen Werte entsprechen gleichzeitig den Umrechnungsfaktoren:

```
Enum eDateigrösse
    eDG_Byte = 1
    eDG_kByte = 1024
    eDG_MByte = 1048576
End Enum
```

Indem innerhalb der Deklarationszeile der Parameter `lngDG` nicht vom Typ `Long`, sondern vom Typ `eDateigrösse` deklariert wurde, wird vom Editor automatisch IntelliSense unterstützt:

```
ByVal lngDG As eDateigrösse
```

**Abbildg. 3.26** Unterschiedlicher Rückgabewert der Funktion in Abhängigkeit der beiden Parameter



Die aufgeführten Codesequenzen finden Sie in der Beispieldatei *Bsp03\_05.doc* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap03*.

# Zusammenfassung

In diesem Kapitel wurden die Grundlagen zu VBA vermittelt. Dabei ging es um allgemeingültiges Wissen wie Variablen, Konstanten, Bedingungen usw.

- Als Erstes wurde der Umgang mit Variablen, deren Standard-Datentypen, die Sichtbarkeit derselben und die Übergabe an Prozeduren vorgestellt (Seite 78 ff.).
- In einem zweiten Abschnitt wurden die Konstanten (Seite 91 ff.) und die benutzerdefinierten Typen (Seite 93 ff.) erläutert.
- Ein weiterer Abschnitt widmete sich nützlichen VBA-Funktionen (Seite 100 ff.), Programmbedingungen und Schleifen (Seite 105 ff.).
- Ebenso wurde aufgezeigt, wie der Programmcode mit dem Debugger untersucht (Seite 111 ff.) und auffällige Programmfehler mittels einer Fehlerbehandlung aufgefangen werden können (Seite 118 ff.).
- Praktische Beispiele zum Dateisystem (Seite 125 ff.) runden das erlernte in diesem Kapitel ab.

## Kapitel 4

# Windows-APIs in VBA nutzen

### In diesem Kapitel:

Aufbau der API-Funktionen	138
Kombinieren und Abschließen von Pfaden	139
Datei über die Dateieindung ausführen	143
Zugriff auf Registry-Einträge	147
Zugriff auf INI-Dateien	152
Name des angemeldeten Benutzers ermitteln	157
Verarbeitung für eine bestimmte Zeit unterbrechen	159
Wave-Datei abspielen	160
Tastenstatus abfragen	163
Zusammenfassung	168

Mit VBA lassen sich viele Aufgaben und Abläufe realisieren, solange man sich innerhalb der Office-Programme bewegt. Aber nicht immer ist eine Lösung alleine mit den VBA-Befehlen und VBA-Möglichkeiten die einfachste oder kürzeste.

In vielen Fällen ist es nicht sinnvoll oder sogar unmöglich, mit VBA-Funktionen bestimmte Aufgaben erledigen zu wollen.

In diesen Fällen kann der Entwickler unter Windows auf Tausende von Befehlen und fertigen Funktionen zugreifen, die das Windows-Betriebssystem zur Verfügung stellt und auf die das Betriebssystem und die Systemprogramme selbst zugreifen. Der Zugriff auf diese internen Funktionen erfolgt über API-Funktionen (Application Program Interface), die in DLLs (Dynamic Link Library) enthalten sind.

In diesem Kapitel werden einige nützliche API-Funktionen (APIs) vorgestellt, die entweder besondere Funktionalitäten bereitstellen oder bestimmte Aufgaben einfacher und schneller erledigen und sich ohne großen Aufwand in VBA integrieren lassen.



In der Datei *Bsp04\_1.doc* auf der Buch-CD finden Sie im Ordner *\Beispiele\Kap04* alle Beispiele zu den einzelnen Abschnitten in einzelnen Modulen, z.T. mit zusätzlichen Anwendungsbeispielen. Zusätzlich enthält dieser Ordner die Datei *Bsp04-1.INI* für die Beispiele im Abschnitt »Zugriff auf INI-Dateien« in diesem Kapitel.

## Aufbau der API-Funktionen

Jede API-Funktion besitzt den prinzipiell gleichen Aufbau, der sich hauptsächlich aus dem Namen der Funktion und der Bibliothek, in der sich die Funktion befindet, zusammensetzt. Zusätzlich können einzelne notwendige Argumente oder auch ganze Argumentlisten für den Aufruf angegeben werden. Die grundlegende Syntax einer API-Funktion sieht wie folgt aus (die einzelnen Teile werden in Tabelle 4.1 erläutert):

```
[Public|Private] Declare Sub|Function Name Lib "LibName" [Alias "AliasName"] [(arglist)]
[As Type]
```

Tabelle 4.1 Aufbau der API-Funktion

Parameter	Beschreibung
[Public Private]	Legt optional den Zugriffstyp fest: Öffentlich in einem Modul deklariert kann auf diese Funktion auch von anderen Modulen aus zugegriffen werden. Privat deklariert kann nur innerhalb des Moduls auf die Funktion zugegriffen werden. Die Standardeinstellung ist <b>Public</b> .
Declare	Notwendiges Schlüsselwort, mit dem der Zugriff auf eine API- oder DLL-Funktion definiert wird.
Sub Function	Legt den Typ der API-Funktion fest. Wenn die API-Funktion einen Rückgabewert liefert, sollte sie als <b>Function</b> deklariert werden; andernfalls kann sie als <b>Sub</b> deklariert werden.
Name	Der Zugriffsname der Funktion.

Tabelle 4.1 Aufbau der API-Funktion (Fortsetzung)

Parameter	Beschreibung
Lib	Notwendiges Schlüsselwort für den nachfolgenden Namen der Bibliothek.
"LibName"	Name der Bibliothek, in der die Funktion enthalten ist. Wird kein Pfad angegeben sucht Windows selbst nach der DLL.
Alias	Optionales Schlüsselwort, das ausdrückt, dass der angegebene Zugriffsname nicht gleichzeitig der (exportierte) Name der Funktion in der Bibliothek ist.
"AliasName"	Wird das Schlüsselwort <b>Alias</b> angegeben, stellt es den Namen der Funktion in der Bibliothek dar. Die korrekte Groß- und Kleinschreibung des Namens ist wichtig.
[([arglist])]	Weitere optionale Argumente für den Funktionsaufruf.
[As Type]	Legt den Typ der Funktion fest. Diese Angabe ist für den Rückgabewert der Funktion wichtig; normalerweise liefert eine Funktion einen Wert vom Typ <b>Long</b> zurück.

## Kombinieren und Abschließen von Pfaden

Eine häufige Fragestellung taucht bei der Angabe oder beim Kombinieren von Pfadangaben auf: Schließt ein Pfad mit einem Backslash (»\«) ab oder nicht?

So ist z.B. zum Öffnen von Dateien oder beim Überprüfen, ob Verzeichnisse vorhanden sind, die korrekte (unterschiedliche) Syntax wichtig: Beispielsweise verlangt die `Dir`-Funktion **keinen** abschließenden Backslash, wenn ein Verzeichnis überprüft werden soll, während ein Backslash bei der Dateinamenssuche im Verzeichnis **notwendig** ist.

So liefert in Listing 4.1 der erste Aufruf der `Dir`-Funktion den existierenden Ordernamen zurück, während mit Backslash der erste Dateieintrag im Ordner zurückgeliefert wird (sofern der Ordner existiert)

Listing 4.1 Unterschiedliche Rückgabewerte der `Dir`-Funktion

```
Debug.Print Dir("C:\temp",vbDirectory )
> temp
Debug.Print Dir("C:\temp\",vbDirectory )
> .
```

Ein weiteres Problem tritt bei der Kombination von Pfad und Dateinamen bzw. bei zwei Ordernamen auf: Es muss sichergestellt werden, dass der Gesamtpfad korrekt gesetzte Backslash beinhaltet. So sollten normalerweise beide Pfadteile auf beginnende bzw. endende Backslash überprüft werden, bevor sie kombiniert werden können, da andernfalls Fehler bezüglich unbekannter oder falscher Pfade auftreten können.

Listing 4.2 Fehlerhafte Auswertung bei fehlendem oder doppeltem Backslash

```
Debug.Print "C:\temp" & "Test.doc"
> C:\tempTest.doc
Debug.Print "C:\temp\" & "\Test.doc"
> C:\temp\Test.doc
```

Diese Problematik ließe sich zwar durch eine Abfrage des letzten Zeichens prüfen und für zukünftige Verwendung in Form einer Funktion schreiben, aber warum nicht auf vorhandene Wege zurückgreifen. So steht unter Windows ein Satz von API-Funktionen zur Verfügung, die sich um die korrekte Kombination und Erstellung von Pfaden kümmern.

## Abschließen eines Pfades

Mit Hilfe der API-Funktion `PathAddBackslash` wird automatisch ein Backslash an einen Pfad gesetzt, sofern noch keiner vorhanden ist. Die Deklaration lautet wie folgt (die Erklärung befindet sich in Tabelle 4.2):

```
Declare Function PathAddBackslash Lib "shlwapi.dll" Alias "PathAddBackslashA" _
    (ByVal pszPath As String) As Long
```

**Tabelle 4.2** Parameter der API-Funktion *PathAddBackslash*

Parameter	Bedeutung	Ein-/Ausgabe
pszPath	Variable mit zu prüfendem Pfad	Ein/Aus

Diese Funktion erwartet als Ein- und Ausgabe-Parameter eine Variable vom Typ `String`. Da der erweiterte Pfad in derselben Variablen wie der Eingangsparameter ausgegeben wird, können Sie den Pfad nicht direkt angeben, sondern müssen diesen über eine Variablen (Buffer) festlegen. Dieser Buffer muss groß genug sein, um den Rückgabewert aufnehmen zu können. Ist der Buffer zu klein, wird das Ergebnis an der Buffergrenze abgeschnitten.

Um den Buffer anzulegen, wird normalerweise der Pfad um eine Anzahl von Null-Zeichen (`vbNullChar`) erweitert. Die Erweiterung um diese Zeichen hat den Vorteil, dass diese Funktion die Position des ersten Null-Zeichens, die die Zeichenkette abschließt, zurückliefert. Das Listing 4.3 veranschaulicht dieses Prinzip.

**Listing 4.3** Abschließen des Pfades mit einem Backslash mittels *PathAddBackslash*

```
Sub subAddBackslash()
    Dim strPath As String, strTemp As String
    strPath = "C:\Temp" & String(254, vbNullChar)
    PathAddBackslash strPath
    MsgBox strPath, vbInformation, "Rückgabepfad"
End Sub
```

**Abbildg. 4.1** Ausgabe des abgeschlossenen Pfades





# Kombinieren von Pfaden

Um zwei Pfade miteinander zu kombinieren, können Sie die API-Funktion `PathCombine` verwenden. Die Deklaration lautet wie folgt (die Erklärung befindet sich in Tabelle 4.3):

```
Declare Function PathCombine Lib "shlwapi.dll" Alias "PathCombineA" _
    (ByVal szDest As String, _
    ByVal lpszDir As String, _
    ByVal lpszFile As String) As Long
```

Tabelle 4.3 Parameter der API-Funktion *PathCombine*

Parameter	Bedeutung	Ein-/Ausgabe
szDest	Variable mit dem kombinierten Pfad	Aus
lpszDir	Variable mit dem Basispfad	Ein
lpszFile	Variable mit dem anzuhängenden Pfad	Ein

Diese Funktion erwartet neben den beiden Pfadangaben `lpszDir` und `lpszFile`, die kombiniert werden sollen, eine Variable `szDest`, in die das Ergebnis ausgegeben wird.

Die Buffer-Variable `szDest` muss dabei wieder so groß angelegt werden, dass der kombinierte Pfad auf jeden Fall hineinpasst, wie Listing 4.4 veranschaulicht.

Listing 4.4 Kombinieren zweier Pfade mittels *PathCombine*

```
Sub subPathCombine()
    Dim strPath1 As String, strPath2 As String
    Dim strPathCombine As String
    strPath1 = "C:\Temp\"
    strPath2 = "Test\"
    strPathCombine = String(1024, vbNullChar)
    PathCombine strPathCombine, strPath1, strPath2
    MsgBox strPathCombine, vbInformation, "Rückgabepfad"
End Sub
```

Abbildg. 4.2 Ausgabe des zusammengesetzten Pfades



## WICHTIG

Wichtig dabei ist, dass der zweite Pfad relativ angegeben werden muss; d.h. er darf keine Laufwerksbuchstaben besitzen. Andernfalls wird die erste Pfadangabe ignoriert.

## Dateinamen an Pfad anhängen

Eine weitere nützliche API-Funktion ist PathAppend. Mit dieser Funktion können Sie einen Dateinamen an einen Pfad anhängen. Die Deklaration lautet wie folgt (die Erklärung befindet sich in Tabelle 4.4):

```
Declare Function PathAppend Lib "shlwapi.dll" Alias "PathAppendA" _
    (ByVal pszPath As String, ByVal pMore As String) As Long
```

Tabelle 4.4 Parameter der API-Funktion *PathAppend*

Parameter	Bedeutung	Ein-/Ausgabe
pszPath	Variable mit dem Basispfad	Ein/Aus
pMore	Variable mit dem Dateinamen	Ein

Diese Funktion erwartet neben dem Pfad pszPath den Dateinamen pMore; der kombinierte Pfad mit der Datei wird dabei in der Variablen pszPath wieder ausgegeben. Die Buffer-Variable pszPath muss dabei wieder so groß angelegt werden, dass der kombinierte Pfad auf jeden Fall hineinpasst, wie Listing 4.5 veranschaulicht.

Listing 4.5 Anhängen eines Dateinamens an einen Pfad mittels *PathAppend*

```
Sub subPathAppend()
    Dim strPath As String
    Dim strFile As String
    strPath = "C:\Temp\" & String(254, vbNullChar)
    strFile = "Test.doc"
    PathAppend strPath, strFile
    MsgBox strPath, vbInformation, "Rückgabepfad"
End Sub
```

Abbildg. 4.3 Ausgabe des angehängten Dateinamens



## Dateierweiterung an Datei anhängen

Mit der API-Funktion PathAddExtension können Sie an einen Dateinamen, den Sie z.B. aus anderen Angaben zusammengesetzt haben, mit einer Dateierweiterung versehen, sofern der Dateiname noch keine besitzt. Die Deklaration lautet:

```
Declare Function PathAddExtension Lib "shlwapi.dll" Alias "PathAddExtensionA" ( _
    ByVal pszPath As String, _
    ByVal pszExt As String ) As Long
```

Diese Funktion erwartet neben dem Dateinamen `pszPath` die Erweiterung `pszExt`; der Dateiname mit Erweiterung wird dabei wieder in der Variable `pszPath` ausgegeben. Die Buffer-Variablen `pszPath` muss dabei wieder so groß angelegt werden, dass der vollständige Dateiname auf jeden Fall hineinpasst, wie Listing 4.6 veranschaulicht.

**Listing 4.6** Anhängen einer Erweiterung an einen Dateinamen

```
Sub subPathAddExtension()
    Dim strFile As String
    Dim strExt As String
    strFile = "C:\Temp\Test" & String(254, vbNullChar)
    strExt = ".doc"
    PathAddExtension strFile, strExt
    MsgBox strFile, vbInformation, "Rückgabepfad"
End Sub
```

**Abbildg. 4.4** Ausgabe der angehängten Dateierweiterung



Die Beispiele zu diesem Abschnitt finden Sie in der Beispieldatei `\Beispiele\Kap04\Bsp04_1.doc` im Modul »modPathAPIs«.

## Datei über die Dateierendung ausführen

Wenn Sie eine Datei oder ein Programm starten möchten, klicken Sie im Explorer normalerweise doppelt auf den Dateinamen und Windows versucht, die Datei mit dem zugewiesenen Programm zu starten bzw. startet das Programm. Dieses Verhalten können Sie mit der API-Funktion `ShellExecute` auch über eigene Makros erreichen. Diese Funktion ermittelt bei einer Datei aus der Registry das verknüpfte zugehörige Programm und startet das Programm mit der angegebenen Datei. Diese etwas längere Deklaration befindet sich in Listing 4.7, die Erläuterung der Parameter ist in Tabelle 4.5 enthalten.

**Listing 4.7** Deklaration der API-Funktion *ShellExecute*

```
Declare Function ShellExecute Lib "shell32.dll" Alias "ShellExecuteA" ( _
    ByVal hwnd As Long, _
    ByVal lpOperation As String, _
    ByVal lpFile As String, _
    ByVal lpParameters As String, _
    ByVal lpDirectory As String, _
    ByVal ShowTypeEnum As Long _
) As Long
Private Enum ShowTypeEnum
    SW_HIDE = 0
```

**Listing 4.7** Deklaration der API-Funktion *ShellExecute* (Fortsetzung)

```

SW_SHOWNORMAL = 1
SW_SHOWMINIMIZED = 2
SW_SHOWMAXIMIZED = 3
SW_SHOWNOACTIVATE = 4
SW_SHOW = 5
SW_MINIMIZE = 6
SW_SHOWMINNOACTIVE = 7
SW_SHOWNA = 8
SW_RESTORE = 9
SW_SHOWDEFAULT = 10
SW_FORCEMINIMIZE = 11
End Enum

```

**Tabelle 4.5** Parameter der API-Funktion *ShellExecute*

Parameter	Bedeutung	Ein-/Ausgabe
hwnd	Zugriffsnummer (handle) des Fensters, an das alle Meldungen gesendet werden.	Ein
lpOperation	Variable mit der auszuführenden Aktion	Ein
lpFile	Variable mit dem Dateipfad bzw. Programmpfad	Ein
lpParameters	Variable mit Parametern, die bei einem Programm dem Programmaufruf mitgegeben werden. Bei einem Dateiaufruf sollte dieser Wert <b>Null</b> sein.	Ein
lpDirectory	Variable mit einem Standardverzeichnis; kann durch einen Leerstring ersetzt werden.	Ein
ShowTypeEnum	Ein Wert aus der <b>ShowTypeEnum</b> -Auflistung, der die Anzeige des Programmfensters steuert.	Ein

Als Parameter müssen Sie neben dem Dateipfad die Aktion angegeben, die auf die Datei angewendet werden soll. Dieses wird in den meisten Fällen **show** (anzeigen) sein. Sie können eine Datei aber auch über die Aktion **print** ausdrucken. Wenn Sie nur einen Ordnerpfad angeben, können Sie über die Aktion **explore** den Ordnerinhalt im Explorer anzeigen lassen.

Bei einer ausführbaren Datei (.exe) können Sie optional Parameter über die Variable **lpParameters** an den Programmaufruf weitergeben. Das Beispiel in Listing 4.8 zeigt die Verwendung von Parametern. Als Programm wird der Befehlsinterpreter **cmd** verwendet, was der Eingabeaufforderung entspricht. Diesem wird als Parameter der Befehl zur Anzeige des Benutzernamens und das Umlenken der Ausgabe in eine Datei angegeben. Anschließend wird die Ausgabedatei über den gleichen Befehl angezeigt.

**Listing 4.8** Beispiel zur Verwendung von Programmparametern

```

Sub subShellExecute5()
    Dim strTemp As String
    strTemp = "C:\Test.txt"
    fktShellExecute Environ$("COMSPEC"), "/C set Username > C:\Test.txt"
    fktShellExecute strTemp, "open"
End Sub

```

Der Parameter `lpDirectory` zur Angabe eines Standardverzeichnis wird normalerweise in VBA nicht benötigt.

Die Option `ShowTypeEnum` für das verknüpfte Programm bzw. das auszuführende Programm steuert die Anzeige des Programms, ob das Programmfenster z.B. maximiert, minimiert oder normal angezeigt werden soll.

Als Rückgabewert liefert die Funktion entweder die Zugriffsnummer des gestarteten Programms oder DDE-Servers oder im Fehlerfall die Fehlernummer. Über die Zuordnung zu einer Fehlerbeschreibung erhält man dann Rückschluss über die Ursache des Fehlers:

```
2& = ERROR_FILE_NOT_FOUND
```

**HINWEIS**

Die Zugriffsnummer `hwnd` wird in VBA normalerweise nicht benötigt und kann durch die Angabe `&00` ersetzt werden.

Dieses liegt auch daran, dass die aus VBA aufgerufenen Benutzerformulare (UserForm) und Hinweisdialoge (MsgBox) keine Zugriffsnummer besitzen bzw. keine Zugriffsnummer zurückliefern.

Listing 4.9 bis Listing 4.10 zeigen verschiedene Möglichkeiten zur Anwendung dieses API. Wenn kein E-Mail-Programm als Standard-E-Mail-Programm festgelegt ist oder die verwendete Datei vom System gesperrt ist, wird eine entsprechende Fehlermeldung ausgegeben.

**Listing 4.9** Öffnen einer Datei mittels *ShellExecute*

```
Sub subShellExecute()  
    ' Öffnet die Datei c_FilePath  
    Const c_FilePath = "C:\MVPBuch\Kap04\Bsp04-1.INI"  
    fktShellExecute c_FilePath, "open"  
End Sub
```

**Listing 4.10** Versenden einer Mail mittels *ShellExecute*

```
Sub subShellExecute3()  
    ' Versendet eine E-Mail über das Standard-E-Mail-Programm  
    fktShellExecute "mailto:test@chf-online.de", , , SW_SHOWNORMAL  
End Sub
```

In Listing 4.11 wird das API in einer Funktion gekapselt und um eine Fehlerauswertung erweitert. Die optionalen Aufrufparameter legen Standardparameter fest, wenn keine Parameter angegeben werden. Aus Platzgründen wird die Deklaration und die Enumeration nicht aufgeführt, Sie finden diese aber im Beispiel auf der Buch-CD.

**Listing 4.11** Kapselung der API-Funktion *ShellExecute* in eine Funktion mit Fehlerausgabe

```
' Die Deklaration der Konstanten und Funktionen muss an erster Stelle im Modul stehen  
Private Const ERROR_FILE_NOT_FOUND = 2& ' Datei nicht gefunden  
Private Const ERROR_PATH_NOT_FOUND = 3& ' Pfad nicht gefunden  
Private Const ERROR_BAD_FORMAT = 11& ' Falsches Dateiformat  
Private Const SE_ERR_ACCESSDENIED = 5 ' Zugriff verweigert  
Private Const SE_ERR_ASSOCINCOMPLETE = 27 ' Dateityp ist nicht ausreichend assoziiert  
Private Const SE_ERR_DDEBUSY = 30 ' DDE konnte nicht gestartet werden
```

**Listing 4.11** Kapselung der API-Funktion *ShellExecute* in eine Funktion mit Fehlerausgabe (Fortsetzung)

```

Private Const SE_ERR_DDEFAIL = 29 ' DDE ist gescheitert
Private Const SE_ERR_DDETIMEOUT = 28 ' DDE-Zeitlimit wurde erreicht
Private Const SE_ERR_DLLNOTFOUND = 32 ' eine benötigte DLL wurde nicht gefunden
Private Const SE_ERR_FNF = 2 ' Datei wurde nicht gefunden
Private Const SE_ERR_NOASSOC = 31 ' Dateityp ist nicht assoziiert
Private Const SE_ERR_OOM = 8 ' Nicht genügend Speicher verfügbar
Private Const SE_ERR_PNF = 3 ' Pfad wurde nicht gefunden
Private Const SE_ERR_SHARE = 26 ' Datei konnte nicht geöffnet werden, _
    da sie bereits verwendet wird

Function fktShellExecute(sFile, _
    Optional ByVal lpParameter As String = vbNullString, _
    Optional ByVal lpOperation As String = "open", _
    Optional ByVal lpDirectory As String = vbNullString, _
    Optional ByVal ShowTypeEnum As Integer = SW_SHOWNORMAL)
Dim lngRet As Long
Dim strMSG As String
lngRet = ShellExecute(&00, lpOperation, sFile, lpParameter, lpDirectory, ShowTypeEnum)
Select Case lngRet
Case ERROR_FILE_NOT_FOUND
    strMSG = "Die angegebene Datei wurde nicht gefunden."
Case ERROR_PATH_NOT_FOUND
    strMSG = "Der angegebene Pfad wurde nicht gefunden."
Case ERROR_BAD_FORMAT
    strMSG = "Die .EXE-Datei ist ungültig " & _
        "(keine Win32-Anwendung oder Fehler in der Dateistruktur)."
Case SE_ERR_ACCESSDENIED
    strMSG = "Der Zugriff auf die angegebene Datei wurde vom Betriebssystem verweigert."
Case SE_ERR_ASSOCINCOMPLETE
    strMSG = "Die Dateinamen-Zuordnung ist unvollständig oder ungültig."
Case SE_ERR_DDEBUSY
    strMSG = "Der DDE-Vorgang konnte nicht beendet werden," & _
        "da andere DDE-Vorgänge bearbeitet wurden."
Case SE_ERR_DDEFAIL
    strMSG = "Der DDE-Vorgang schlug fehl."
Case SE_ERR_DDETIMEOUT
    strMSG = "Der DDE-Vorgang konnte wegen einer Zeitüberschreitung nicht beendet werden."
Case SE_ERR_DLLNOTFOUND
    strMSG = "Die angegebene Dynamic-Link Library (DLL) wurde nicht gefunden."
Case SE_ERR_FNF
    strMSG = "Die angegebene Datei wurde nicht gefunden."
Case SE_ERR_NOASSOC
    strMSG = "Es ist kein Programm der angegebenen Dateiendung zugeordnet."
Case SE_ERR_OOM
    strMSG = "Nicht genügend Speicher zum Beenden des Vorgangs verfügbar."
Case SE_ERR_PNF
    strMSG = "Der angegebene Pfad wurde nicht gefunden."
Case SE_ERR_SHARE
    strMSG = "Die angegebene Datei konnte nicht geöffnet werden, " & _
        "da sie bereits verwendet wird."
End Select
If strMSG <> "" Then MsgBox strMSG, vbCritical, "ShellExecute-Error"
End Function

```



Die Beispiele zu diesem Abschnitt finden Sie in der Beispieldatei `\Beispiele\Kap04\Bsp04_1.doc` im Modul »modShellExecuteAPI«.

## Zugriff auf Registry-Einträge

### WICHTIG

Der Zugriff auf die Registrierungsdateien (Registry) von Windows ist mit nicht unerheblichen Gefahren verbunden, da dieses das Herzstück des Betriebssystems darstellt. Bei falschem Zugriff und Ändern an den Einträgen besteht die Gefahr, dass Windows anschließend nicht mehr korrekt funktioniert und evtl. sogar komplett neu installiert werden muss.

Daher ist es unbedingt empfehlenswert, vor dem Zugriff auf die Registry eine Sicherung dieser Dateien (z.B. mit Hilfe des Sicherungs-Assistenten die Systemstatusdateien) oder der gesamten Systempartition (z.B. über einen Wiederherstellungspunkt) anzulegen!

## Ermitteln von Registry-Einträgen und Werten

Wie mit VBA und dem Word-Objektmodell Einstellungen in der Registry einzeln geschrieben und gelesen werden können, steht in Kapitel 12 beschrieben. Diese Funktionalität bietet jedoch keine Möglichkeit, sich eine Übersicht der Einträge zu machen.

Mit der API-Funktion `RegEnumKeyEx` können Sie für einen Registry-Eintrag alle vorhandenen Unter-einträge (SubKeys) ermitteln.

Dazu muss zuerst der Registry-Eintrag, dessen Untereinträge ermittelt werden sollen, für den Zugriff geöffnet werden: Dieses erfolgt mit der API-Funktion `RegOpenKey`.

Die Funktion `RegOpenKey` erwartet als Parameter den Hexadezimalwert des Registry-Hauptzweiges, den zu öffnenden Eintrag; zurückgegeben wird eine Variable, die eine Zugriffsnummer auf den geöffneten Zweig beinhaltet.

Die Tabelle 4.6 listet die Hauptzweige der Registry auf.

**Tabelle 4.6** Hexadezimalwerte der Hauptzweige

Hauptzweig	Hexadezimalwert
<code>HKEY_CLASSES_ROOT</code>	<code>&amp;H0000000</code>
<code>HKEY_CURRENT_CONFIG</code>	<code>&amp;H80000005</code>
<code>HKEY_CURRENT_USER</code>	<code>&amp;H80000001</code>
<code>HKEY_LOCAL_MACHINE</code>	<code>&amp;H80000002</code>
<code>HKEY_USERS</code>	<code>&amp;H80000003</code>

Wichtig beim Zugriff auf Registry-Einträge ist, dass Sie anschließend alle geöffneten Einträge mit der API-Funktion `RegCloseKey` wieder schließen. Die Deklarationen dieser Funktionen sind wie folgt; die Parameter werden in Tabelle 4.7 vorgestellt.

```
Private Declare Function RegOpenKey Lib "advapi32.dll" Alias "RegOpenKeyA" ( _
    ByVal hKey As Long, _
    ByVal lpSubKey As String, _
    phkResult As Long ) As Long
Private Declare Function RegCloseKey Lib "advapi32.dll" _
    (ByVal hKey As Long) As Long
```

**Tabelle 4.7** Parameter der API-Funktionen *RegOpenKey* und *RegCloseKey*

Parameter	Bedeutung	Ein-/Ausgabe
hKey	Zugriffsnummer des Hauptzweigs	Ein
lpSubKey	Name des Registry-Eintrags	Ein
phkResult	Variable mit Zugriffsnummer zum Eintrag	Aus

Zum einfachen Auslesen der Einträge werden nicht alle Parameter des APIs *RegEnumKeyEx* benötigt, auch sind nicht alle Parameter beschrieben.

Wichtig ist vor allem die Angabe der Zugriffsnummer des auszulesenden Eintrags sowie eine Buffer-Variable zur Aufnahme des ermittelten Namens eines Untereintrags. Die Parameter der folgenden Deklaration sind in Tabelle 4.8 aufgelistet.

```
Private Declare Function RegEnumKeyEx Lib "advapi32.dll" Alias "RegEnumKeyExA" ( _
    ByVal hKey As Long, _
    ByVal dwIndex As Long, _
    ByVal lpName As String, _
    lpcbName As Long, _
    ByVal lpReserved As Long, _
    ByVal lpClass As String, _
    lpcbClass As Long, _
    lpftLastWriteTime As Any ) As Long
```

**Tabelle 4.8** Parameter der API-Funktion *RegEnumKeyEx*

Parameter	Bedeutung	Ein-/Ausgabe
hKey	Zugriffsnummer des Registry-Eintrags	Ein
dwIndex	Index des zu ermittelnden Untereintrags	Ein
lpName	Buffer zur Aufnahme des Untereintragsnamens	Ein
lpcbName	Länge des Buffereintrags	Ein
lpReserved	Muss NULL sein	Ein
lpClass	Nicht benötigt; Null-String	Ein
lpcbClass	Länge von <i>lpClass</i>	Ein
lpftLastWriteTime	Variable mit dem Zeitpunkt des letzten Zugriffs	Ein

Wenn kein Fehler aufgetreten ist, liefert die Funktion als Rückgabewert `ERROR_SUCCESS = 0`, andernfalls einen von Null verschiedenen Error-Code.



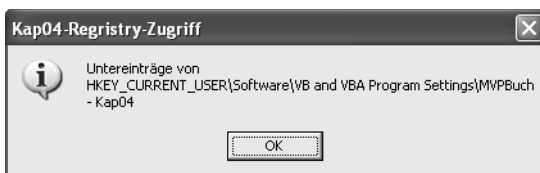
Das Listing 4.13 ermittelt im Registry-Eintrag der VB- und VBA-Einstellungen (siehe Kapitel 12) die Untereinträge im Eintrag *MVPBuch*. Wenn Sie das Makro *SaveVBSetting* in der Beispieldatei *Bsp04\_1.doc* von der Buch-CD (im Ordner *\Beispiele\Kap04*) ausgeführt haben, liefert die nachstehende Prozedur den Namen des Untereintrags zurück: *Kap04*.

**Listing 4.12** Ermitteln aller Untereinträge eines Registry-Eintrags

```
Sub EnumRegKeys()
    Const c_Title = "Kap04-Regristry-Zugriff"
    Dim hKey As Long, lngCnt As Long
    Dim strName As String, strData As String
    Dim lngRet As Long, lngRetData As Long
    Dim strMSG As String
    Const BUFFER_SIZE As Long = 255
    lngRet = BUFFER_SIZE
    Const c_Base = "Software\VB and VBA Program Settings\MVPBuch"
    Const HKEY_CURRENT_USER = &H80000001
    Const ERROR_NO_MORE_ITEMS = 259&
    ' Öffnen des Eintrags
    ' HKEY_CURRENT_USER\Software\VB and VBA Program Settings\MVPBuch
    If RegOpenKey(HKEY_CURRENT_USER, c_Base, hKey) = 0 Then
        ' Buffer zur Aufnahme der Untereinträge erstellen
        strName = String(BUFFER_SIZE, vbNullChar)
        ' Durchlaufen der Untereinträge bis keine weiteren Einträge
        While RegEnumKeyEx(hKey, lngCnt, strName, lngRet, ByVal 0&, _
            vbNullString, ByVal 0&, ByVal 0&) <> ERROR_NO_MORE_ITEMS
            ' sammeln der Untereinträge
            strMSG = strMSG & "- " & Left$(strName, lngRet) & vbCrLf
            ' Zähler für den nächsten Eintrag setzen
            lngCnt = lngCnt + 1
            ' Buffer neu anlegen
            strName = String(BUFFER_SIZE, vbNullChar)
            lngRet = BUFFER_SIZE
        Wend
        ' Schließen des Registry-Eintrags
        RegCloseKey hKey
    Else
        MsgBox "Beim Aufruf der Funktion 'RegOpenKey' ist ein Fehler aufgetreten."
    End If
    strMSG = "Untereinträge von " & vbCrLf & "HKEY_CURRENT_USER\" & c_Base & vbCrLf & strMSG
    MsgBox strMSG, vbInformation, c_Title
End Sub
```

Als Ergebnis erhalten Sie eine Übersicht der gefundenen Untereinträge.

**Abbildg. 4.5** Ausgabe der gefundenen Untereinträge eines Registry-Eintrags



Zur Ermittlung aller Schlüssel und ihrer Werte eines Eintrags können Sie die API-Funktion `RegEnumValue` verwenden, dessen Deklaration folgt. Die Beschreibung der Parameter befindet sich in Tabelle 4.9

```
Private Declare Function RegEnumValue Lib "advapi32.dll" Alias "RegEnumValueA" ( _
    ByVal hKey As Long, _
    ByVal dwIndex As Long, _
    ByVal lpValueName As String, _
    lpcbValueName As Long, _
    ByVal lpReserved As Long, _
    lpType As Long, _
    lpData As Any, _
    lpcbData As Long ) As Long
```

**Tabelle 4.9** Parameter der API-Funktion *RegEnumValue*

Parameter	Bedeutung	Ein-/Ausgabe
<code>hKey</code>	Zugriffsnummer des Registry-Eintrags	Ein
<code>dwIndex</code>	Index des zu ermittelnden Untereintrags	Ein
<code>lpValueName</code>	Buffer zur Aufnahme des Schlüsselnamens	Ein
<code>lpcbValueName</code>	Länge des Buffereintrags	Ein
<code>lpReserved</code>	Muss NULL sein	Ein
<code>lpType</code>	Schlüsseltyp <code>REG_SZ</code> , <code>REG_MULTI_SZ</code> , <code>REG_EXPAND_SZ</code>	Ein
<code>lpData</code>	Buffer zur Aufnahme des Schlüsselwertes	Ein
<code>lpcbData</code>	Länge des Buffereintrags <code>lpData</code>	Ein

Wenn kein Fehler aufgetreten ist, liefert die Funktion als Rückgabewert `ERROR_SUCCESS = 0&`, andernfalls einen von Null verschiedenen Error-Code.

Zur Ermittlung aller Schlüssel und ihrer Werte wird die Prozedur aus Listing 4.12 dahingehend geändert, wie Listing 4.13 veranschaulicht, dass zuerst alle Untereinträge in einem Array gesammelt werden und anschließend für diese Array-Einträge die Schlüssel ermittelt werden. Dieses ist notwendig, da beide APIs Registry-Einträge (Eintrag und Untereintrag) per `RegOpenKey` geöffnet werden müssen, was aber nicht direkt nacheinander möglich ist; es muss immer erst der Eintrag wieder geschlossen werden, bevor ein anderer Eintrag geöffnet werden kann.

**Listing 4.13** Ermitteln aller Schlüssel und Werte aller Untereinträge zu einem Registry-Eintrag

```
Sub EnumRegKeyValues()
    ' Prozedur zur Ermittlung aller Untereinträge eines Eintrags
    Const c_Title = "Kap04-Registry-Zugriff"
    Dim hKey As Long, lngCnt As Long
    Dim strName() As String, strData As String
    Dim lngRet As Long, lngRetData As Long
    Dim strMSG As String
    Const BUFFER_SIZE As Long = 255
    lngRet = BUFFER_SIZE
```

Listing 4.13 Ermitteln aller Schlüssel und Werte aller Untereinträge zu einem Registry-Eintrag (Fortsetzung)

```

Const c_Base = "Software\VB and VBA Program Settings\MVPBuch"
Const HKEY_CURRENT_USER = &H80000001
Const ERROR_NO_MORE_ITEMS = 259&
' Öffnen des Eintrags
' HKEY_CURRENT_USER\Software\VB and VBA Program Settings\MVPBuch
If RegOpenKey(HKEY_CURRENT_USER, c_Base, hKey) = 0 Then
    ReDim strName(0)
    ' Buffer zur Aufnahme der Untereinträge erstellen
    strName(UBound(strName)) = String(BUFFER_SIZE, vbNullChar)
    ' Durchlaufen der Untereinträge, bis keine weiteren Einträge
    Do While RegEnumKeyEx(hKey, lngCnt, strName(UBound(strName)), lngRet, ByVal 0&, _
        vbNullString, ByVal 0&, ByVal 0&) <> ERROR_NO_MORE_ITEMS
        ' Auf leere Untereinträge prüfen
        If Trim(Left(strName(UBound(strName)), lngRet)) <> "" Then
            ' sammeln der Untereinträge im Array
            strName(UBound(strName)) = Trim(Left(strName(UBound(strName)), lngRet))
        Else
            Exit Do
        End If
        ' Zähler für nächsten Eintrag setzen
        lngCnt = lngCnt + 1
        ' Buffer neu anlegen
        ReDim Preserve strName(UBound(strName()) + 1)
        strName(UBound(strName)) = String(BUFFER_SIZE, vbNullChar)
        lngRet = BUFFER_SIZE
    Loop
    'Schließen des Registry-Eintrags
    RegCloseKey hKey
Else
    MsgBox "Beim Aufruf der Funktion 'RegOpenKey' ist ein Fehler aufgetreten."
End If
strMSG = "Untereinträge von " & vbCrLf & "HKEY_CURRENT_USER\" & c_Base & vbCrLf & strMSG
Dim strTemp As String
For lngCnt = LBound(strName()) To UBound(strName()) - 1
    strTemp = strTemp & fktGetKeyValues(HKEY_CURRENT_USER, c_Base & "\" & strName(lngCnt),
hKey) & vbCrLf
Next lngCnt
MsgBox strMSG & strTemp, vbInformation, c_Title
End Sub

Function fktGetKeyValues(lngBase As Long, strRoot As String, lngKey As Long) As String
    ' Funktion zum Ermitteln aller Schlüssel/Werte eines Untereintrags
    Dim intCnt As Integer
    Dim strName As String, strData As String
    Dim lngRet As Long, lngRetData As Long
    Dim strTemp As String
    intCnt = 0
    Const BUFFER_SIZE As Long = 255
    ' Öffnen des Eintrags
    ' HKEY_CURRENT_USER\Software\VB and VBA Program Settings\MVPBuch\ + strName(lngCnt)
    If RegOpenKey(lngBase, strRoot, lngKey) = 0 Then
        ' Buffer zur Aufnahme der Schlüsselnamen und Werte erstellen
        strName = Space(BUFFER_SIZE)
        strData = Space(BUFFER_SIZE)
        lngRet = BUFFER_SIZE
        lngRetData = BUFFER_SIZE
        'Durchlaufen der Schlüssel

```

**Listing 4.13** Ermitteln aller Schlüssel und Werte aller Untereinträge zu einem Registry-Eintrag (*Fortsetzung*)

```

While RegEnumValue(IngKey, intCnt, strName, lngRet, 0, ByVal 0&, _
    ByVal strData, lngRetData) <> ERROR_NO_MORE_ITEMS
    'sammeln der Schlüsselnamen und Werte
    If lngRetData > 0 Then
        strTemp = strTemp & " - " & Left$(strName, lngRet) + "=" & _
            Left$(strData, lngRetData - 1) & vbCrLf
    End If
    ' Zähler und Buffer für nächsten Eintrag vorbereiten
    intCnt = intCnt + 1
    strName = Space(BUFFER_SIZE)
    strData = Space(BUFFER_SIZE)
    lngRet = BUFFER_SIZE
    lngRetData = BUFFER_SIZE
Wend
'Schließen des Registry-Eintrags
RegCloseKey lngKey
fktGetKeyValues = strTemp
End If
End Function

```

Als Ergebnis erhalten Sie neben den gefundenen Untereinträgen die vorhandenen Schlüssel und Werte.

**Abbildg. 4.6** Ausgabe der gefundenen Untereinträge mit ihren Schlüssel und Werten


Die Beispiele zu diesem Abschnitt finden Sie in der Beispieldatei `\Beispiele\Kap04\Bsp04_1.doc` im Modul »modRegAPIs«.

## Zugriff auf INI-Dateien

Eine INI-Datei ist im Prinzip eine normale Textdatei, in der Informationen abschnittsweise gespeichert sind. Die Informationen werden dabei nach Abschnitten (Sections) sortiert und beinhalten sowohl einen Schlüssel als auch den dem Schlüssel zugeordneten Wert, wie in Listing 4.14 dargestellt.

**Listing 4.14** Beispiel einer INI-Datei mit sprachspezifischen Informationen

```

[1031]
1=Ihr Nachname
2=Ihr Vorname
3=Straße

```

Listing 4.14 Beispiel einer INI-Datei mit sprachspezifischen Informationen (Fortsetzung)

```

4=0rt
5=Land
6=Bemerkung
[0407]
1=Your surname
2=Your given name
3=Street
4=City
5=Country
6=Remarks

```

Welche Daten und Informationen genau in einer INI-Datei gespeichert werden, ist nicht entscheidend. Wichtig ist der Aufbau in Abschnitte, die in eckigen Klammern angegeben werden müssen ([Section]), und die Schlüssel-Werte-Einträge (Key=Value). Sind die Daten in dieser Struktur hinterlegt, können Sie mit Hilfe eines Satzes von API-Funktionen gezielt auf ganze Abschnitte oder auch einzelne Schlüssel zugreifen.

**HINWEIS**

INI-Dateien sind hilfreich, um Konfigurationsdaten zu speichern. Mehr darüber erfahren Sie in Kapitel 12.

## Auslesen und Schreiben von Abschnitten

Mit Hilfe der beiden API-Funktionen `WritePrivateProfileSection` und `GetPrivateProfileSection` können Sie komplette Einträge schreiben und lesen. Nachfolgend finden Sie die Deklaration für `WritePrivateProfileSection`, die Einträge in eine INI-Datei schreibt (die Parameter sind in Tabelle 4.10 aufgelistet):

```

Public Declare Function WritePrivateProfileSection Lib "kernel32" _
    Alias "WritePrivateProfileSectionA" _
    (ByVal lpAppName As String, _
    ByVal lpString As String, _
    ByVal lpFileName As String) As Long

```

Tabelle 4.10 Parameter der API-Funktion `WritePrivateProfileSection`

Parameter	Bedeutung	Ein-/Ausgabe
lpAppName	Name des zu schreibenden Abschnittes	Ein
lpString	Variable mit den zu schreibenden Schlüsseln und Werten	Ein
lpFileName	Name mit Pfad der INI-Datei	Ein

Wenn kein Fehler aufgetreten ist, liefert die Funktion als Rückgabewert einen von Null verschiedenen Wert, andernfalls ist der Rückgabewert Null.

Das Beispiel in Listing 4.15 schreibt den String

```
"Eintrag1=TestEintrag1" & vbCrLf & "Eintrag2=TestEintrag2"
```

in die INI-Datei *Bsp04-1.INI*.

**Listing 4.15** Schlüssel und Werte in einen Abschnitt schreiben

```
Sub subWriteINISection()
    Const c_INIPath = "C:\MVPBuch\Kap04\Bsp04-1.INI"
    Const c_SecName As String = "Kap04"
    Dim strSection As String
    strSection = "Eintrag1=TestEintrag1" & vbCrLf & "Eintrag2=TestEintrag2"
    WritePrivateProfileSection c_SecName, strSection, c_INIPath
End Sub
```

**WICHTIG** Wenn ein angegebener Schlüssel in dem Abschnitt bereits vorhanden ist, wird der dazugehörige Wert ohne Nachfrage überschrieben!

Zum Auslesen aller Schlüssel und Werte eines Abschnittes können Sie die API-Funktion *GetPrivateProfileSection* verwenden, deren Deklaration wie folgt lautet:

```
Public Declare Function GetPrivateProfileSection Lib "kernel32" _
    Alias "GetPrivateProfileSectionA" _
    (ByVal lpAppName As String, _
    ByVal lpReturnedString As String, _
    ByVal nSize As Long, _
    ByVal lpFileName As String) As Long
```

Eine Auflistung der Parameter finden Sie in Tabelle 4.11.

**Tabelle 4.11** Parameter der API-Funktion *GetPrivateProfileSection*

Parameter	Bedeutung	Ein-/Ausgabe
lpAppName	Name des zu lesenden Abschnittes	Ein
lpReturnedString	Buffer zur Aufnahme der Schlüssel und Werte im angegebenen Abschnitt	Aus
nSize	Größe des Buffers	Ein
lpFileName	Name mit Pfad der INI-Datei	Ein

Die Funktion schreibt in die Buffervariable alle gefundenen Schlüssel mit ihren Einträgen. Gleichzeitig liefert sie als Rückgabewert die Anzahl der geschriebenen Bufferzeichen.

Um die zurückgelieferten Schlüssel lesbar darzustellen, wird die Buffervariable zuerst auf die Anzahl der geschriebenen Zeichen reduziert. Da die Schlüssel, nur von Null-Zeichen getrennt, hintereinander in den Buffer geschrieben werden, werden anschließend alle Null-Zeichen durch Zeilenwechsel ersetzt, wie Listing 4.16 veranschaulicht. Das Ergebnis ist in Abbildung 4.7 abgebildet.

**Listing 4.16** Auslesen aller Schlüssel und Werte aus dem Abschnitt »Kap04«

```
Sub subReadINISection()
    Const c_Read = "Abschnitt einlesen"
    Const c_INIPath = "C:\MVPBuch\Kap04\Bsp04-1.INI"
```

Listing 4.16 Auslesen aller Schlüssel und Werte aus dem Abschnitt »Kap04« (Fortsetzung)

```

Const c_SecName As String = "Kap04"
Dim lngSec As Long
Dim strSec As String
lngSec = 255
strSec = String(255, vbNullChar)
GetPrivateProfileSection c_SecName, strSec, lngSec, c_INIPath
strSec = c_SecName & ":" & vbCrLf & fktStripString(strSec)
MsgBox strSec, vbInformation, c_Read
End Sub
Function fktStripString(strInput As String, lngRet As Long) As String
    Dim strTemp As String
    strTemp = Left$(strInput, lngRet)
    strTemp = Replace(strTemp, vbNullChar, vbCrLf)
    fktStripString = strTemp
End

```

Mit dieser API-Funktion `GetPrivateProfileSection` erhalten Sie somit alle Schlüssel eines Abschnittes zurück.

Abbildg. 4.7 Ausgabe der im Abschnitt »Kap04« enthaltenen Schlüssel mit ihren Werten

**HINWEIS**

Wenn Sie jedoch gezielt auf einen einzelnen Schlüssel zugreifen und den Wert auslesen oder ändern möchten, stellt das Word-Objektmodell die Funktion `PrivateProfileString` zur Verfügung, die in Kapitel 12 ausführlich beschrieben ist.

## Ermitteln aller Abschnitte einer INI-Datei

Mit den genannten API-Funktionen können Sie auf bekannte Abschnitte und ihre Schlüssel und Werte zugreifen. Dazu muss aber zwingend der Abschnitt bekannt sein, auf den zugegriffen werden soll.

Sind diese nicht bekannt, müssen Sie entweder die INI-Datei zeilenweise auslesen und auf die eckigen Klammern als Abschnittskennung prüfen, oder Sie verwenden die API-Funktion `GetPrivateProfileSectionNames` zur Ermittlung aller Abschnittsnamen in der Datei. Nachfolgend finden Sie die Deklaration (die Parameter sind in Tabelle 4.12 aufgelistet):

```

Public Declare Function GetPrivateProfileSectionNames Lib "kernel32.dll" _
    Alias "GetPrivateProfileSectionNamesA" _
    (ByVal lpszReturnBuffer As String, _
    ByVal nSize As Long, _
    ByVal lpFileName As String) As Long

```

**Tabelle 4.12** Parameter der API-Funktion *GetPrivateProfileSectionNames*

Parameter	Bedeutung	Ein-/Ausgabe
lpzReturnBuffer	Buffer zur Aufnahme der gefundenen Abschnitte	Aus
nSize	Größe des Buffers	Ein
lpFileName	Name mit Pfad der INI-Datei	Ein

Die Funktion schreibt in die Buffervariable alle gefundenen Abschnittsnamen. Gleichzeitig liefert die Funktion als Rückgabewert die Anzahl der geschriebenen Bufferzeichen.

Das Beispiel in Listing 4.17 ermittelt erst die Anzahl und Namen der vorhandenen Abschnitte in der INI-Datei *Bsp04-1.INI*, um anschließend die Schlüssel und Werte in diesen Abschnitten auszulesen. Da die bisherigen Beispiele nur einen Abschnitt verwendeten, werden vorher mit dem Makro *subWrite2INISection* zwei Abschnitte angelegt und mit Schlüsseln gefüllt. Das Resultat ist in Abbildung 4.8 ersichtlich.

**Listing 4.17** Ermitteln und Auslesen aller Abschnitte in der Datei *Bsp04-1.INI*

```

Sub subWrite2INISection()
    Const c_INIPath = "C:\MVPBuch\Kap04\Bsp04-1.INI"
    Const c_SecName As String = "Kap04"
    Const c_SecName2 As String = "Kap05"
    Dim strSection As String
    strSection = "Eintrag1=TestEintrag1" & vbCrLf & "Eintrag2=TestEintrag2"
    ret = WritePrivateProfileSection(c_SecName, strSection, c_INIPath)
    strSection = "Eintrag3=TestEintrag3" & vbCrLf & "Eintrag4=TestEintrag4"
    ret = WritePrivateProfileSection(c_SecName2, strSection, c_INIPath)
End Sub

Sub subReadAllINISection()
    Const c_Read = "Abschnitte ermitteln"
    Const c_INIPath = "C:\MVPBuch\Kap04\Bsp04-1.INI"
    Dim lngSec As Long
    Dim strBuffer As String
    Dim strSec() As String, strSection As String
    Dim strAllSections As String
    Dim intSec As Integer
    lngSec = 1024
    strBuffer = String(lngSec, vbNullChar)
    ret = GetPrivateProfileSectionNames(strBuffer, lngSec, c_INIPath)
    strSec() = fktSplitString(strBuffer, ret)
    For intSec = LBound(strSec()) To UBound(strSec()) - 1
        strAllSections = strAllSections & strSec(intSec) & ":" & vbCrLf
        strSection = String(lngSec, vbNullChar)
        ret = GetPrivateProfileSection(strSec(intSec), strSection, lngSec, c_INIPath)
        strAllSections = strAllSections & fktStripString(strSection, ret) & vbCrLf
    Next intSec
    strAllSections = "Enthaltene Abschnitte: " & vbCrLf & vbCrLf & _
        strAllSections
    MsgBox strAllSections, vbInformation, c_Read
End Sub

Function fktSplitString(strInput As String, lngRet As Long) As Variant
    Dim strTemp As String

```



**Listing 4.17** Ermitteln und Auslesen aller Abschnitte in der Datei *Bsp04-1.INI* (Fortsetzung)

```
strTemp = Left$(strInput, lngRet)
fktSplitString = Split(strTemp, vbNullChar)
End Function
```

**Abbildg. 4.8** Ausgabe der gefundenen Abschnitte mit ihren Schlüsseln und Werten



Die Beispiele zu diesem Abschnitt finden Sie in der Beispieldatei *\Beispiele\Kap04\Bsp04\_1.doc* im Modul »modINIAPIs«.

## Name des angemeldeten Benutzers ermitteln

Manchmal kann es für bestimmte Aktionen in Makros notwendig sein, den am Rechner gerade angemeldeten Benutzernamen zu wissen. Z.B. um benutzerspezifische Daten anzuzeigen.

Dazu stehen Ihnen unter Windows 2000, Windows XP und Windows Server 2003 zwei Möglichkeiten zur Verfügung.

Die erste greift auf die Umgebungsvariablen des Systems zu und liest sie mittels der *Environ*-Funktion aus. Über die Umgebungsvariablen *Computername* und *Username* erhalten Sie den Namen des Computers und des aktiven Benutzers, wie Listing 4.18 veranschaulicht.

**Listing 4.18** Ausgabe des angemeldeten Benutzernamens mittels *Environ*-Funktion

```
Sub subGetUserNameEnviron()
    Dim strEnv As String
    strEnv = Environ("Computername") & "\" & Environ("Username")
    MsgBox strEnv, vbInformation, "Angemeldeter Benutzer(Umgebungsvariablen)"
End Sub
```

Die zweite Möglichkeit ermittelt diese Informationen mittels der API-Funktion *GetUserNameEx*, dessen Deklaration sich in Listing 4.19 befindet. Eine Erläuterung der Parameter befindet sich in Tabelle 4.13.

**ACHTUNG** Die API-Funktion `GetUserNameEx` funktioniert nur unter Windows 2000, Windows XP und Windows Server 2003. Unter Windows 9x/ME können Sie die API-Funktion `GetUserName` verwenden:

```
Private Declare Function GetUserName Lib "advapi32.dll" Alias "GetUserNameA" ( _
    ByVal lpBuffer As String, _
    nSize As Long ) As Long
```

Listing 4.19 Deklaration der API-Funktion `GetUserNameEx`

```
Private Enum EXTENDED_NAME_FORMAT
    NameUnknown = 0
    NameFullyQualifiedDN = 1
    NameSamCompatible = 2
    NameDisplay = 3
    NameUniqueId = 6
    NameCanonical = 7
    NameUserPrincipal = 8
    NameCanonicalEx = 9
    NameServicePrincipal = 10
End Enum
Private Declare Function GetUserNameEx Lib "secur32.dll" Alias "GetUserNameExA" ( _
    ByVal NameFormat As EXTENDED_NAME_FORMAT, _
    ByVal lpNameBuffer As String, _
    ByRef nSize As Long _
) As Long
```

Tabelle 4.13 Parameter der API-Funktion `GetUserNameEx`

Parameter	Bedeutung	Ein-/Ausgabe
NameFormat	Wert aus der Aufzählung, die das Format des Namens angibt. Dieser Wert kann nicht <b>NameUnknown</b> sein.	Ein
lpNameBuffer	Buffervariable, die den Benutzernamen aufnimmt.	Aus
nSize	Größe der Buffervariablen; liefert die Länge des Benutzernamens zurück.	Ein/Aus

Diese Funktion liefert in der Buffervariablen `lpNameBuffer` den Benutzernamen zurück. Gleichzeitig liefert `nSize` die Länge des Benutzernamens zurück. Das Format des Namens wird (abhängig von der verwendeten Windows-Umgebung) über den Wert `NameFormat` aus der `EXTENDED_NAME_FORMAT`-Auflistung festgelegt. Das Listing 4.20 veranschaulicht den Gebrauch der API-Funktion.

**HINWEIS** Auf einem Windows XP-Einzelplatz-System, der nicht an einem Domaincontroller angemeldet ist, liefert nur

```
NameSamCompatible = 2
```

den Benutzernamen zurück. Alle anderen Werte liefern keine Informationen zurück.

Listing 4.20 Ermitteln des Computer- und Benutzernamens mittels *GetUserNameEx*

```

Sub subGetUserNameWin2k()
    Dim strBuffer As String
    Dim lngBSize As Long
    Dim lngRet As Long
    strBuffer = String(255, vbNullChar)
    lngBSize = Len(strBuffer)
    lngRet = GetUserNameEx(NameSamCompatible, strBuffer, lngBSize)
    If lngRet > 0 Then
        MsgBox Left(strBuffer, lngBSize), vbInformation, "Angemeldeter Benutzer (API)"
    Else
        MsgBox "Es ist ein Fehler beim Ermitteln des Benutzernamens aufgetreten!", _
            vbCritical, "Angemeldeter Benutzer (API)"
    End If
End Sub

```



Die Beispiele zu diesem Abschnitt finden Sie in der Beispieldatei `\Beispiele\Kap04\Bsp04_1.doc` im Modul »modUserNameAPIs«.

## Verarbeitung für eine bestimmte Zeit unterbrechen

Normalerweise kann die Abarbeitung einer Prozedur nicht schnell genug erfolgen, vor allem, wenn umfangreiche Bearbeitungen oder Berechnungen durchzuführen sind. Aber ab und an wäre es praktisch, die Verarbeitung kurz zu unterbrechen, um z.B. dem Anwender geänderte Informationen lesbar darzustellen oder um beim Drucken zwischen den einzelnen Druckjobs eine Pause einzuschieben. Meistens wird diese Unterbrechung mit einer einfachen For...Next-Schleife realisiert. Diese Schleife besitzt aber den großen Nachteil, dass die Systemauslastung auf nahezu 100 % ansteigt, obwohl eigentlich nichts gemacht werden soll. Auch ist eine Schleife nicht genau, da das Durchlaufen der Schleife von der Systemgeschwindigkeit abhängt. Wenn Sie eine exakte und System entlastende Unterbrechung benötigen, können Sie dazu die API-Funktion *Sleep* verwenden. Diese Funktion unterbricht die aktuelle Verarbeitung für eine bestimmte Anzahl von Millisekunden. Die Deklaration lautet:

```
Private Declare Sub Sleep Lib "kernel32" (ByVal lngMilliseconds As Long)
```

Die Erläuterung der Parameter befindet sich in Tabelle 4.14 und das Listing 4.21 veranschaulicht den Gebrauch.

Tabelle 4.14 Parameter der API-Funktion *Sleep*

Parameter	Bedeutung	Ein-/Ausgabe
lngMilliseconds	Angabe der Zeit in Millisekunden, für die die Verarbeitung unterbrochen werden soll.	Ein

**Listing 4.21** Beispiel zur Anwendung der API-Funktion *Sleep*

```

Sub subSleep()
    Dim lngRet As Long
    lngRet = CInt(InputBox("Pause in Millisekunden:", "Sleep-API", 3000))
    Sleep lngRet
    MsgBox "Pause ist vorbei", vbInformation, "Sleep-API"
End Sub

```

**WICHTIG** Einen großen Nachteil hat diese API-Funktion:

Sie unterbricht die gesamte Anwendung, also in diesem Fall Word, für den angegebenen Zeitraum. In dieser Zeit wird z.B. auch die Bildschirmansicht von Word nicht aktualisiert.

Als Alternative zu dieser API-Funktion können Sie evtl. die Timer-Funktion von Visual Basic verwenden und in einer Do...Loop-Schleife die vergangene Zeit überprüfen.

**Listing 4.22** Verarbeitung mittels *Timer*-Funktion unterbrechen

```

Sub subSleepTimer()
    Dim tTime As Long
    Dim lngRet As Long
    lngRet = CInt(InputBox("Pause in Sekunden:", "Timer-Funktion", 3))
    tTime = Timer
    Do While Timer < tTime + lngRet
        DoEvents
    Loop
    MsgBox "Pause ist vorbei", vbInformation, "Timer-Funktion"
End Sub

```

Bei dieser Funktion steigt die Systemauslastung allerdings wieder auf nahezu 100 %.



Die Beispiele zu diesem Abschnitt finden Sie in der Beispieldatei `\Beispiele\Kap04\Bsp04_1.doc` im Modul »modSleepAPI«.

## Wave-Datei abspielen

Eine nette Erweiterung z.B. für Benutzerformulare ist die Untermalung von Ereignissen mit einem Sound. So können Sie eine falsche Benutzereingabe zusätzlich mit einem Fehler-Sound quittieren oder das Beenden einer Verarbeitung mit einem Hinweis-Sound. Dazu können Sie entweder eigene Wave-Dateien oder die standardmäßig installierten System-Sounds verwenden.

Zum Abspielen von Wave-Dateien können Sie die API-Funktion `sndPlaySound` in Ihre Anwendung einbinden. Im Folgenden sehen Sie die Deklaration, die Parameter sind in Tabelle 4.15 aufgelistet:

```

Public Declare Function sndPlaySound Lib "WinMM.dll" Alias "sndPlaySoundA" ( _
    ByVal lpszSoundName As String, _
    ByVal uFlags As Long ) As Long

```

Tabelle 4.15 Parameter der API-Funktion *sndPlaySound*

Parameter	Bedeutung	Ein-/Ausgabe
lpszSoundName	Name des System-Sounds oder Name mit Pfad zur eigenen Wave-Datei.	Ein
uFlags	Parameter zur Abspielsteuerung	Ein

Wenn der Sound erfolgreich abgespielt werden konnte, liefert die Funktion *True* zurück, und *False*, wenn ein Fehler aufgetreten ist.

Zur Abspielsteuerung stehen Ihnen die folgenden *uFlag*-Parameter zur Verfügung:

Tabelle 4.16 Bedeutung der Abspielsteuerparameter *uFlag*

uFlag Parameter	Bedeutung
SND_SYNC	Der Sound wird synchron abgespielt; d.h. erst wenn der Sound vollständig abgespielt wurde, läuft die Verarbeitung weiter.
SND_ASYNC	Der Sound wird asynchron abgespielt, d.h. die Verarbeitung wird parallel zum Abspielen fortgesetzt.
SND_NODEFAULT	Wenn die Sound-Datei nicht gefunden wird, wird <b>nicht</b> die konfigurierte Standard-Datei »Standardton Warnsignal« abgespielt.
SND_LOOP	Legt fest, dass die Datei endlos abgespielt wird, bis entweder eine andere Datei abgespielt oder die Funktion mit einem Leerstring aufgerufen wird.
SND_NOSTOP	Wenn bereits eine Sound-Datei asynchron abgespielt wird, wird das Abspielen einer neuen Datei sofort beendet, ohne die andere Datei zu beenden.
SND_PURGE	Beendet das Abspielen einer Datei, die mittels <b>SND_LOOP</b> in einer Endlosschleife abgespielt wird.
SND_NOWAIT	Spielt die Datei sofort ab, auch wenn bereits eine andere Datei abgespielt wird.

Das Beispiel in Listing 4.22 öffnet ein Auswahldialogfeld im Windows-Ordner *Media* und setzt zur Auswahl den Dateityp auf *Wave*-Dateien. Wenn eine Datei ausgewählt wurde, wird diese anschließend asynchron abgespielt.

Listing 4.23 Abspielen einer beliebigen Wave-Datei unter Word 2002 (XP) und 2003

```
Public Function procPlayWAVKlang(ByVal cKlangname As String, ByVal LFlags As Long) _
    As Boolean
    'Prozedur zum Abspielen des gewünschten Klanges, sofern der Klang
    'im Setup überhaupt aktiviert wurde.
    Dim boolRet As Boolean
    boolRet = sndPlaySound(cKlangname, LFlags)
    procPlayWAVKlang = boolRet
End Function

Sub subPlaySoundWord2003()
    ' Wav-Datei unter Word 2003 abspielen
    Dim bRet As Boolean
    Dim lRet As Single
```

**Listing 4.23** Abspielen einer beliebigen Wave-Datei unter Word 2002 (XP) und 2003 (Fortsetzung)

```

Dim strName As String
' AuswahlDialog im Media-Verzeichnis
With Application.FileDialog(msoFileDialogFilePicker)
    .Filters.Add "Wave-Dateien", "*.wav", 1
    .InitialFileName = Environ("SystemRoot") & "\\Media\\"
    .AllowMultiSelect = False
    lRet = .Show
    ' Wenn keine Datei ausgewählt wurde, abbrechen
    If lRet <> "-1" Then
        MsgBox "Keine Datei ausgewählt.", vbCritical, "Datei abspielen"
        Exit Sub
    End If
    ' Ersten Eintrag auswählen
    strName = .SelectedItems(1)
End With
' Sound abspielen
bRet = procPlayWAVKlang(strName, SND_ASYNC)
If bRet = False Then
    MsgBox "Konnte Datei nicht abspielen.", vbCritical, "Datei abspielen"
End If
End Sub

```

Unter Word 2000 existiert die in diesem Beispiel verwendete Eigenschaft `FileDialog` noch nicht, so dass Ihnen dieses Dialogfeld mit eigenem Dateifilter nicht zur Verfügung steht. Der Code für diese Word-Version befindet sich in Listing 4.24.

**HINWEIS**

Das `FileDialog`-Objekt wird in Kapitel 14 näher vorgestellt.

**Listing 4.24** Abspielen einer Wave-Datei unter Word 2000

```

Public Function procPlayWAVKlang(ByVal cKlangname As String, ByVal LFlags As Long) As Boolean
'Prozedur zum Abspielen des gewünschten Klanges, sofern der Klang
'im Setup überhaupt aktiviert wurde.
    Dim boolRet As Boolean
    boolRet = sndPlaySound(cKlangname, LFlags)
    procPlayWAVKlang = boolRet
End Function

Sub subPlaySoundWord2000()
' Wav-Datei unter Word 2000 abspielen
    Dim bRet As Boolean
    bRet = procPlayWAVKlang("tada.wav", SND_ASYNC)
    If bRet = False Then
        MsgBox "Konnte Datei nicht abspielen.", vbCritical, "Datei abspielen"
    End If
End Sub

```

Wenn Sie eine längere Wave-Datei oder eine Datei in einer Endlosschleife abspielen, haben Sie zwei Möglichkeiten, um das Abspielen zu beenden (siehe auch das Listing 4.25):

- Durch Angabe des `NULL`-Zeichens als Dateinamen.
- Durch Verwendung des `SND_PURGE`-Parameters.

Listing 4.25 Möglichkeit einer Endlosschleife oder um das Abspielen einer Datei zu beenden

```

Sub subStopSound()
' Endlosschleife/Abspielen mittels NULL beenden
  procPlayWAVKlang vbNull, SND_ASYNC
End Sub

Sub subStopSound2()
' Endlosschleife/Abspielen mittels SND_PURGE beenden
  procPlayWAVKlang "tada.wav", SND_PURGE
End Sub

```

**TIPP**

Wenn Sie zwei oder mehrere Abspielparameter verwenden möchten, werden diese mittels dem OR-Operator verknüpft:

```
SND_ASYNC Or SND_LOOP
```



Die Beispiele zu diesem Abschnitt finden Sie in der Beispieldatei `\Beispiele\Kap04\Bsp04_1.doc` im Modul »modsndPlaySoundAPI«.

## Tastenstatus abfragen

In manchen Anwendungen kann man durch zusätzliches Drücken einer bestimmten Taste das Programmverhalten ändern. So ruft in einem Benutzerdialog die Taste **F1** die Onlinehilfe des Dialogfensters auf, während über **⇧ + F1** die Feldhilfe, sofern verfügbar, zum aktiven Feld aufgerufen wird.

Über die API-Funktion `GetKeyState` können Sie den Status (gedrückt/nicht gedrückt) einer Taste abfragen und darauf reagieren, indem Sie z.B. ein Benutzerformular beim Aufruf mit gedrückter **⇧**-Taste mit bestimmten Vorgabewerten füllen, anderenfalls ohne. Nachfolgend finden Sie die Deklaration dieser API-Funktion, die Parameter sind in Tabelle 4.17 aufgelistet.

```

Declare Function GetKeyState Lib "user32" ( _
  ByVal nVirtKey As Long ) As Integer

```

Tabelle 4.17 Parameter der API-Funktion `GetKeyState`

Parameter	Bedeutung	Ein-/Ausgabe
nVirtKey	Konstante der zu prüfenden virtuellen Taste.	Ein

Als Rückgabewert liefert diese Funktion den Status der geprüften Taste, der angibt, ob die Taste gedrückt, nicht gedrückt oder die Funktion der Taste umgeschaltet ist.

Für die Sonder- und Steuertasten werden Hexadezimalwerte (virtuelle Tastenwerte) verwendet, für die Zahlen und Buchstaben müssen die ASCII-Werte verwendet werden. Die Tabelle 4.18 liefert eine Übersicht über die gängigen Tastenkonstanten.

**Tabelle 4.18** Virtuelle Tastenkonstanten mit Bedeutung und Hexadezimalwert











Virtuelle Tastenkonstante	Taste	Hexadezimalwert
VK_LBUTTON	Linke Maustaste	&H1
VK_RBUTTON	Rechte Maustaste	&H2
VK_CANCEL	Strg + Untbr	&H3
VK_MBUTTON	Mittlere Maustaste	&H4
VK_XBUTTON1	X1-Maustaste	&H5
VK_XBUTTON2	X2-Maustaste	&H6
VK_BACK	←	&H8
VK_TAB	↩	&H9
VK_CLEAR	Entf	&HC
VK_RETURN	↵	&HD
VK_SHIFT	⇧	&H10
VK_CONTROL	Strg	&H11
VK_MENU	Alt	&H12
VK_PAUSE	Pause	&H13
VK_CAPITAL	CapsLock	&H14
VK_ESCAPE	Esc	&H1B
VK_SPACE	Leertaste	&H20
VK_PRIOR	Bild ↑	&H21
VK_NEXT	Bild ↓	&H22
VK_END	Ende	&H23
VK_HOME	Pos1	&H24
VK_LEFT	←	&H25
VK_UP	↑	&H26
VK_RIGHT	→	&H27
VK_DOWN	↓	&H28
VK_SELECT	Markieren	&H29
VK_PRINT	Druck	&H2A
VK_EXECUTE	Ausführen	&H2B
VK_SNAPSHOT	Bildschirm drucken	&H2C
VK_INSERT	Einfg	&H2D
VK_DELETE	Entf	&H2E
VK_HELP	Hilfe	&H2F



Tabelle 4.18 Virtuelle Tastenkonstanten mit Bedeutung und Hexadezimalwert (Fortsetzung)

Virtuelle Tastenkonstante	Taste	Hexadezimalwert
VK_NUMPAD0	Nummernfeld	&H60
VK_NUMPAD1	Nummernfeld	&H61
VK_NUMPAD2	Nummernfeld	&H62
VK_NUMPAD3	Nummernfeld	&H63
VK_NUMPAD4	Nummernfeld	&H64
VK_NUMPAD5	Nummernfeld	&H65
VK_NUMPAD6	Nummernfeld	&H66
VK_NUMPAD7	Nummernfeld	&H67
VK_NUMPAD8	Nummernfeld	&H68
VK_NUMPAD9	Nummernfeld	&H69
VK_MULTIPLY	Nummernfeld	&H6A
VK_ADD	Nummernfeld	&H6B
VK_SEPARATOR	Nummernfeld	&H6C
VK_SUBTRACT	Nummernfeld	&H6D
VK_DECIMAL	Nummernfeld	&H6E
VK_DIVIDE	Nummernfeld	&H6F
VK_F1		&H70
VK_F2		&H71
VK_F3		&H72
VK_F4		&H73
VK_F5		&H74
VK_F6		&H75
VK_F7		&H76
VK_F8		&H77
VK_F9		&H78
VK_F10		&H79
VK_F11		&H7A
VK_F12		&H7B
VK_F13		&H7C
VK_F14		&H7D
VK_F15		&H7E
VK_F16		&H7F

**Tabelle 4.18** Virtuelle Tastenkonstanten mit Bedeutung und Hexadezimalwert (Fortsetzung)

Virtuelle Tastenkonstante	Taste	Hexadezimalwert
VK_F17		&H80
VK_F18		&H81
VK_F19		&H82
VK_F20		&H83
VK_F21		&H84
VK_F22		&H85
VK_F23		&H86
VK_F24		&H87
VK_NUMLOCK		&H90
VK_SCROLL		&H91

Zum Prüfen des Tastenstatus wird ein bitweiser Vergleich zwischen dem Rückgabewert, der als Integer-Wert eine 16-Bit-Zahl darstellt, und dem Hexadezimalwert &HF000& (Dezimal: 61440, Binär: 1111000000000000) durchgeführt, wie in Listing 4.26 ersichtlich. Ist das Ergebnis wieder der Hexadezimalwert &HF000&, dann ist die Taste gedrückt.

**Listing 4.26** Prüfen des Tastenstatus mittels *GetKeyState*

```
Public Function funcIsTasteGedrückt(iTastenCode As Integer) As Boolean
    ' Überprüft bitweise, ob eine Taste gedrückt ist
    If (GetKeyState(iTastenCode) And &HF000&) = &HF000& Then
        funcIsTasteGedrückt = True
    End If
End Function
```

**HINWEIS** Beim bitweisen Vergleich, oder auch Binärvergleich, werden zwei Zahlen in Binärschreibweise miteinander verglichen. Binärstellen, die bei beiden Zahlen identisch sind, werden ins Ergebnis übernommen, für unterschiedliche Zahlen ist das Ergebnis 0.

Liefert die API-Funktion als Rückgabewert »-128«, ist das Ergebnis des bitweisen Vergleichs mit &HF000& wieder dieser Hexadezimalwert:

&HFF80	-128	1111111110000000
&HF000	61440	1111000000000000
-----		
&HF000	61440	1111000000000000


Hingegen liefert ein Rückgabewert von 0 im Vergleich wieder 0:

&H0000	0	0000000000000000
&HF000	61440	1111000000000000
-----		
&H0000	0	0000000000000000

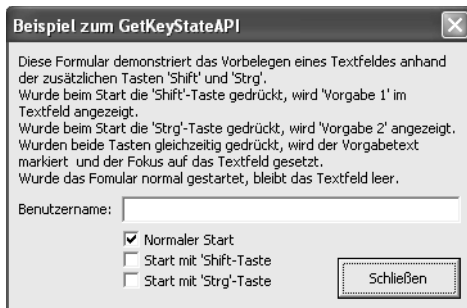
Als Beispiel wird der Aufruf der Userform `frmGetKeyStateAPI` in der Beispieldatei *Bsp04\_1.doc* verwendet. In Listing 4.27 wird die Abfrage evtl. gedrückter Tasten in einer `Select Case`-Anweisung geprüft, wobei mit Hilfe der Funktion `funcIsTasteGedrückt` aus Listing 4.26 der Tastenstatus ermittelt wird. Im ersten Case-Abschnitt wird geprüft, ob beide Tasten, und in den beiden nächsten Abschnitten, ob die beiden einzeln gedrückt wurden.



**Listing 4.27** Anweisung zum Überprüfen, ob die Tasten  und/oder  beim Start gedrückt wurden


```
Select Case True
    Case funcIsTasteGedrückt(VK_SHIFT) And funcIsTasteGedrückt(VK_CONTROL)
        txtName.Value = "<Bitte Namen eintragen>"
        txtName.SelStart = 0
        txtName.SelLength = Len(txtName.Text)
        fktSetCheckbox VK_SHIFT, VK_CONTROL
        txtName.SetFocus
    Case funcIsTasteGedrückt(VK_SHIFT)
        txtName.Value = "Vorgabe 1"
        fktSetCheckbox VK_SHIFT
    Case funcIsTasteGedrückt(VK_CONTROL)
        txtName.Value = "Vorgabe 2"
        fktSetCheckbox VK_CONTROL
    Case Else
        fktSetCheckbox &00
End Select
```

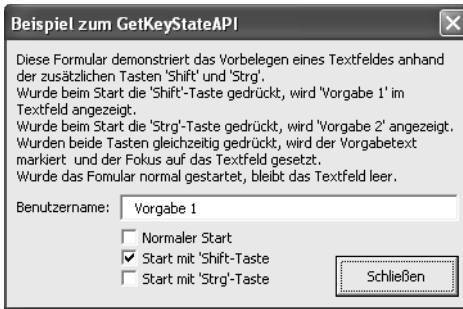
Öffnen Sie im Visual Basic-Editor mit einem Doppelklick die Userform `frmGetKeyStateAPI`. Starten Sie anschließend die Userform entweder über die Taste  oder über den Menüpunkt *Ausführen/ Sub/Userform ausführen*. Es wird die in Abbildung 4.9 gezeigte Userform mit einem leeren Feld *Benutzername* angezeigt.

**Abbildg. 4.9** Normaler Start des Benutzerformulars ohne zusätzlich gedrückte Taste





Wenn Sie beim Start zusätzlich die -Taste gedrückt halten, wird es mit dem Benutzernamen »Vorgabe 1« angezeigt. Wenn Sie die -Taste gedrückt halten, wird der Benutzername »Vorgabe 2« angezeigt.

Abbildg. 4.10 Start des Benutzerformulars mit gedrückter  -Taste



Wenn Sie die beiden Tasten gleichzeitig beim Aufruf des Benutzerformulars gedrückt halten, wird im Feld »Benutzername« die Vorgabe »<Bitte Namen eintragen>« vorgegeben, der Eintrag gleichzeitig markiert und der Eingabefokus auf dieses Feld gesetzt. Somit kann direkt in das Feld ein Eintrag vorgenommen werden.

Abbildg. 4.11 Start des Benutzerformulars mit gedrückter  - und  -Taste



Bei jedem Aufruf des Benutzerformulars wird zusätzlich über die Kontrollkästchen angezeigt, wie das Benutzerformular aufgerufen wurde.



Die Beispiele zu diesem Abschnitt finden Sie in der Beispieldatei `\Beispiele\Kap04\Bsp04_1.doc` im Modul »modGetKeyStateAPI«.

## Zusammenfassung

In diesem Kapitel wurde Ihnen gezeigt, welche Möglichkeiten Ihnen die Verwendung von APIs auch in Word bietet, auch wenn nicht alle Windows-APIs in VBA genutzt werden können.

- Nach dem prinzipiellen Aufbau der API-Funktionen (Seite 138) wurden Ihnen API-Funktionen vorgestellt, mit denen Sie relativ einfach mehrere Pfade (Seite 139 ff.) und Dateinamen (Seite 142) syntaktisch korrekt miteinander kombinieren und sogar Dateien mit Erweiterungen ergänzen können, sofern diese noch keine Erweiterung besitzen (Seite 142).

- Des Weiteren wurde gezeigt, wie Sie eine Datei über ihre Dateierweiterung aufrufen können (Seite 143).
- Es wurden Funktionen vorgestellt, um gezielt auf Einträge in der Windows-Registry zuzugreifen und mit denen Sie Registry-Einträge und -Werte ermitteln können (Seite 147 ff.).
- Ein weiterer Schwerpunkt dieses Kapitels war das Auslesen von und Schreiben in INI-Dateien mittels API-Funktionen (Seite 152 ff.).
- Weitere Beispiele haben Ihnen gezeigt, wie Sie den Namen des gerade angemeldeten Benutzers ermitteln (Seite 157) und die Verarbeitung von Makros für eine bestimmte Zeit unterbrechen können (Seite 159).
- Anhand eines weiteren Beispiels wurde eine API-Funktion vorgestellt, um eine beliebige Wave-Datei abzuspielen (Seite 160 ff.).
- Abgeschlossen wurde dieser Exkurs in die Welt der API-Funktionen mit einem Beispiel, das Ihnen zeigt, wie Sie abhängig von bestimmten Tastenstatus bestimmte Aktionen steuern können (Seite 163).

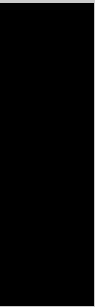


## Teil B

# Das Objektmodell von Word

### In diesem Teil:

<b>Kapitel 5</b>	Überblick über die Arbeit mit Objekten	173
<b>Kapitel 6</b>	Das Objektmodell von Word	187
<b>Kapitel 7</b>	Ereignisse in Word	431





## Kapitel 5

# Überblick über die Arbeit mit Objekten

### In diesem Kapitel:

Arbeiten mit Objekten	174
Auflistung von Objekten	179
Auflistung bearbeiten	180
Selection – das verpönte Objekt	183
Auto-Makros als Pseudo-Ereignisse	184
Zusammenfassung	185

Im ersten Teil dieses Buches haben Sie in den jeweiligen Kapiteln – von der Entwicklungsumgebung bis hin zu den Grundlagen von VBA – einen ersten Eindruck vom Programmieren in Word erhalten. Im zweiten Teil versuchen wir Ihnen nun das Objektmodell von Word sowie die Dokument- bzw. Programmereignisse näher zu bringen.

Von den einzelnen Microsoft Office-Anwendungen weist Word das umfangreichste Objektmodell auf. Und es liegt wohl an der Komplexität dieses Objektmodells, dass jeder, der zum ersten Mal Word automatisieren muss, mit Startschwierigkeiten zu kämpfen hat. Schon vielfach wurden wir mit der Aussage konfrontiert: »Ich weiß nicht, welche Objekte in Word vorhanden sind.«

Die passende Antwort liegt auf der Hand. Doch hilft sie dem Fragesteller selten weiter, sondern verwirrt ihn zusätzlich. Denn jeder Buchstabe, jedes Wort, jeder Absatz, jeder Abschnitt, die Kopf- und Fußzeilen, die Fußnoten, jede Tabelle und jede Grafik, alle Dokumente und deren Dokumentvorlagen sind Objekte – ja, sogar Word selbst ist ein Objekt. Eigentlich ganz logisch.

Vom Anfänger hingegen werden oft ganz andere Objekte gesucht: eine bestimmte Zeile oder eine bestimmte Seite. Doch diese Objekte gibt es nicht. Bei Word handelt es sich um eine Textverarbeitung. Und deren oberstes Gebot ist der so genannte Fließtext. Die gesuchten Objekte können gar nicht existieren, weil eine bestimmte Zeile nur eine Momentaufnahme darstellt. Das Ändern eines einzigen Wortes wirkt sich auf den ganzen Text aus, so dass die gewünschten Objekte bereits neu definiert werden müssten. Die Nicht-Existenz dieser Objekte ist also ebenfalls ganz logisch.

## Arbeiten mit Objekten

Die nachstehenden Abschnitte vermitteln zunächst einen kurzen Einblick in die Arbeit mit Objekten. Das eigentliche Objektmodell wird anschließend in Kapitel 6 ausführlich erörtert.

### Das gesuchte Objekt aufspüren

Word selber stellt uns einige Hilfsmittel zur Verfügung, um ein gesuchtes Objekt schnell aufzuspüren. Kombiniert mit einer gewissen Portion Neugierde haben sie bisher noch jedes gut versteckte Objekt ans Tageslicht befördert und dürfen hier keinesfalls unerwähnt bleiben.



Online-  
Hilfe

- Die *Online-Hilfe* wurde bereits in Kapitel 2 im Abschnitt »Die VBA-Hilfe – eine versteckte Schatzkammer« vorgestellt. Aus unserer Sicht ist der wichtigste Hinweis auf jeder Hilfeseite die Verknüpfung *Siehe auch*. Denn hier geht es zu den verwandten Themen, und wenn die Neugierde erst einmal geweckt wurde, dann werden viele interessante Sachen entdeckt.

Abbildg. 5.1

Lesen Sie die verwandten Themen, um ein Objekt aufzuspüren und besser kennen zu lernen



Objekt-  
katalogIntelli-  
SenseMakro-  
rekorder

- Ein weiteres Hilfsmittel ist der *Objektkatalog*, der einen umfassenden Einblick in jede Software-Bibliothek freigibt und dessen Möglichkeiten schon in Kapitel 2 im Abschnitt »Wo alle Fäden zusammenlaufen: Der Objektkatalog« vorgestellt wurden.
- Die *IntelliSense*-Funktion unterstützt den Programmierer während des Erstellens von Programmzeilen und legt gleichzeitig die »Unterobjekte« zu einem Objekt offen. Denn in vielen Fällen entspricht die Eigenschaft eines Objekts einem weiteren Objekt.
- Das wohl wichtigste Hilfsmittel aber ist der *Makrorekorder*. Er wurde bereits in Kapitel 1 im Abschnitt »Den Makrorekorder einsetzen« vorgestellt. Im aktuellen Fall jedoch wird er nicht zum Generieren der benötigten Programmzeilen, sondern zum Aufspüren der entsprechenden Objekte verwendet.

Die Kombination dieser vier Hilfsmittel, der persönlichen Neugierde sowie der wachsenden Erfahrung im Umgang mit dem Objektmodell bringt jedes Objekt zum Vorschein.

## Objekte als Variablen

Den Datentyp *Object* haben Sie in Kapitel 3 im Abschnitt »Variablen« bereits kennen gelernt. Doch statt eine allgemeine Variable vom Typ *Object* anzulegen, ist es sinnvoller, eine Variable des benötigten Typs zu deklarieren:

```
Dim rng_1 As Object      'nicht optimal
Dim rng_2 As Range      'besser
Dim rng_3 As Word.Range 'optimal
```

### PROFITIPP

Wir Autoren empfehlen Ihnen, bei der Deklaration von Objektvariablen nebst der benötigten Klasse (beispielsweise *Range*) grundsätzlich die zugehörige Bibliothek (beispielsweise *Word*) zu nennen:

```
Dim rng As Word.Range
```

Die Gründe sind nahe liegend und Sie können sich bei konsequenter Anwendung viel Ärger ersparen:

- Die Bezeichnung einer Klasse muss nicht eindeutig sein. In Abhängigkeit der in das VBA-Projekt eingebundenen Bibliotheken kann die Klasse mehrmals vorkommen. Die Klasse *Range* beispielsweise ist sowohl in der Bibliothek von *Word* als auch in jener von *Microsoft Excel* definiert.
- Der Compiler verfügt über eine klare Anweisung, die angeforderte Klasse muss nicht innerhalb aller eingebundenen Bibliotheken zusammengesucht werden.
- Die Unterstützung durch *IntelliSense* funktioniert für alle Objekte richtig, denn ohne Definition der Bibliothek werden nur die Eigenschaften und Methoden des ersten Treffers in der Liste der eingebundenen Bibliotheken aufgelistet.

Neben dem zusätzlichen Aufwand, den die Deklaration dieser Objektvariablen mit sich bringt, steht der daraus resultierende Nutzen im Vordergrund:

- Die Programmzeilen werden kürzer, da das entsprechende Objekt direkt bearbeitet wird. Sie bleiben übersichtlicher und für den Programmierer besser lesbar. Gleichzeitig erfolgt eine indirekte Dokumentation des Programms:

```
tbl.Rows.Add
```

anstelle von

```
ActiveDocument.Tables(1).Rows.Add
```

- Der Bezug zum Dokument wird eindeutig, denn zusammen mit der Zuweisung des Objekts an die Variable wird diese festgelegt und bleibt bis zu ihrer Freigabe bestehen.
- Nur bei deklarierten Objektvariablen erfolgt eine Unterstützung durch IntelliSense. Bei allgemeinen Variablen vom Typ Object steht sie nicht zur Verfügung, da das eigentliche Objekt erst während der Zuweisung, also zur Laufzeit des Programms, interpretiert wird.

**HINWEIS**

Bevor eine Objektvariable einer bestimmten Klasse deklariert werden kann, muss dem VBA-Projekt ein so genannter *Verweis* auf die entsprechende Bibliothek hinzugefügt werden. Mehr zum Thema »Verweise auf externe Bibliotheken« erfahren Sie in Kapitel 8.

## Zuweisen von Objektvariablen

Dim xyz  
Set xyz =

Damit eine Objektvariable in einem Programmteil verwendet werden kann, muss sie zuerst deklariert und in einem zweiten Schritt mittels der Set-Anweisung einem gültigen Objekt zugewiesen werden:

```
Dim doc As Word.Document  
Set doc = ActiveDocument
```

Die Zuweisung des Objekts an die Objektvariable setzt nur einen Verweis auf die Speicheradresse des entsprechenden Objekts.

In der Syntax der Dim-Anweisung kann das optionale Schlüsselwort New angegeben werden. Die Verwendung von New weist den Compiler an, ein neues Objekt implizit zu erstellen. Es wird eine neue Instanz auf das Objekt erstellt. (Damit die nachstehende Zeile erfolgreich getestet werden kann, muss vorab ein Verweis auf die »Microsoft Excel 11.0 Object Library« gesetzt werden.)

```
Dim xls As New Excel.Application
```

Es zeugt jedoch von einem unsauberen Programmierstil, wenn innerhalb der gleichen Programmzeile das Objekt deklariert und gleichzeitig eine neue Instanz desselben angelegt wird. Denn in diesem Fall werden Deklarations- und Programmzeilen vermischt und der Anweisung kommt eine doppelte Bedeutung zu. In Listing 5.1 wurden die Deklaration der Objektvariablen und das Anlegen einer neuen Instanz von Excel bewusst getrennt.

Innerhalb der ersten Zeilen der Prozedur wird die benötigte Objektvariable angelegt:

```
Dim xls As Excel.Application
```

In einem getrennten zweiten Schritt wird die neue Instanz auf das gewünschte Objekt, in diesem Fall auf Excel, erzeugt:

```
Set xls = CreateObject("excel.application")
```

**Listing 5.1** Das Erzeugen der Objektvariablen und das Anlegen der Instanz wurden getrennt

```
Sub ExcelObjekt()  
'Verweis auf »Microsoft Excel 11.0 Object Library« nötig  
Dim xls As Excel.Application  
Dim xwb As Excel.Workbook  
Dim xws As Excel.Worksheet  
  
Set xls = CreateObject("excel.application")  
xls.Visible = True  
  
Set xwb = xls.Workbooks.Add  
Set xws = xwb.Sheets.Add  
  
xws.Range("A1").Formula = "Hallo Welt"  
xws.PrintOut  
xwb.Close SaveChanges:=False  
  
xls.Quit  
  
Set xws = Nothing  
Set xwb = Nothing  
Set xls = Nothing  
End Sub
```

**HINWEIS** Im .NET Framework sieht man oft, dass ein Objekt in der gleichen Codezeile deklariert und ihm eine neue Instanz einer Klasse zugewiesen wird. Hier ist das möglich, weil die Instanz gleichzeitig ins Leben gerufen werden kann, anders als im klassischen VBA. Kann die neue Instanz aus irgendeinem Grund nicht angelegt werden, besteht auch hier das gleiche Problem und das Einsetzen des Schlüsselworts New muss in einer separaten Zeile erfolgen:

```
Dim xls as Excel.Application = New Excel.Application
```

In C#:

```
Excel.Application xls = new Excel.Application();
```

### Standardeigenschaft von Objekten

Jedes Objekt aus dem Objektmodell von Word verfügt über eine Standardeigenschaft. Als Beispiel dient das Range-Objekt, für das standardmäßig die Text-Eigenschaft definiert ist.

Beim Erfassen der Programmzeilen kann somit die Zuweisung eines Textes an einen festgelegten Range mittels

```
rng = "Hallo Welt"
```

oder



```
rng.Text = "Hallo Welt"
```

erfolgen. Aus Sicht der Programmlogik besteht kein Unterschied zwischen den beiden Anweisungen. Egal welche der beide Zeilen zum Einsatz kommt, im Dokument wird an der gewünschten Stelle der Gruß »Hallo Welt« eingefügt.

Wir empfehlen jedoch, der Verlockung, die Standardeigenschaft zu verwenden, zu widerstehen und stattdessen jede Eigenschaft voll zu qualifizieren:

- Die Programmzeile ist nicht selbst erklärend, da nur geübte Entwickler die Standardeigenschaften aller Objekte kennen. Die Dokumentation der Programmlogik wird umso wichtiger.
- Es ist nicht gewährleistet, dass in allen und auch zukünftigen Programmversionen von Word die Standardeigenschaft eines Objekts immer die gleiche ist.
- Das Portieren des Programms nach C# wird bedeutend aufwändiger, denn in C# muss jede Eigenschaft voll qualifiziert werden; C# kennt keine Standardeigenschaften.

## Freigeben von Objektvariablen

Set xyz =  
Nothing

Objektvariablen müssen grundsätzlich nach deren Verwendung wieder freigegeben werden. Mit dem Schlüsselwort `Nothing` wird veranlasst, dass die Verbindung einer Objektvariablen zum entsprechenden Objekt aufgehoben wird. Die Freigabe erfolgt explizit durch die Verwendung der `Set`-Anweisung oder implizit, nachdem die letzte Objektvariable den Gültigkeitsbereich verlässt und somit auf `Nothing` gesetzt wird:

```
Set xls = Nothing
```

### HINWEIS

Wir empfehlen, Objektvariablen, die auf andere Applikationen verweisen, grundsätzlich durch eine explizite Verwendung der `Set`-Anweisung freizugeben. Sie ersparen sich so die mühsame Suche nach Programmfehlern, die nur sporadisch und in Abhängigkeit Ihrer eigenen Applikation auftreten werden. Dies vor allem dann, wenn die Anwendung mehr als einmal in einer Sitzung ausgeführt wird.

Bei der Freigabe von Objektvariablen ist, wie in Listing 5.1 ersichtlich, darauf zu achten, dass diese grundsätzlich in der umgekehrten Reihenfolge, wie diese erzeugt wurden, freigegeben werden. Nur so ist sichergestellt, dass wirklich alle Verbindungen auf die betreffenden Speicheradressen entfernt und die reservierten System- und Speicherressourcen ebenfalls freigegeben werden:

```
Set xws = Nothing
Set xwb = Nothing
Set xls = Nothing
```

# Auflistung von Objekten

Beim Erforschen des Objektmodells von Word stößt der Anwender schnell auf eine Besonderheit. Von den meisten aufgeführten Objekten sind im Objektkatalog innerhalb der gleichen Bibliothek zwei Einträge vorhanden (beispielsweise Document und Documents, Paragraph und Paragraphs usw.).

Bei jenen Objekten, deren Bezeichnung im Singular steht (beispielsweise Document, Paragraph usw.) handelt es sich um ein einzelnes spezifisches Objekt.

Bei den anderen Objekten, deren Bezeichnung im Plural steht (beispielsweise Documents, Paragraphs usw.), handelt es sich um eine so genannte Auflistung eines spezifischen Objektes.

Ein einzelnes Objekt kann entweder vorhanden oder eben nicht mehr vorhanden sein (beispielsweise das Dokument wurde geschlossen, der Absatz wurde entfernt). Die Auflistung eines Objekts hingegen ist dynamisch und beinhaltet immer die Werte des aktuellen Zustands (beispielsweise die Auflistung aller geöffneten Dokumente, die Auflistung aller Absätze im Dokument).

## Zugreifen auf ein spezifisches Objekt

Um auf ein spezifisches Objekt aus einer Auflistung heraus zuzugreifen, kann dieses mit seinem Index oder über seinen Namen, sofern ein solcher vorhanden ist, angesprochen und einer Objektvariablen zugewiesen werden:

```
Set doc = Application.Documents(1)           'via Index
Set doc = Application.Documents("Document1.doc") 'via Name
```

## Index eines Objekts ermitteln

Ein Objekt kann also über seinen Index direkt angesprochen werden. Es gibt jedoch Situationen, in welchen der Index eines markierten Objekts ermittelt werden muss, um sicherzustellen, ob sich die Einfügemarke an oder innerhalb der gewünschten Position befindet.

Der Index eines Objekts ist jedoch meistens nur innerhalb der Auflistung vorhanden und ist auch keine Eigenschaft des einzelnen Objekts. Da die einzelnen Objekte innerhalb der Auflistung ab Dokumentanfang durchnummeriert werden, ist es leicht, die Nummer des markierten Objekts zu bestimmen.

In Listing 5.2 wird der Index der markierten Tabelle bestimmt. Dies, nachdem in einem ersten Programmschritt festgestellt wurde, dass sich die Einfügemarke innerhalb einer Tabelle befindet. Dieses Programmbeispiel kann analog auf andere Objekte angewendet werden.

**Listing 5.2** Indexnummer der aktuellen Tabelle ermitteln

```
Sub IndexDerTabelleErmitteln()
    Dim rng As Word.Range
    Dim lngIndex As Long

    If Selection.Information(wdWithInTable) Then
        Set rng = Selection.Tables(1).Range
        rng.Start = ActiveDocument.Range.Start
        lngIndex = rng.Tables.Count
    End If

    MsgBox "Dies ist Tabelle " & CStr(lngIndex)
End Sub
```

Die Prozedur setzt in einem ersten Schritt ein Range-Objekt auf die markierte Tabelle. Im folgenden Schritt wird der Bereich dieses Objekts angepasst. Die Startposition wird der Startposition des Dokuments gleichgesetzt. Im dritten Schritt werden die Tabellen des angepassten Range-Objekts gezählt. Da die markierte Tabelle gleichzeitig die letzte Tabelle innerhalb des Range-Objekts ist, stimmt die Anzahl der Tabellen mit der Indexnummer der markierten Tabelle überein.

## Auflistung bearbeiten

Um alle zugehörigen Objekte einer Auflistung zu bearbeiten, gibt es zwei Arten von Schleifen. Die eine ist die *For...Next*-Anweisung, die andere die *For Each...Next*-Anweisung (die *For...Next*-Anweisung wurde bereits in Kapitel 3 detailliert vorgestellt).

Grundsätzlich verfügen alle Auflistungen über eine Eigenschaft mit der Bezeichnung *Count*. In dieser Eigenschaft ist die jeweils aktuelle Anzahl der in der Auflistung vorhandenen Elemente aufgeführt. Anhand dieser Eigenschaft und einem Zähler kann eine Schleife aufgebaut werden, welche alle Elemente der betreffenden Auflistung durchläuft. Die einzelnen Elemente der Auflistung werden durch deren Index angesprochen. Ein entsprechendes Beispiel ist in Listing 5.3 dargestellt.

**Listing 5.3** Auflistung aller Elemente einer Auflistung mittels einer *For...Next*-Schleife

```
Sub Demo_For_Next()
    Dim intZähler As Integer
    Dim strText As String

    With Application.Documents
        For intZähler = 1 To .Count
            strText = strText & .Item(intZähler).Name & vbCrLf
        Next intZähler
    End With

    MsgBox strText, , "Alle geöffneten Dokumente"
End Sub
```

Anhand der *For Each...Next*-Anweisung können jedoch die einzelnen Objekte der Auflistung direkt angesprochen werden. Die Zuweisung innerhalb der Schleife weist das einzelne Objekt der betreffenden Objektvariablen zu. Das Objekt kann direkt bearbeitet werden. Ein entsprechendes Beispiel ist in Listing 5.4 dargestellt.

**Listing 5.4** Auflistung aller Elemente einer Auflistung mittels einer *For Each...Next*-Schleife

```
Sub Demo_ForEach_Next()
    Dim doc As Word.Document
    Dim strText As String

    For Each doc In Application.Documents
        strText = strText & doc.Name & vbCrLf
    Next doc

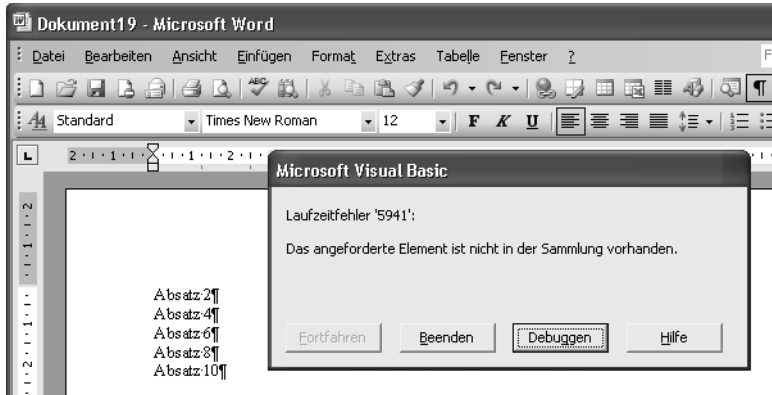
    MsgBox strText, , "Alle geöffneten Dokumente"
End Sub
```



### Elemente aus der Auflistung entfernen

Das Entfernen einzelner oder auch aller Elemente aus einer Auflistung kann heimtückisch sein. Wird dieses Vorhaben mit einer For...Next-Anweisung umgesetzt, wird regelmäßig der Laufzeitfehler 5941, »Das angeforderte Element ist nicht in der Sammlung vorhanden«, auftreten. Die Programmzeilen aus Listing 5.5 erzeugen eben diesen Fehler.

Abbildg. 5.2 Laufzeitfehler bei der Verwendung der For...Next-Schleife



Die Problematik liegt in der Count-Eigenschaft und deren Zuweisung an die For...Next-Schleife. Das Beispiel erzeugt ein Dokument mit zehn Absätzen. Bei der Zuweisung an die Schleife beträgt der Wert der Count-Eigenschaft 10. Dies bedeutet, dass die Schleife zehnmal durchlaufen wird. Bei jedem Durchgang wird ein Absatz (jener mit der Nummer der Variable intZähler) aus dem Dokument entfernt. Beim sechsten Schleifendurchgang sollte der sechste Absatz aus dem Dokument entfernt werden, im Dokument selber sind jedoch nur noch deren fünf enthalten, da in den ersten fünf Durchgängen je ein Absatz entfernt wurde. Der sechste Absatz ist nicht vorhanden, das Makro quittiert mit der Fehlermeldung 5941 seinen Dienst.

#### HINWEIS

Beachten Sie in Abbildung 5.2, welche Absätze durch die For...Next-Schleife tatsächlich bearbeitet wurden.

Der Effekt, dass nur jeder zweite Absatz bearbeitet wird, liegt daran, dass nach dem Löschen des ersten Absatzes der zweite an dessen Stelle rückt. Der zweite Schleifendurchgang löscht dann den zweiten Absatz, also jenen Absatz mit der Zahl drei im Text, usw.

Listing 5.5 Entfernen von Elementen aus der Auflistung erzeugt den Laufzeitfehler 5941

```
Sub Demo_Entfernen_ForNext()
    Dim doc As Word.Document
    Dim intZähler As Integer

    Set doc = fktNeuesDokumentErzeugen()

    For intZähler = 1 To doc.Paragraphs.Count
        doc.Paragraphs(intZähler).Range.Delete
    Next intZähler
End Sub
```

Dennoch ist es möglich, mit einer *For...Next*-Schleife eine Auflistung zu bearbeiten und einzelne oder alle Elemente zu löschen. Bei der Definition kann das optionale Schlüsselwort *Step* verwendet werden.

Das Schlüsselwort *Step* dient zur Definition der Schrittweite innerhalb der Schleife. In Listing 5.6 wird als Wert für die Schrittweite minus Eins (–1) verwendet. Dies hat zur Folge, dass die Schleife vom größten Wert zum kleinsten Wert rückwärts durchlaufen wird.

Auf diese Weise wird das Problem mit dem Laufzeitfehler 5941 elegant gelöst. In dieser Konstellation wird die Schleife vom höchsten Wert zum niedrigsten Wert durchlaufen. Das Beispiel erzeugt wiederum ein Dokument mit zehn Absätzen. Bei der Zuweisung an die Schleife beträgt der Wert der *Count*-Eigenschaft 10. Dies bedeutet, dass die Schleife zehnmal durchlaufen wird. Bei jedem Durchgang wird ein Absatz (jener mit der Nummer der Variable *intZähler*) aus dem Dokument entfernt. Da jetzt aber mit dem Wert 10 gestartet wird, wird im ersten Durchgang der zehnte Absatz entfernt, beim zweiten Durchgang wird der neunte Absatz entfernt. Dies geht so weiter, bis die Schleife abgearbeitet ist.

**Listing 5.6** Entfernen von Elementen aus der Auflistung mittels *For...Next* und *Step –1*

```
Sub Demo_Entfernen_ForNext_Step1()
    Dim doc As Word.Document
    Dim intZähler As Integer

    Set doc = fktNeuesDokumentErzeugen()

    For intZähler = doc.Paragraphs.Count To 1 Step -1
        doc.Paragraphs(intZähler).Range.Delete
    Next intZähler
End Sub
```

**WICHTIG** Beim Entfernen von Elementen aus einer Auflistung muss die *For...Next*-Schleife unbedingt vom größten zum kleinsten Element durchlaufen werden. Dies kann durch die Angabe einer negativen Schrittweite unter Verwendung des Schlüsselworts *Step –1* erreicht werden.

Wird wie in Listing 5.7 statt einer *For...Next*-Schleife eine *For Each...Next*-Schleife verwendet, taucht die genannte Problematik gar nicht erst auf. In diesem Fall wird jedes Objekt direkt der Variablen *para* zugewiesen.

Im Gegensatz zur statischen Zuweisung der benötigten Durchgänge bei der *For...Next*-Schleife ist die *For Each...Next*-Schleife dynamisch. Bei jedem Schleifendurchgang werden die entsprechenden Objekte definiert. Dies hat jedoch zur Folge, dass nach dem Entfernen eines Elements aus der Auflistung dessen Indizierung neu aufgebaut werden muss. Dieses Verhalten wird bei größeren Dokumenten zu Problemen führen, weil die Schleife dadurch immer langsamer abgearbeitet wird.

Dieses Verhalten kann sehr einfach überprüft werden. Das Makro aus Listing 5.7 muss im Einzelschritt abgearbeitet werden. Sobald die ersten Absätze entfernt wurden, wird das Dokument bearbeitet. Es können zusätzliche Absätze aufgenommen oder bestehende manuell entfernt werden. Wird die Bearbeitung durch das Makro weiter ausgeführt, ist das problemlos möglich, da die benötigte Anzahl der Schleifendurchgänge dynamisch ermittelt wird.

Wird dieser Versuch bei den anderen beiden Beispielen durchgeführt, wird die Schleife einen Laufzeitfehler erzeugen oder eben keinen Fehler mehr erzeugen, da entsprechend viele Absätze eingefügt

wurden. Aber egal, wie Sie das Dokument modifizieren, es werden garantiert nicht alle Absätze entfernt.

**Listing 5.7** Entfernen von Elementen aus der Auflistung mittels *For Each...Next*

```
Sub Demo_Entfernen_ForEachNext()
    Dim doc As Word.Document
    Dim para As Word.Paragraph

    Set doc = fktNeuesDokumentErzeugen()

    For Each para In doc.Paragraphs
        para.Range.Delete
    Next para
End Sub
```

#### HINWEIS

Bei einer *For...Next*-Schleife wird die Anzahl der benötigten Schleifendurchgänge statisch bei der Zuweisung der Schleife festgelegt.

Bei einer *For Each...Next*-Schleife wird die Anzahl der benötigten Schleifendurchgänge dynamisch bei jedem Durchgang neu überprüft.

## Selection – das verpönte Objekt

Werden für Word Makros erstellt oder wird das Programm von außen durch eine andere Applikation gesteuert, sollte grundsätzlich auf die Verwendung des *Selection*-Objekts verzichtet werden. Als Alternative zu *Selection* wird die Verwendung des *Range*-Objekts empfohlen (mehr zum *Range*-Objekt und dessen Möglichkeiten erfahren Sie in Kapitel 6).

- Das *Selection*-Objekt entspricht der Einfügemarke im Dokument. Werden komplexe Eingaben und Formatierungen vorgenommen, springt die Einfügemarke von Ort zu Ort. Die Bildschirmdarstellung ist sehr unruhig.
- Das *Range*-Objekt ermöglicht es, jeden Bereich des Dokuments direkt anzusprechen. Es können gleichzeitig mehrere Ranges definiert sein und ohne Wechsel der Einfügemarke bearbeitet werden.
- Durch die konsequente Verwendung vom *Range*-Objekt werden die einzelnen Anweisungen einfacher. Das Programm wird übersichtlicher und indirekt auch dokumentiert. Vergleichen Sie dazu Listing 5.8 und Listing 5.9.
- Wird Word als verborgenes Programm durch eine andere Applikation ferngesteuert, steht kein *Selection*-Objekt unmittelbar zur Verfügung. Dies hat zur Folge, dass alle Programmzeilen überarbeitet werden müssen und zumindest durch das Hinzufügen einer Objektvariablen, die auf die Word-Anwendung hinweist, zu ergänzen sind.
- Dem *Selection*-Objekt stehen im Verhältnis zum *Range*-Objekt nur wenige zusätzliche Eigenschaften und Methoden zur Verfügung. Für die professionelle Software-Entwicklung sind diese zusätzlichen Eigenschaften und Methoden meistens von keiner großen Bedeutung.

Trotz dieser Einwände, dass auf die Verwendung des *Selection*-Objekts verzichtet werden sollte, gibt es Konstellationen, die nur oder besser mit diesem Objekt umgesetzt werden können. Diese werden in Kapitel 6 im Abschnitt »Die gegenwärtige Markierung: *Selection* und ähnliche Objekte« zusammengefasst.

**Listing 5.8** Einfügen eines Textes in die Kopfzeile unter Verwendung des *Selection*-Objekts

```
Sub Demo_Selection()
    If ActiveWindow.View.SplitSpecial <> wdPaneNone Then
        ActiveWindow.Panes(2).Close
    End If
    If ActiveWindow.ActivePane.View.Type = wdNormalView Or ActiveWindow. _
        ActivePane.View.Type = wdOutlineView Then
        ActiveWindow.ActivePane.View.Type = wdPrintView
    End If
    ActiveWindow.ActivePane.View.SeekView = wdSeekCurrentPageHeader
    Selection.TypeText Text:="Text in Kopfzeile"
End Sub
```

**Listing 5.9** Einfügen eines Textes in die Kopfzeile unter Verwendung des *Range*-Objekts

```
Sub Demo_Range()
    Dim rng As Word.Range
    Set rng = ActiveDocument.Sections(1).Headers(wdHeaderFooterPrimary).Range
    rng.Text = "Ein anderer Text in der Kopfzeile"
End Sub
```

Die Zeilen in Listing 5.8 wurden mit dem Makrorekorder aufgezeichnet. Um einen Text mittels *Selection* in die Kopfzeile einzufügen, muss sichergestellt sein, dass sich die Einfügemarke auch wirklich in der Kopfzeile befindet. Hierfür werden acht zusätzliche Programmzeilen benötigt.

Das Gleiche wird in Listing 5.9 unter Verwendung eines *Range*-Objekts mit drei Programmzeilen erreicht. Zusätzlich ist auf den ersten Blick ersichtlich, auf welchen Bereich sich der definierte Range bezieht.



Die aufgeführten Codesequenzen finden Sie in der Beispieldatei *Bsp05\_01.doc* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap05*.

## Auto-Makros als Pseudo-Ereignisse

Im aktuellen Teil dieses Buchs werden die Möglichkeiten zu den Dokument- und Programmereignissen aufgezeichnet (siehe Kapitel 7). Neben den eigentlichen Ereignissen, die zu einem festgelegten Zeitpunkt ausgeführt werden, existieren in Word spezielle Makros, die automatisch ohne zusätzliche Interaktion des Anwenders angestoßen werden: die so genannten Auto-Makros.

Damit Word diese automatisch auszuführenden Makros als solche erkennt, müssen sie besonders benannt werden.

**Tabelle 5.1** Zusammenstellung der Auto-Makros und deren Ausführungszeitpunkt

Bezeichnung	Zeitpunkt der Ausführung
AutoOpen	Beim Öffnen eines bestehenden Dokuments.
AutoClose	Beim Schließen eines Dokuments.
AutoNew	Beim Erstellen eines neuen Dokuments.

Tabelle 5.1 Zusammenstellung der Auto-Makros und deren Ausführungszeitpunkt (Fortsetzung)

Bezeichnung	Zeitpunkt der Ausführung
AutoExec	Beim Starten von Word bzw. beim Laden eines Add-Ins.
AutoExit	Beim Beenden von Word bzw. beim Entladen eines Add-Ins.

Auto-Open  
Auto-Close

Die Makros `AutoOpen` und `AutoClose` können im aktuellen Dokument, der zugehörigen Dokumentvorlage oder in der *Normal.dot* abgespeichert werden.

AutoNew


Das Makro `AutoNew` kann in einer Dokumentvorlage oder in der *Normal.dot* abgespeichert werden.

AutoExec  
AutoExit

Die Makros `AutoExec` und `AutoExit` können in der *Normal.dot* oder einer globalen Vorlage (Add-In) abgespeichert werden.

**HINWEIS** Besteht ein Namenskonflikt, da mehrere Makros mit der gleichen Bezeichnung vorhanden sind, wird nur das Makro ausgeführt, welches eher zum Kontext passt. Demzufolge wird ein Makro im aktuellen Dokument vor jenem der zugehörigen Dokumentvorlage, ein Makro innerhalb der Dokumentvorlage vor jenem aus der *Normal.dot* und das Makro aus der *Normal.dot* vor jenem aus dem Add-In abgearbeitet.

Diese Regelung gilt nicht nur für die Auto-Makros, sondern grundsätzlich für alle Makros. Spezialfälle sind lediglich `AutoExec` und `AutoExit`. Diese beiden Makros werden für die *Normal.dot* und für jedes geladene Add-In einzeln abgearbeitet. Die Reihenfolge für das Laden der im Autostart-Ordner gespeicherten Add-Ins kann nicht beeinflusst werden.

Durch Drücken der -Taste kann verhindert werden, dass ein Auto-Makro gestartet wird (beispielsweise beim Starten von Word, beim Öffnen eines Dokuments usw.).

Das spätere Ausführen von Auto-Makros kann innerhalb einer VBA-Prozedur unterbunden werden. Die Anweisung `WordBasic.DisableAutoMacros True` verhindert das automatische Starten so lange, bis diese Anweisung zurückgesetzt oder Word erneut gestartet wird (mehr zu `WordBasic` finden Sie in Kapitel 6).

## Zusammenfassung

In diesem Kapitel soll dem Leser der Einstieg in die Welt der Objekte vermittelt werden. Das Objektmodell von Word ist mächtig und deshalb sind die Themen in diesem Kapitel wichtig.

- Es wurde vermittelt, wie ein gesuchtes Objekt einfach ermittelt (Seite 174) und dieses einer Variablen zugewiesen werden kann (Seite 175).
- Es wurde aufgezeigt wie eine Auflistung von Objekten bearbeitet und wie auf deren einzelne Objekte zugegriffen wird (Seite 179 ff.).
- Es wurde näher auf das `Selection`-Objekt eingegangen und dessen Vor- und Nachteile gegenüber dem `Range`-Objekt erläutert (Seite 183 ff.).
- Mit der Behandlung der speziellen Auto-Makros wird das Kapitel abgeschlossen (Seite 184 ff.).



## Kapitel 6

# Das Objektmodell von Word

### In diesem Kapitel:

Das <i>System-Objekt</i>	189
Die Anwendung: Das <i>Application-Objekt</i>	191
Die gegenwärtige Markierung: <i>Selection</i> und ähnliche Objekte	200
Der Kern der Sache: Das <i>Dokument-Objekt</i>	202
Dokumentvorlagen: Das <i>Template-Objekt</i>	214
Mit Bereichen arbeiten: Das <i>Range-Objekt</i>	219
Die Nadel im Heuhaufen: <i>Find/Replace einsetzen</i>	233
Formatieren mit Stil: Das <i>Style-Objekt</i>	254
Zielscheibe Textmarke: Das <i>Bookmark-Objekt</i>	271
Inhalt mit Tabellen strukturiert darstellen	280
Automatische Nummerierung mit Listen	309
Feldfunktionen	320
Grafiken: Die <i>InlineShape-</i> und <i>Shape-Objekte</i>	334
Abschnitte im Dokument: Das <i>Section-Objekt</i>	357
Bereiche im Dokument: Das <i>StoryRanges-Objekt</i>	359

Die Seite definieren: Das <i>PageSetup</i> -Objekt	363
Die Seite gestalten: Das <i>HeaderFooter</i> -Objekt	371
Lange Dokumente	377
Formulare: Das <i>FormField</i> -Objekt	381
Der Seriendruck: Das <i>MailMerge</i> -Objekt	388
Zusammenfassung	429

Sobald Sie das Ergebnis des Makrorekorders ändern oder gar ganze Prozeduren schreiben wollen, ist die Kenntnis des Word-Objektmodells unabdingbar. Und damit meinen wir nicht nur Objektnamen oder die Syntax von Methoden und Eigenschaften, sondern wie Word funktioniert. »Word for Windows«, eine Windows-Anwendung der ersten Stunde, existierte als eine steuerbare Anwendung schon vor Visual Basic. Seine Programmierschnittstellen sind organisch aus seiner Benutzerschnittstelle gewachsen, was manchem Vollblut-Entwickler gelegentlich ziemlich fremd vorkommt!

Oder, anders gesagt, Word verhält sich nicht immer auf eine Art und Weise, die der »objektorientierte« Mensch als logisch betrachtet. (Ja, sogar langjährige, fortgeschrittene Anwender hadern gelegentlich mit der internen Logik von Word ...)

Dieses Kapitel ist daher kein Wiederkauen der VBA-Hilfdateien. Vielmehr möchten wir Ihnen einerseits interessante Möglichkeiten vorstellen und Sie andererseits mit den Word-internen Zusammenhängen vertraut machen, so dass Sie in Ihrem Code das Word-Objektmodell zweckmäßig einsetzen können. Wir stellen die meist gebrauchten (oder missverstandenen) Objekte vor und erläutern anhand kurzer Code-Beispiele deren programmtechnischen Einsatz.

Ausführlichere Beispiele zum Zusammenspiel mehrerer Teile des Objektmodells finden Sie im Teil V »Lösungen«.

Die Code-Beispiele liegen in Word-VBA sowie teilweise in der .NET-Sprache C# vor. Die Autoren haben sich bemüht, Objekttypen vollumfänglich zu qualifizieren. Entwickler, die Word aus einer anderen, klassischen Visual Basic-Sprache automatisieren, sollten deshalb den VBA-Code von Word problemlos umsetzen können. Gleiches hinsichtlich des Word-Objektmodells gilt für VB.NET-Entwickler. Sie können .NET-bezogene Sachen aus den C#-Beispielen ableiten.

Etwas anders sieht es bei der Programmierung mit C# aus, da diese Sprache eine differenziertere Datentypenqualifizierung voraussetzt. Leider ist die Seitenanzahl eines Buches begrenzt und C#-Codebeispiele beinhalten meist mehr Zeilen als die VBA-Versionen. Wir konzentrieren uns deshalb auf die Verwendung des Word-Objektmodells und klammern einiges an »Drumherum«, wie z.B. die Fehlerbehandlung, aus. Beachten Sie dies bitte, wenn Sie unsere Anregungen in Ihre Projekte einbauen; versuchen Sie nicht, diese »einfach so« zu übernehmen.

Fragen zur Automatisierung finden Sie in Teil III dieses Buchs beantwortet, wo die Interoperabilität mit Word behandelt wird.



# Das System-Objekt

Wir fangen mit einer der obersten Ebenen des Word-Objektmodells an, mit dem System-Objekt. Dieses Objekt ist wenig bekannt und entsprechend wenig genutzt. Es stellt der VBA-Umgebung in Word jedoch einige Dienstleistungen und Schnittstellen der Windows-Umgebung zur Verfügung, die sonst nur über die Windows-API anzusprechen sind (siehe auch Kapitel 4).

Damit lassen sich Informationen zu Rechnertyp, Bildschirmauflösung, zur Verfügung stehende Prozessoren und Speicherplatz abfragen. Eigenschaften von besonderem Interesse stellen wir hier kurz vor.





Country-  
Region

In der Hilfe heißt es: »Gibt die Länder- bzw. Regionseinstellung des Systems zurück. Schreibgeschützter WdCountry-Wert«. Gemeint ist die Liste im Abschnitt *Standards und Formate* der Registerkarte *Regionale Einstellungen* in den *Regions- und Sprachoptionen* der Systemsteuerung. Die Hilfe präzisiert nicht, dass nicht allen Ländern ein WdCountry-Wert zugewiesen wurde. Deutschland ist beispielsweise mit wdGermany dabei, die Schweiz und Österreich jedoch nicht. Falls Sie diese Einstellung abfragen wollen, müssen Sie die numerischen Werte herausfinden, indem Sie die Einstellung ändern und diese Eigenschaft abfragen. Es stellt sich heraus, dass viele Werte der Ländervorwahl entsprechen, was die Sache etwas vereinfacht. Die Schweiz hat den Wert 41, Österreich den Wert 43.

Cursor

Mit Cursor ist der Mauszeiger gemeint. Wie wir alle wissen, dient er nicht nur zur Auswahl auf dem Bildschirm, sondern er teilt uns auch mit, wenn wir warten müssen (die Sanduhr), ob wir Text markieren können und Ähnliches. Die verschiedenen Formen der Mauszeiger werden von Anwendungsprogrammierern festgelegt, die sich ihrerseits auf Einstellungen in Windows stützen. Diese Eigenschaft ermöglicht es dem Word-VBA-Entwickler, die Form des Mauszeigers abzufragen *und festzulegen*. Da Word-Dokumente keine mausbezogenen Ereignisse wie »Mouseover« unterstützen, sind die Einsatzmöglichkeiten begrenzt, aber die Sanduhr bietet sich für lange Prozeduren geradezu an. Die Mauszeigertypen sind in Tabelle 6.1 aufgelistet.

Tabelle 6.1 Die verschiedenen Mauszeigerformen des Cursor-Objekts

WdCursor-Wert	Mauszeigerform
wdCursorIBeam	
wdCursorNormal	
wdCursorNorthwest	
wdCursorWait	

**HINWEIS** Benutzerdefinierte Formulare (»UserForms«) unterstützen 16 verschiedene Mauszeigerformen. Schauen Sie in den Eigenschaften eines Formulars nach, wenn Sie dafür die Mauszeigerform ändern möchten. Mehr über Formulare erfahren Sie in Kapitel 14.

Language  
Designa-  
tion

Gibt die gleiche Windows-Einstellung wie CountryRegion zurück, jedoch als Zeichenkette wie beispielsweise »Deutsch (Deutschland)« oder »Deutsch (Schweiz)«. Der Ausdruck wird in der Sprache des Betriebssystems wiedergegeben – in einer englischen Windows-Version z.B. »German (Germany)«.

Das Objektmodell von  
Word

Operating-System sowie Private-Profile-String

Die Eigenschaft `OperatingSystem` liefert den Namen des Betriebssystems, die Eigenschaft `Version` die zugehörige Versionsnummer. Allerdings werden Sie nicht »Windows 2000« oder »Windows XP« erhalten. Diese beiden Versionen gehören zur Familie »Windows NT«, wobei Windows 2000 der Version 5.0, Windows XP der Version 5.1 und Windows Server 2003 der Version 5.2 entspricht.

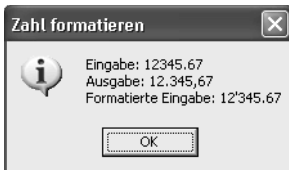
Diese ist die nützlichste aller System-Eigenschaften. Damit können Einstellungen in der Registry beliebig gelesen und geschrieben werden (im Gegensatz zur `ProfileString`-Eigenschaft, die nur den Registry-Abschnitt für Anwendungen anspricht). So greifen Sie beispielsweise direkt auf die Einstellungen unter *Regions- und Sprachoptionen* der Systemsteuerung zu, um das Dezimal- und Listentrennzeichen oder das Symbol für die Zifferngruppierung herauszufinden. Da Word nur die Zahlen- und Datumsformate umsetzt, die im Betriebssystem festgelegt sind, ist der Zugang zu dieser Information für die Arbeit mit Feldfunktionen von unschätzbarem Wert.

#### HINWEIS

Den Windows-Registry-Editor starten Sie über die Windows-Befehlsfolge *Start/Ausführen*. Geben Sie in das Feld *Öffnen* »regedit« (ohne Einführungszeichen) ein und bestätigen Sie dann mit *OK*.

Die Prozedur in Listing 6.1 zeigt, wie diese Einstellungen abgefragt und verwendet werden, um eine Zahl nach deutschem Muster zu formatieren. Das Resultat sehen Sie in Abbildung 6.1.

**Abbildg. 6.1** Die Tausender- und Dezimaltrennzeichen für die gegenwärtige Ländereinstellung wurden aus der Registry ermittelt und mit den deutschen ersetzt



**Listing 6.1** Einstellungen aus der Registry lesen

```
Private Const strMSGTITEL As String = "Zahl formatieren"

Sub ZahlMitSystemEinstellungenFormatieren()
    Dim strDezimalTrennzeichen As String
    Dim strGruppierungssymbol As String
    Dim strEingabe As String, strZahl As String
    Dim strAusgabe As String, strGanzzahl As String
    Dim strDezimal As String
    Dim lTrennstelle As Long, lZaehler As Long

    strEingabe = "12345.67"
    With System
        strDezimalTrennzeichen = .PrivateProfileString( _
            FileName:="", _
            Section:="HKey_Current_User\Control Panel\International", _
            Key:="sDecimal")
        strGruppierungssymbol = .PrivateProfileString(FileName:="", _
            Section:="HKey_Current_User\Control Panel\International", _
            Key:="sThousand")
    End With
    If IsNumeric(strEingabe) Then
```

Listing 6.1 Einstellungen aus der Registry lesen (Fortsetzung)

```

'Wurde vom System als gültige Zahl erkannt.
'In eine Zahl konvertieren, um Tausendertrennzeichen zu entfernen
strZahl = CStr(CDbl(strEingabe))
'Wenn nötig, System-Dezimaltrennzeichen mit deutschen ersetzen
If strDezimalTrennzeichen <> "," Then
    strZahl = Replace(strZahl, strDezimalTrennzeichen, ",")
End If
lTrennstelle = InStr(strZahl, ",")
'Dezimalinformation separat speichern
If lTrennstelle = 0 Then
    'Dezimalstellen hinzufügen, wenn keine vorhanden
    strDezimal = ".00"
    strZahl = strZahl & strDezimal
Else
    strDezimal = Mid(strZahl, lTrennstelle, Len(strZahl) - lTrennstelle + 1)
End If

'Ganzzahlinformation abtrennen und mit Tausendertrennzeichen ergänzen
strGanzzahl = Left(strZahl, Len(strZahl) - Len(strDezimal))
For lZaehler = 1 To Len(strGanzzahl)
    strAusgabe = Mid(strGanzzahl, Len(strGanzzahl) - lZaehler + 1, 1) & strAusgabe
    If (lZaehler Mod 3) = 0 Then
        strAusgabe = "." & strAusgabe
    End If
Next lZaehler
strAusgabe = strAusgabe & strDezimal
End If
MsgBox "Eingabe: " & strEingabe & vbCrLf & "Ausgabe: " & strAusgabe & vbCrLf & _
    "Formatierte Eingabe: " & Format(strEingabe, "#,##0.00"), _
    vbInformation + vbOKOnly, strMSGITITEL
End Sub

```

PrivateProfileString lässt sich auch für den Zugriff auf INI-Dateien verwenden. Geben Sie die Pfadangaben zur INI-Datei in das Argument FileName ein. Die Funktion kann aber nur Schlüssel lesen, schreiben und erstellen, eignet sich also nicht für die allgemeine Verwaltung und Erstellung von INI-Dateien. Dafür braucht man die Windows-API (siehe Kapitel 4).



Die Beispieldatei *Bsp06\_01\_System.doc* finden Sie auf der CD-ROM im Ordner \Kap06.

**HINWEIS**

Mehr zum Thema Registry, INI-Dateien und das Speichern von Benutzer- und Anwendungseinstellungen enthält Kapitel 12.

## Die Anwendung: Das *Application*-Objekt

Das Application-Objekt ist die Word-Anwendung. Denken Sie zurück an die erste Abbildung dieses Buches in Kapitel 1: ein grafisches Diagramm dieses Objekts. Es beinhaltet alles, was mit Word zu tun hat, unter anderem:

- die Dokumente,

- die Fenster, worin diese angezeigt werden,
- die Symbolleisten, womit der Benutzer arbeitet,
- die Einstellungen und Optionen.

Solange ein Entwickler innerhalb von Word mit dem VB-Editor arbeitet, muss er dieses Objekt nur selten, wenn überhaupt, in ein Code-Modul eintippen. »Visual Basic für Applikationen« ist Anwendungs-spezifisch und weiß, zu welcher Anwendung es gehört. Das erspart dem Programmierer die »zusätzliche« Arbeit, Objektnamen der obersten Ebene mit `Application` qualifizieren zu müssen.

Wird Word dagegen aus einer anderen Umgebung heraus gesteuert, sei es aus Excel, Access, Visual Basic oder dem .NET Framework, muss eine Objektvariable für das `Application`-Objekt deklariert und ihr eine Instanz der laufenden Anwendung zugewiesen werden. Über diese Objektvariable werden die in der Objektmodellhierarchie tiefer gestellten Objekte angesprochen. Dieser Vorgang wird in Teil III dieses Buches über die Interoperabilität behandelt und taucht auch in manchem .NET-Codebeispiel dieses Buches auf.

Einzelne Objekte von besonderem Interesse werden in anderen Abschnitten und Kapiteln behandelt. Hier verweisen wir lediglich auf die folgenden Elemente:

- `ScreenUpdating`
- `Visible`
- `DisplayAlerts`
- `AutomationSecurity`
- `PathSeparator`
- `Language`
- Optionen (Options-Objekt)
- `DefaultFilePath`
- WordBasic-Befehle

Screen-  
Updating

Durch die Verwendung bestimmter Objekte – statt der allgemeinen Objekte wie `Selection` oder `ActiveDocument` – wird das »Flackern« des Bildschirms bereits erheblich reduziert. Eine weitere Verringerung kann erreicht werden, indem wir `Application.ScreenUpdating` (Bildschirmaktualisierung) auf `False` setzen:

```
Application.ScreenUpdating = False
```

Wenn Sie Word von einer anderen Umgebung aus automatisieren und Ihr Code interaktiv mit dem Benutzer agiert, sollten Sie die Eigenschaft wieder auf `True` setzen, bevor die Kontrolle dem Benutzer übergeben wird. Sonst bleibt der Bildschirm »gesperrt«, was die Bearbeitung des Dokuments erheblich erschwert.

Diese Eigenschaft unterdrückt nicht alle Handlungen im Word-Fenster. Wechselt der Code beispielsweise die Ansicht, das Fenster oder die Fenstergröße, werden diese Änderungen sichtbar wiedergegeben. Auch Meldungen werden durch diese Eigenschaft nicht tangiert.

Visible

Es ist auch möglich, ein Dokument-Fenster zu minimieren oder sogar das Dokument oder die ganze Anwendung mittels der `Visible`-Eigenschaft unsichtbar zu machen:

```
Application.Visible = False
```

Falls Sie Word von einer anderen Umgebung aus steuern, startet Word automatisch unsichtbar. Um mit dem Benutzer interaktiv zu arbeiten, müssen Sie zuerst die Anwendung sichtbar machen, indem Sie diese Eigenschaft auf True stellen.

**WICHTIG** Die Word-Anwendung wurde als interaktive Schnittstelle konzipiert. Das Layout eines Dokuments stützt sich auf den dynamischen Textfluss auf der virtuellen Seite. Ist das Dokumentfenster unsichtbar, wird das Fertigstellen des Layouts eventuell beeinträchtigt, was abweichende Automatisierungsergebnisse liefern kann. Dies wird hauptsächlich bei der Arbeit mit dem Selection-Objekt zum Problem, kann aber auch in anderen Fällen auftreten. Liefert Ihr Code bei nicht sichtbarem Dokument-Fenster ein unterschiedliches Resultat, müssen Sie das Dokument-Fenster sichtbar machen.

Display-  
Alerts

Wie bereits erwähnt, ist Word als interaktive Anwendung konzipiert und blendet für den Benutzer gelegentlich auch Meldungen ein. Für eine Automatisierung sind diese oft hinderlich, lassen sich jedoch nicht gänzlich abschalten. Immerhin können wir einige der (aus Sicht der Anwendung) weniger kritischen Meldungen mit der Eigenschaft DisplayAlerts unterbinden:

```
Application.DisplayAlerts = wdAlertsNone
```

In C#:

```
wdApp.DisplayAlerts = wd.WdAlertLevel.wdAlertsNone
```

Dabei sind drei Einstellungen zu unterscheiden: wdAlertsNone (so viele Meldungen wie möglich werden unterbunden), wdAlertsMessageBox (nur Hinweise werden unterdrückt, Fehlermeldungen jedoch weiterhin eingeblendet) und wdAlertsAll (normales Verhalten).

**ACHTUNG** Nach der Beendigung der Code-Ausführung wird diese Eigenschaft *nicht* automatisch auf wdAlertsAll zurückgesetzt. Sie müssen in Ihrem Code selbst dafür sorgen. Das gilt sowohl für die Automatisierung aus einer anderen Umgebung als auch für Code in Word-VBA-Modulen.

Automa-  
tion-  
Security

Nicht selten muss Automatisierungscode Dokumente in Word öffnen. Eine Meldung, die DisplayAlerts nicht unterdrückt, ist die Makrosicherheitsmeldung, wenn der Benutzer die Stufe *Mittel* festgelegt hat (mehr zum Thema Makrosicherheit lesen Sie in Kapitel 1). Aus verständlichen Sicherheitsgründen bietet das Objektmodell keinen Zugang zu dieser Einstellung. Aber sobald eine Anwendung läuft, soll es eigentlich Dokumente öffnen können, ohne dass der Ablauf durch Meldungen unterbrochen wird.

In Office XP wurde die Eigenschaft AutomationSecurity eingeführt. Damit kann der Entwickler die Sicherheitsstufe *für die Laufzeit seines Codes* festlegen. Dafür gibt es drei Konstantenwerte: msoAutomationSecurityByUI (verwendet die im Dialogfeld *Sicherheit* angegebene Sicherheitseinstellung), msoAutomationSecurityForceDisable (deaktiviert alle Makros in allen programmatisch geöffneten Dateien, ohne Sicherheitsmeldungen anzuzeigen) sowie msoAutomationSecurityLow (aktiviert alle Makros = der Standardwert der Eigenschaft).

Meistens wählt der Entwickler msoAutomationSecurityForceDisable oder msoAutomationSecurityLow, je nachdem, ob er die Ausführung von darin enthaltenen Makros erlauben will oder nicht.

```
Application.DisplayAlerts = msoAutomationSecurityLow
```

In C# muss explizit auf das Office-Objektmodell verwiesen werden:

```
wdApp.AutomationSecurity = _  
Microsoft.Office.Core.MsoAutomationSecurity.msoAutomationSecurityForceDisable;
```

#### HINWEIS

Ein großes Problem für das automatisierte Öffnen von Dokumenten sind darin enthaltene Auto-Makros, die beim Öffnen eine Handlung ausführen. Diese kommen oft der Ausführung des eigenen Codes in die Quere. Mit `ApplicationSecurity` auf `msoAutomationSecurityForceDisable` kann das Problem umgangen werden, die übrige Makro-Funktionalität bleibt aber gesperrt. Es gibt einen alten WordBasic-Befehl, `DisableAutoMacros`, der nur die Auto-Makros ausschaltet. Mehr über die Verwendung von WordBasic finden Sie weiter unten in diesem Abschnitt.

Path-  
Separa-  
tor

Nicht nur Word für Windows (WinWord), sondern auch Word für Macintosh (MacWord) bietet VBA als Programmierschnittstelle. Die beiden Betriebssysteme definieren Pfadangaben jedoch anders. In Plattform-übergreifendem Code ist daher die `PathSeparator`-Eigenschaft des `Application`-Objekts sehr hilfreich, die das Pfadtrennzeichen für das aktuelle Betriebssystem – für Windows einen Backslash (\); für Macintosh ein Kolon (:) – zurückgibt.

Language

Die `Language`-Eigenschaft gibt eine Ganzzahl zurück, die einem `MsoLanguageID`-Konstantwert entspricht. Daraus wird ermittelt, welche Sprache in der Word-Umgebung herrscht. Eine Liste der `MsoLanguageID`-Konstantwerte finden Sie im Objektkatalog des VB-Editors.

#### HINWEIS

Für die englische Word-Version haben Firmen mit Lizenzverträgen die Möglichkeit, das Office Multilingual User Interface Pack zu erwerben. Damit können Menüeinträge, Dialogfelder und die Hilfe in der ausgewählten Sprache angezeigt werden.

Options

Mittlerweile enthält Word 2003 etwa 120 Eigenschaften zum `Options`-Objekt. Jede steht für einen Eintrag im Menüfeld *Extras/Optionen* sowie *Extras/AutoKorrektur-Optionen/AutoFormat* und *AutoFormat während der Eingabe*. Manche sind in der deutschen Version nicht sichtbar, da es – vor allem für die asiatischen Versionen – auch sprachspezifische Einstellungen gibt.

#### HINWEIS

Sie werden nicht alle Einträge aus *Extras/Optionen* in dieser Liste finden. Es geht aus dem Dialogfeld leider nicht klar hervor, aber einige Optionen gelten für die Anwendung, während andere dokumentspezifisch sind. Letztere sind Eigenschaften des `Document`-Objekts.

Manchmal ist es schwierig, herauszufinden, welche Option für den bestimmten Eintrag zuständig ist, den Sie beeinflussen wollen, denn die Eigenschaftennamen sind nicht immer eindeutig. Sie können entweder die Hilfe zu jeder Eigenschaft durchforsten oder sich des Makrorekorders bedienen.

Wenn Sie für diese Aufgabe den Makrorekorder hinzuziehen, denken Sie daran, dass er alle Optionen einer Registerkarte aufzeichnet und nicht nur diejenigen, deren Einstellungen geändert wurden. Dies bedeutet:

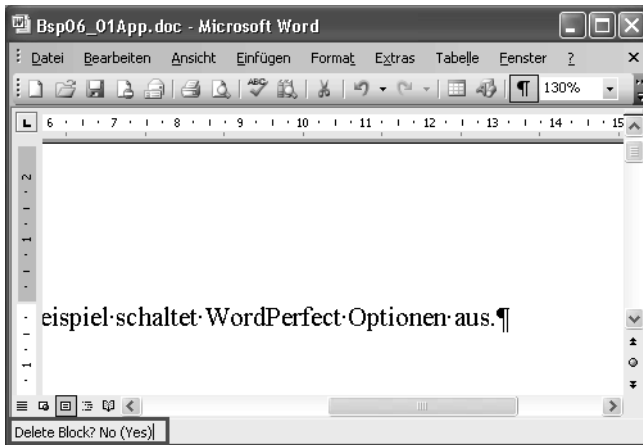
- Um die Übersicht zu behalten, sollten Sie beim Aufzeichnen nur eine Registerkarte bearbeiten und nur wenige Optionen ändern.

- Unter Umständen müssen Sie zwei Makros aufzeichnen – eines, das die Einstellungen einschaltet, und ein zweites, das sie ausschaltet – und den resultierenden Code vergleichen, um festzustellen, welche Eigenschaften geändert wurden.

Vergessen Sie nie, sich dem Benutzer gegenüber höflich zu verhalten. Wenn Sie für die Ausführung einer Aufgabe Optionen ändern, sollten diese am Schluss auf den ursprünglichen Wert zurückgestellt werden. (Dieser Grundsatz gilt natürlich nicht, wenn es der Zweck einer Prozedur ist, eine Einstellung zu ändern.)

Hier als Beispiel die kleine Prozedur in Listing 6.2 bzw. Listing 6.3, um ein ärgerliches Verhalten auszuschalten, das gelegentlich in unseren Breitengraden auftaucht. In der deutschen Benutzerschnittstelle stehen die Optionen für die WordPerfect-Hilfe und -Tastaturkürzel nicht zur Verfügung, aber irgendein Add-In bringt es fertig, sie einzuschalten. Plötzlich wird, wie in Abbildung 6.2 ersichtlich, in der Statusleiste auf Englisch nachgefragt, ob der markierte Textblock tatsächlich gelöscht werden soll – »Delete Block? No (Yes)«. Der Benutzer muss entsprechend »Y« oder »N« eingeben, was natürlich überaus lästig ist.

Abbildg. 6.2 Die Aufforderung »Delete Block?« ist die Auswirkung einer eingeschalteten WordPerfect-Option



Listing 6.2 Die WordPerfect-Optionen ausschalten

```
Sub DeleteBlockAusschalten()
    Application.Options.WPDocNavKeys = False
    Application.Options.WPHelp = False
End Sub
```

Listing 6.3 .NET: C#-Version: Eine bestehende Verbindung zur Word-Anwendung über *wdApp* wird angenommen

```
private void DeleteBlockAusschalten_CS()
{
    wdApp.Options.WPDocNavKeys = false;
    wdApp.Options.WPHelp = false;
}
```

Default-  
FilePath

Eine weitere wichtige Option entspricht den Einstellungen der Registerkarte *Speicherort für Dateien*. Damit können Sie beispielsweise ermitteln, wo sich der *Startup*-Ordner von Word sowie die Benutzer- und Arbeitsgruppenvorlagen befinden. Die unterstützten Werte sind in der Tabelle 6.2 aufgelistet. Einige dieser Angaben werden von der Word-Anwendung nicht mehr benutzt, stehen aber aus Gründen der Rückwärtskompatibilität noch zur Verfügung. Solche dürfen Sie für eigene Zwecke einsetzen, mit dem Vorbehalt, dass Microsoft sie jederzeit entfernen oder wieder beanspruchen könnte. Um den Pfad zum *Startup*-Ordner zu ermitteln:

```
strStartupPfad = Application.Options.DefaultFilePath(wdStartupPath)
```

In C#:

```
wd.WdDefaultFilePath sup = wd.WdDefaultFilePath.wdStartupPath;  
string startupPfad = wdApp.Options.get_DefaultFilePath(sup);
```

**Tabelle 6.2** Pfadnamen zu Anwendungs-relevanten Ordnern

WdDefaultFilePath-Enum	Wert	Beschreibung
wdAutoRecoverPath	5	Speicherort für AutoWiederherstellen-Dateien
wdBorderArtPath	19	Speicherort für Rahmenmuster (irrelevant in Word)
wdCurrentFolderPath	14	Der momentan aktive Ordner der Word-Anwendung. Oft, aber nicht immer, Speicherort des aktuellen Dokuments. Enthält ein Dokument relative Verknüpfungen zu anderen Dateien, wird die relative Pfadangabe zu diesem Speicherort gerechnet.
wdDocumentsPath	0	Ordner, in den neue Dokumente standardmäßig gespeichert werden. Entspricht dem Dateityp <i>Dokumente</i> im Dialogfeld.
wdGraphicsFiltersPath	10	Der Ordner, in den das Installationsprogramm von Word (Office) die Konvertierfilter für grafische Dateien gespeichert hat.
wdPicturesPath	1	Entspricht dem Dateityp <i>ClipArtgrafiken</i> im Dialogfeld. Wird für den Programmablauf der neueren Word-Versionen nicht gebraucht und ist standardmäßig leer. Das Objektmodell gibt den Pfad zu den Office-Anwendungen zurück. Wird ein Pfadname zugewiesen, startet <i>Einfügen/Grafik/Aus Datei</i> immer in diesem Ordner.
wdProgramPath	9	Installationsort der Datei <i>winword.exe</i> .
wdProofingToolsPath	12	Installationsort der *.lex-Dateien. Bis Word 97 befanden sich hier standardmäßig auch die *.dic-Dateien. Seit Word 2000 werden diese im Benutzerprofil gespeichert.
wdStartupPath	8	Entspricht dem Dateityp <i>AutoStart</i> im Dialogfeld. Vorlagen in diesem Ordner werden von Word automatisch beim Starten geladen. Ferner wird allen darin stehenden Makros automatisch vertraut.
wdStyleGalleryPath	15	Wo der Formatvorlagenkatalog die aufgelisteten Vorlagen findet. (Die Funktionalität steht seit Word 2002 nicht mehr in der Benutzerschnittstelle, ist aber im Objektmodell noch vorhanden.)



Tabelle 6.2 Pfadnamen zu Anwendungs-relevanten Ordnern (Fortsetzung)

WdDefaultFilePath-Enum	Wert	Beschreibung
wdTempFilePath	13	Speicherort für temporäre Dateien von Word, die während der Bearbeitung von Dokumenten angelegt werden. Erwartet und gibt Pfadnamen zurück, die dem DOS 8.3-Muster entsprechen.
wdTextConvertersPath	11	Installationsort der Office-Textkonvertierfilter
wdToolsPath	6	Installationsort der Office-Anwendungen
wdTutorialPath	7	Wurde in einigen Word-Versionen für den Installationsort der Einführungsdateien gebraucht. In Word 2003 ist diese Eigenschaft standardmäßig leer.
wdUserOptionsPath	4	Als die Office-Anwendungen Benutzereinstellungen in INI-Dateien festhielten, der Pfad zu diesem Ordner. In neueren Versionen von Word wird standardmäßig der Pfad zum Ordner <i>Eigene Dateien</i> zurückgegeben. Daraus kann man den Pfad zu allen Benutzerprofilordnern ableiten.
wdUserTemplatesPath	2	Entspricht dem Dateityp <i>Benutzervorlagen</i> im Dialogfeld. Speicherort der Vorlagen, die im Dialogfeld <i>Vorlagen</i> (Menübefehl <i>Datei/Neu</i> ) aufgelistet sind. Darin enthaltenen Makros wird automatisch vertraut.
wdWorkgroupTemplatesPath	3	Entspricht dem Dateityp <i>Arbeitsgruppenvorlagen</i> im Dialogfeld. Speicherort, meistens im Netzwerk, von Vorlagen, die mehrere Benutzer teilen.

Word-Basic

Der Kerncode der Word-Anwendung wurde in den späten achtziger Jahren geschrieben. Die damalige Philosophie war, dass Word möglichst anpassungsfähig sein sollte. Deshalb wurden die Schnittstellen zu allen internen Befehlen über die Programmiersprache WordBasic offen gelegt. Es war möglich, Dialogfeldeinstellungen zu lesen und festzulegen sowie einen internen Ablauf durch den eigenen Code zu ersetzen. Viele dieser Möglichkeiten sind im VBA-Zeitalter noch vorhanden und werden in Teil IV dieses Buchs über die Benutzerschnittstelle behandelt.

In vielen Fällen haben VBA-Methoden und -Eigenschaften die alten WordBasic-Befehle ersetzt. Es gibt jedoch einige sehr nützliche Elemente, für die in VBA kein Äquivalent bereitgestellt wurde. Zudem wurde, aus irgendeinem unerklärlichen Grund, für einige Funktionalität keine VBA-Schnittstelle geschaffen. Da aber Word intern immer noch auf der alten WordBasic-Basis aufgebaut ist, haben wir über WordBasic-Befehle einen begrenzten Zugang.

Einige nützliche WordBasic-Befehle sind in der Tabelle 6.3 aufgelistet. Ein Beispiel sehen Sie in Listing 6.4 bzw. in Listing 6.5, dessen Resultat in Abbildung 6.3 abgebildet ist. Wichtig bei der Arbeit mit WordBasic-Befehlen ist, immer daran zu denken, dass diese Befehle sich ausschließlich auf die gegenwärtige Markierung auswirken.

**HINWEIS**

Falls Sie WordBasic in VBA konvertieren müssen, wird Ihnen das Hilfe-Thema »Visual Basic-Entsprechungen zu WordBasic-Befehlen« gute Dienste leisten.

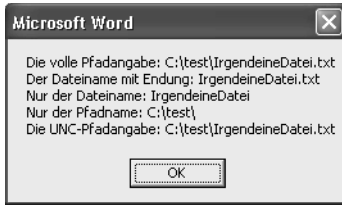
**Tabelle 6.3** Nützliche *WordBasic*-Befehle

WordBasic-Befehl	Beschreibung
<code>WordBasic.SelectSimilarFormatting</code>	<p>In Word 2002 wurde eine Funktionalität eingeführt, die es dem Benutzer ermöglicht, nicht zusammenhängende Dokumentbereiche gleichzeitig zu markieren (<b>[Strg]</b>-Taste festhalten und mit der Maus markieren). Zudem enthält das Kontextmenü eines Eintrags aus dem Aufgabenbereich <i>Formatvorlagen und Formatierungen</i> den Befehl <i>Alle Instanzen markieren</i>, der sämtliche Vorkommen einer bestimmten Formatierung markiert. Keine dieser Möglichkeiten wurde in der VBA-Schnittstelle berücksichtigt.</p> <p>Für die letztere gibt es jedoch einen Befehl im WordBasic-Bereich der Anwendung. Er sucht im Dokument weitere Vorkommen der Formatierung, in der sich die Einfügemarke gegenwärtig befindet, und markiert sie alle.</p>
<code>WordBasic.DisableAutoMacros</code>	<p>Die Word-Anwendung unterstützt seit jeher einen Satz »Auto-Makros«, die bei bestimmten Handlungen (z.B. Erstellen, Öffnen oder Schließen eines Dokuments) automatisch ausgeführt werden. Obwohl diese unter VBA noch laufen, müsste der Entwickler nach den Vorstellungen von Microsoft Dokument- und Application-Ereignisse einsetzen. Was in beiden Fällen in die VBA-Schnittstelle fällt*, ist die Möglichkeit, die Ausführung zu unterbinden. Dieser WordBasic-Befehl reguliert die Ausführung von »Auto-Makros«.</p> <p><b>WordBasic.DisableAutoMacros 1</b> schaltet sie aus; <b>WordBasic.DisableAutoMacros 0</b> lässt die Ausführung zu.</p>
<code>WordBasic.SortArray aArray()</code>	<p>Um per VBA etwas zu sortieren, müssen Sie selber die Funktionalität einbauen; es gibt dafür keine internen Befehle. Mit <b>WordBasic.SortArray</b> können ohne großen Aufwand einfache Arrays sortiert werden. Mehr Informationen zu den Optionen enthält die Dokumentation.</p>
<code>WordBasic.FilePrintSetup</code>	<p>Wenn Sie in Word-VBA mit <b>Application.ActivePrinter</b> den Drucker ändern, wird auch die standardmäßige Druckereinstellung des Systems geändert. Der WordBasic-Befehl ändert den Drucker in Word, ohne die Systemeinstellung zu ändern.</p>
<code>WordBasic.FileNameInfo\$()</code>	<p>Damit können Sie die verschiedenen Teile einer beliebigen Pfadangabe (Zeichenkette) herauslösen, ohne die Funktionalität selber schreiben zu müssen. Bitte beachten Sie: der Pfad muss in der Umgebung vorhanden sein, die Datei aber nicht.</p>
<p>* In Word 2002 wurde die Eigenschaft <b>AutomationSecurity</b> eingeführt, die alle Makros eines Dokuments beim Öffnen sperren kann (mehr dazu lesen Sie weiter oben unter <b>Application</b>).</p>	

**TIPP**

Die WordBasic-Befehle werden in keiner aktuellen VBA-Dokumentation aufgeführt. Die letzte Quelle war die Hilfe zu Word 95. Microsoft hat diese (in der englischen Version; nur englische Befehle werden in den neueren Word-Versionen erkannt) freundlicherweise zum Herunterladen bereitgestellt: <http://www.microsoft.com/downloads/details.aspx?displaylang=en&FamilyID=1A24B2A7-31AE-4B7C-A377-45A8E2C70AB2> und wurde zusätzlich auf der CD-ROM zum Buch im Ordner `\Beilagen\Word95 Wordbasic Hilfe` abgelegt.

Abbildg. 6.3 Ergebnisse der Funktion *WordBasic.FileNameInfo\$*



Listing 6.4 Mit der Funktion *WordBasic.FileNameInfo\$()* Pfadinformationen ermitteln

```
Sub PfadangabenInfoAuslisten()
    Dim strPfadangabe As String

    strPfadangabe = "C:\test\IrgendeineDatei.txt"
    With Application.WordBasic
        MsgBox "Die volle Pfadangabe: " & .FileNameInfo(strPfadangabe, 1) & _
            vbCrLf & "Der Dateiname mit Endung: " & .FileNameInfo(strPfadangabe, 3) & _
            vbCrLf & "Nur der Dateiname: " & .FileNameInfo(strPfadangabe, 4) & _
            vbCrLf & "Nur der Pfadname: " & .FileNameInfo(strPfadangabe, 5) & _
            vbCrLf & "Die UNC-Pfadangabe: " & .FileNameInfo(strPfadangabe, 6)
    End With
End Sub
```

Listing 6.5 (.NET): In C# kann WordBasic nur über »Late Binding« angesprochen werden

```
using System.Reflection;

private void Pfadangaben(string pfadangabe)
{
    object oWdBsc = InvokeHelper("WordBasic", GetProp, null, wdApp, null);
    object oFN = (object) pfadangabe;
    object oFI1;
    object oFI3;
    object oFI4;
    object oFI5;
    object oFI6;
    int dateiPfad = 1;
    int dateiNameMitEndung = 3;
    int dateiName = 4;
    int nurPfad = 5;
    int pfadUNC = 6;
    oFI1 = InvokeHelper("FileNameInfo$", InvMethod | GetProp, null, oWdBsc, oFN, dateiPfad);
    oFI3 = InvokeHelper("FileNameInfo$", InvMethod | GetProp, null, oWdBsc, oFN,
        dateiNameMitEndung);
    oFI4 = InvokeHelper("FileNameInfo$", InvMethod | GetProp, null, oWdBsc, oFN, dateiName);
    oFI5 = InvokeHelper("FileNameInfo$", InvMethod | GetProp, null, oWdBsc, oFN, nurPfad);
    oFI6 = InvokeHelper("FileNameInfo$", InvMethod | GetProp, null, oWdBsc, oFN, pfadUNC);
    MessageBox.Show(String.Format(
        "Die volle Pfadangabe: {0}\nDer Dateiname mit Endung: {1}\n" +
        "Nur der Dateiname: {2}\nNur der Pfadname: {3}\nDie UNC-Pfadangabe: {4}",
        oFI1.ToString(), oFI3.ToString(), oFI4.ToString(), oFI5.ToString(), oFI6.ToString()));
    oWdBsc = null;
}

private BindingFlags InvMethod = BindingFlags.InvokeMethod;
```

**Listing 6.5** (.NET): In C# kann WordBasic nur über »Late Binding« angesprochen werden (*Fortsetzung*)

```
private BindingFlags GetProp = BindingFlags.GetProperty;

private object InvokeHelper(string prop, System.Reflection.BindingFlags flgs,
    System.Reflection.Binder bndr, object oTrgt, params object[] args)
{
    return oTrgt.GetType().InvokeMember(prop, flgs, bndr, oTrgt, args);
}
```



Die Beispieldatei *Bsp06\_01\_App.doc* finden Sie auf der CD-ROM im Ordner \Kap06.

## Die gegenwärtige Markierung: *Selection* und ähnliche Objekte

Während unserer Arbeit als MVPs in den Newsgroups sehen wir viel Code, der auf dem *Selection*-Objekt und Ähnlichem basiert, was nicht überrascht, weil der Makrorekorder (siehe Kapitel 1) genau dies liefert. Im Allgemeinen raten wir davon ab, sich im Code auf das *Selection*-Objekt zu verlassen. Es ist unzuverlässig und nicht unproblematisch für die Organisation und Wartung des Codes. Genauer formuliert:

- Es ist unklar, was Code, der die gegenwärtige Markierung im Dokument manipuliert, tun soll, da wir keine Ahnung haben, wo die Markierung sich bei der Ausführung befinden soll. Solche Prozeduren müssen ausgiebig kommentiert werden, um sie längerfristig zu unterhalten.
- Da wir uns auf die Markierung verlassen müssten, ist es schwieriger, zuverlässigen, fehlerfreien Code zu schreiben.
- Weil die Markierung verschoben werden muss, ist die Ausführung langsamer, und der Bildschirm »hüpft« bei fast jeder Handlung.

Machen Sie es sich zur Gewohnheit, *Selection* möglichst aus Ihrem Code zu verbannen, und arbeiten Sie stattdessen mit den Objekten. Muss die gegenwärtige Markierung als Anfangspunkt dienen, weisen Sie sie einer Variablen des entsprechenden Datentyps zu, etwa:

```
Dim rng as Word.Range
Set rng = Selection.Range
```

oder

```
Dim tbl as Word.Table
'Die erste Tabelle in der Markierung
'(auch die Tabelle, worin sich die Markierung befindet)
Set tbl = Selection.Tables(1)
```

Wie bei allen Regeln, gibt es Ausnahmen, wo der Einsatz von *Selection* vorteilhaft oder sogar unabdingbar ist. Ein Beispiel stellt die Formatierung von Tabellenspalten dar. Da eine Tabellenspalte kein zusammenhängender Bereich im Dokument ist, kann sie einer Variablen des Typs *Range* nicht zuge-

wiesen werden; ebenso gibt es kein Objekt vom Typ »Spalte«. Entweder muss jede einzelne Zelle in der Spalte bearbeitet werden, oder die Spalte wird markiert und die Formatierung auf das Selection-Objekt ausgeführt. In diesem Fall ist die Ausführung mit Selection schneller und die auszuführenden Handlungen sind klar auf die markierte Spalte bezogen; somit entfallen zwei der oben erwähnten Einwände.

Was für die Markierung im Text gilt, gilt auch für Objekte wie ActiveDocument und ActiveWindow, die das gegenwärtig bearbeitete Dokument bzw. das Fenster mit dem Fokus repräsentieren. Auch diese sollten Sie einer Objektvariablen zuweisen, etwa: Set dok = ActiveDocument bzw. Set win = ActiveWindow. Beispielen für solche Zuweisungen werden Sie in den Listings dieses Handbuchs immer wieder begegnen.

Die meisten Eigenschaften und Methoden des Selection-Objekts hat auch das Range-Objekt, es gibt jedoch einige zusätzliche, die erwähnenswert sind.

**Type** Eine der wichtigsten Eigenschaften ist Type. Sie ermittelt, wo sich die Markierung befindet. Diese Auskünfte überschneiden sich zum Teil mit denen der Eigenschaft Information (mehr dazu lesen Sie im Abschnitt »Mit Bereichen arbeiten: Das Range-Objekt« in diesem Kapitel).

Soll der Code beispielsweise etwas kopieren, ist es wichtig zu wissen, ob erstens eine Markierung überhaupt vorliegt, und zweitens, ob sie einen Textblock (wdSelectionBlock) umschließt oder normal erstellt wurde (wdSelectionNormal). Im folgenden Code-Schnipsel wird getestet, ob eine normale Markierung vorliegt; wenn ja, wird der markierte Text kopiert.

```
If Selection.Type = wdSelectionNormal Then
    Selection.Copy
End If
```

Die Markierungsarten finden Sie in Tabelle 6.4 aufgelistet.

**Tabelle 6.4** Die Art der Markierung herausfinden

WdSelectionType-Enum	Wert	Beschreibung
wdSelectionBlock	6	Ein Rechteck von Text wurde mit Festhalten der <span style="border: 1px solid black; padding: 0 2px;">Alt</span> -Taste markiert.
wdSelectionFrame	3	Ein Positionsrahmen ist markiert.
wdSelectionIP	1	Die Einfügemarke blinkt im Text.
wdSelectionRow	5	Eine oder mehrere Tabellenzeilen sind markiert.
wdNoSelection	0	(Die Autoren haben diesen Zustand nie feststellen können)
wdSelectionColumn	4	Eine oder mehrere Tabellenspalten sind markiert.
wdSelectionInlineShape	7	Eine Grafik »mit Text in Zeile« ist markiert.
wdSelectionNormal	2	Text (ein oder mehrere Zeichen) wurde normal markiert.
wdSelectionShape	8	Eine mit Textfluss formatierte Grafik ist markiert.



```
Dim doc As Word.Document
Set doc = ActiveDocument
```

In C#:

```
Word.Document doc = wdApp.ActiveDocument
```

Wird ein Dokument geöffnet oder neu angelegt, wird es der Objektvariablen direkt zugewiesen:

```
Dim doc As Word.Document
Dim docNeu As Word.Document
Set doc = Documents.Open("C:\test\Test.doc")
Set docNeu = Documents.Add
```

Das C#-Beispiel fällt etwas länger aus, da in C# *alle* Argumente einer Methode oder Eigenschaft angegeben werden müssen, auch wenn sie als »optional« bezeichnet sind. Dazu verlangt die Schnittstelle zu COM, dass die Argumente in der Form *ref object* übergeben werden müssen. Das Listing 6.6 verwendet die Methoden *Open* und *Add* des Document-Objekts in Word 2003.

#### HINWEIS

Wenn Sie für mehrere Word-Versionen programmieren, denken Sie daran, dass die Anzahl der Argumente für eine Methode nicht unbedingt konstant bleibt. Um neue Funktionalität zu berücksichtigen, werden Methoden mit zusätzlichen, meist »optionalen« Argumenten ergänzt. Die *Open*- und *Add*-Methoden des Document-Objekts sind dafür Paradebeispiele. Diese Tatsache stellt für den VB-Entwickler keine Probleme dar, da seine Umgebung optionale Argumente unterstützt. In C# hingegen müssen die Funktionsaufrufe genau mit den Definitionen der Objektbibliothek übereinstimmen.

Listing 6.6

(.NET): Beispiele für die Methoden *Open* und *Add* des *Document*-Objekts in Word 2003

```
private void btnDoks_Click(object sender, System.EventArgs e)
{
    string dateiName = System.IO.Directory.GetCurrentDirectory();
    System.IO.DirectoryInfo di = new System.IO.DirectoryInfo(dateiName);
    string pfd = (di.Parent.Parent.Parent.Parent.FullName + "\\Bsp06_Test.doc");
    wd.Document doc = WordDokumentOeffnen_CS(wdApp, pfd);
    wd.Document docNeu = WordDokumentAnlegen_CS();
    docNeu = null;
    doc = null;
}

private wd.Document WordDokumentOeffnen_CS(wd.Application WdApp, string fileName)
{
    object objMissing = System.Reflection.Missing.Value;
    object objTrue = (object) true;
    object objFileName = (object) fileName;
    wd.Document doc = WdApp.Documents.Open(ref objFileName,
        ref objMissing, ref objMissing, ref objMissing, ref objMissing,
        ref objMissing, ref objMissing, ref objMissing, ref objMissing,
        ref objMissing, ref objMissing, ref objTrue, ref objMissing,
        ref objMissing, ref objMissing, ref objMissing);
    return doc;
}
```

**Listing 6.6** (.NET): Beispiele für die Methoden *Open* und *Add* des *Document*-Objekts in Word 2003 (Fortsetzung)

```
private wd.Document WordDokumentAnlegen_CS()
{
    object objMissing = System.Reflection.Missing.Value;
    wd.Document doc = wdApp.Documents.Add(ref objMissing, ref objMissing,
        ref objMissing, ref objMissing);
    return doc;
}
```

Da es sich um das Hauptobjekt der Anwendung handelt, ist die Liste von Eigenschaften und Methoden für das *Document*-Objekt entsprechend lang. Darin finden Sie unter anderem die dokumentspezifischen Einstellungen aus *Extras/Optionen*, wie etwa *In Formularen nur Daten speichern* (*SaveFormsData*) und *SmartTags einbetten* (*EmbedSmartTags*) aus der Registerkarte *Speichern*. Im Hinblick auf das Herausfinden, welche Eigenschaft zu welcher Option gehört, gelten für diese Optionen die gleichen Bemerkungen wie im Abschnitt »Die Anwendung: Das *Application*-Objekt« in diesem Kapitel.

Im Folgenden befassen wir uns mit einigen Elementen, die für Sie von Interesse sein können, die aber in keinem eigenen Abschnitt oder Kapitel vorgestellt werden.

Compute-  
Statis-  
tics

Um zu ermitteln, wie viele Wörter, Zeichen, Sätze, Zeilen oder Absätze sich in einem geöffneten Word-Dokument befinden, bietet das Objektmodell die Methode *ComputeStatistics*. Dabei kann der Rückgabewert End- und Fußnoten mit einbeziehen oder auch weglassen. Die Enumeration *WdStatistic* (Tabelle 6.5) legt fest, was gezählt werden soll. Die allgemeine Syntax lautet:

```
ComputeStatistics(wdStatistic-Wert, [IncludeFootnotesAndEndnotes As Boolean])
```

**Tabelle 6.5** Die Enumeration von *WdStatistic*

WdStatistic-Enum	Beschreibung
wdStatisticCharacters	Die Anzahl Zeichen, ohne Leerräume
wdStatisticCharactersWithSpace	Die Anzahl Zeichen; Leerräume werden mitgezählt
wdStatisticFarEastCharacters	Die Anzahl Zeichen in einer asiatischen Umgebung
wdStatisticLines	Die Anzahl Zeilen
wdStatisticPages	Die Anzahl Seiten
wdStatisticParagraphs	Die Anzahl Absätze
wdStatisticWords	Die Anzahl Wörter

**HINWEIS**

*ComputeStatistics* gibt einen anderen Wert zurück als die *Count*-Methode einer Auflistung. *ComputeStatistics(wdStatisticParagraphs)* ergibt beispielsweise ein anderes Resultat als *Paragraphs.Count*. Allgemein entspricht *ComputeStatistics* eher dem, was der Anwender erwartet, während *Count* Elemente in der Dokumentstruktur berücksichtigt, die für den Anwender nicht unbedingt wahrnehmbar sind.



Type Gelegentlich ist es wichtig, zu wissen, ob das vorliegende Document-Objekt ein Dokument ist oder eine Vorlage. (Auch eine geöffnete Vorlage ist für das Objektmodell ein Document-Objekt!) Die Type-Eigenschaft liefert diese Information in Form eines Konstantwertes aus Tabelle 6.6.

**Tabelle 6.6** Die Enumeration von *WdDocumentType*

WdDocumentType-Enum	Wert	Beschreibung
wdTypeDocument	0	Dokument
wdTypeTemplate	1	Vorlage
wdTypeFrameset	2	Definiert HTML-Frames

**HINWEIS** Bitte beachten Sie, dass diese Eigenschaft nur lesbar ist. Sie können damit kein Dokument (\*.doc-Datei) in eine Vorlage (\*.dot-Datei) umwandeln oder umgekehrt. Um aus einem Dokument eine Vorlage zu machen, muss es als Vorlage gespeichert werden. Eine Vorlage kann nicht in ein Dokument umgewandelt werden.

Name Gelegentlich müssen wir den Dateinamen, den Pfadnamen oder die vollständige Pfadangabe eines Dokuments herausfinden. Das Word-Objektmodell stellt hierfür die Eigenschaften Name, Path und FullName zur Verfügung. Bitte achten Sie darauf, dass Path **kein** Trennzeichen am Schluss aufweist. So speichern Sie beispielsweise ein zweites Dokument im gleichen Pfad wie das erste:

```
Set doc1 = ActiveDocument
Set doc2 = Documents.Add
doc2.SaveAs doc1.Path & Application.PathSeparator & "Doc2.doc"
```

Attached Ist das Document-Objekt vom Typ Dokument, kann es von Interesse sein, mit welcher Vorlage es verbunden ist. In der Benutzerschnittstelle findet sich diese Angabe im obersten Textfeld des Dialogfelds *Extras/Vorlagen und Add-Ins*. Im Objektmodell gibt die Eigenschaft *AttachedTemplate* diese Information als ein Template-Objekt (dieses Objekt wird im Abschnitt »Dokumentvorlagen: Das Template-Objekt« in diesem Kapitel behandelt) zurück.

Template

Über diese Eigenschaft kann ein Dokument auch mit einer anderen Vorlage verbunden werden, um ihm die darin enthaltenen Symboleleisten, Makros, Tastaturkürzel und AutoText-Einträge zur Verfügung zu stellen. Oder Sie wollen das Dokument vielleicht mit der »neutralen« Vorlage *Normal.dot* verbinden, bevor es außer Haus geschickt wird.

**HINWEIS** Wenn Word die Vorlage im angegebenen Pfad nicht findet, sucht es zunächst im gleichen Ordner, in dem sich das Dokument befindet, danach in den Ordnern der Benutzer- und Arbeitsgruppenvorlagen. Dieser Vorgang kann ziemlich lange dauern, vor allem in Word 2003 ohne »Hot fix« (siehe den Knowledge Base-Artikel <http://support.microsoft.com/kb/823372/de>). Wird die Vorlage dennoch nicht gefunden, wird eine temporäre Verbindung zur *Normal.dot* hergestellt, was *AttachedTemplate* widerspiegelt. Aber aufgepasst! In *Extras/Vorlagen und Add-Ins* erscheint immer noch die Pfadangabe zur ursprünglichen Vorlage. Dieser Umstand könnte Außenstehenden wichtige Informationen über die Ordnerstrukturen in Ihrer Firma liefern.

Die Prozedur in Listing 6.7 (C#-Version in Listing 6.8) kontrolliert, ob es sich bei der geöffneten Word-Datei um ein Dokument oder um eine Vorlage handelt. Im Falle eines Dokuments wird dieses

mit der Vorlage *Normal.dot* verbunden und gespeichert. Liegt eine Vorlage vor, wird der Benutzer gewarnt, dass daraus ein neues Dokument erstellt wird und dass er dieses speichern soll. Der Pfadname der Vorlage wird ermittelt und eingesetzt, um das neue Dokument zu erstellen, das nun mit der *Normal.dot* verbunden wird. Beim Speichern dieses neuen Dokuments erscheint automatisch das Dialogfeld *Speichern unter*, da es noch nie gespeichert wurde.

**Listing 6.7** Ein Dokument für die Übermittlung außer Haus vorbereiten und mit der *Normal.dot* verbinden

```
Sub DokFuerTransportVorbereiten()
    Dim doc As Word.Document
    Dim strTemplatePath As String
    Dim docNeu As Word.Document

    Set doc = Application.ActiveDocument
    If doc.Type = wdTypeDocument Then
        DokMitNormalVerbinden doc
    ElseIf doc.Type = wdTypeTemplate Then
        MsgBox "Das aktuelle Dokument ist eine Vorlage, und darf nicht außer Haus " & _
            "geschickt werden. Ein neues Dokument wird daraus erstellt. " & _
            "Bitte speichern und verschicken Sie dieses.", vbOKOnly + vbInformation
        strTemplatePath = doc.FullName
        Set docNeu = Application.Documents.Add(strTemplatePath)
        DokMitNormalVerbinden docNeu
    End If
End Sub

Sub DokMitNormalVerbinden(doc As Word.Document)
    doc.AttachedTemplate = NormalTemplate
    'Wenn der Benutzer Speichern unter abbricht,
    'soll die Ausführung nicht abgebrochen werden.
    On Error Resume Next
    doc.Save
End Sub
```

**Listing 6.8** (.NET): C#-Version, um ein Dokument mit der Vorlage *Normal.dot* zu verbinden

```
private void btnAttachedTemplate_Click(object sender, System.EventArgs e)
{
    wd.Document doc = wdApp.ActiveDocument;
    wd.WdDocumentType docType = doc.Type;
    if (docType == wd.WdDocumentType.wdTypeDocument)
    {
        DokMitNormalVerbinden_CS(doc);
    }
    else if (docType == wd.WdDocumentType.wdTypeTemplate)
    {
        object objMissing = System.Reflection.Missing.Value;
        string msgNotDocument = "Das aktuelle Dokument ist eine Vorlage, " +
            "und darf nicht außer Haus geschickt werden. " +
            "Ein neues Dokument wird daraus erstellt. " +
            "Bitte speichern und verschicken Sie dieses.";
        MessageBox.Show(msgNotDocument, "", MessageBoxButtons.OK);
        object templatePath = (object) doc.FullName;
        wd.Document docNeu = wdApp.Documents.Add(ref (object) templatePath, ref objMissing,
            ref objMissing, ref objMissing);
        DokMitNormalVerbinden_CS(docNeu);
    }
}
```

Listing 6.8 (.NET): C#-Version, um ein Dokument mit der Vorlage *Normal.dot* zu verbinden (Fortsetzung)

```

    }
}

private void DokMitNormalVerbinden_CS(wd.Document doc)
{
    object objNormalTemplate = (object) wdApp.NormalTemplate;
    doc.set_AttachedTemplate(ref objNormalTemplate);
    try
    {doc.Save();}
    catch {}
}

```

Save  
SaveAs  
Saved

Die IntelliSense-Liste für das Document-Objekt enthält drei Einträge mit dem Ausdruck »Save« im Elementnamen: SaveAs (speichern unter), Save (speichern) und Saved (ist gespeichert).

Mit SaveAs wird das Dokument unter dem im Argument FileName angegebenen Pfadnamen gespeichert. Eigentlich bedarf die Handhabung von SaveAs keiner näheren Erklärung. Es gibt jedoch einige Argumente, worauf wir aufmerksam machen möchten:

- Um das Dokument als eine andere Dateiarart zu speichern – beispielsweise als Vorlage, Webseite oder Textdatei – legen Sie ein SaveFormat fest.
- Wird das Dokument als Textdatei gespeichert, gibt es einige hilfreiche Argumente, die bestimmen, wie der Text auszugeben ist: Encoding, InsertLineBreaks, AllowSubstitutions und LineEnding.
- Die verschiedenen Kennwortoptionen funktionieren nicht immer zuverlässig über die *Speichern unter*-Schnittstelle. Kennwörter sollten daher besser direkt über die entsprechenden Eigenschaften (Password, WritePassword, ReadOnlyRecommended) des Document-Objekts festgelegt werden, bevor das Dokument gespeichert wird.

Mit der Save-Methode wird das Dokument unter dem bestehenden Pfadnamen gespeichert. Wurde es noch nie gespeichert, leitet Word automatisch *Datei/Speichern unter* ein und zeigt das entsprechende Dialogfeld an, so dass der Benutzer einen Dateinamen eingeben kann. Bricht der Benutzer das Dialogfeld ab, wird ein Fehler verursacht.

Eine Möglichkeit, diesen Fehler zu umgehen, wurde in Listing 6.7 verwendet: Fehlermeldungen werden mit On Error Resume Next einfach ausgeschaltet (mehr über die Fehlerbehandlung lesen Sie in Kapitel 2). Ein Problem dieser Herangehensweise ist, dass keine Kontrolle besteht, ob das Dokument jemals gespeichert wurde.

Um dies zu gewährleisten, können wir uns der Saved-Eigenschaft bedienen. Wurde das Dokument seit der letzten Bearbeitung nicht gespeichert, gibt Saved »falsch« zurück. Der Benutzer wird so lange aufgefordert, das Dokument zu speichern, bis er es getan hat. Ein Beispiel hierfür sehen Sie in Listing 6.9 bzw. Listing 6.10. Da ein neues Dokument nicht immer Änderungen enthält, die Word veranlassen, *Speichern unter* einzuleiten, wird die Saved-Eigenschaft nach Erstellung des neuen Dokuments auf False gesetzt.

Listing 6.9 Die Eigenschaft Saved prüft den Bearbeitungszustand eines Dokuments oder legt ihn fest

```

Sub DokSpeichern()
    Dim doc As Word.Document

```

**Listing 6.9** Die Eigenschaft *Saved* prüft den Bearbeitungszustand eines Dokuments oder legt ihn fest (Fortsetzung)

```
Set doc = Documents.Add
doc.Saved = False
Do While Not doc.Saved
    On Error Resume Next
    doc.Save
    On Error GoTo 0
Loop
End Sub
```

**Listing 6.10** (.NET): Die *Saved*-Eigenschaft in C# wirft keine besonderen Probleme auf

```
private void DokSpeichern()
{
    object objMissing = System.Reflection.Missing.Value;
    wd.Document doc = wdApp.Documents.Add(ref objMissing, ref
        ref objMissing, ref objMissing);
    doc.Saved = false;
    while (!doc.Saved)
    {
        try
        {doc.Save();}
        catch {}
    }
}
```

#### PROFITIPP

Eine Alternative zu diesem Vorgang wäre, das Dialogfeld *Datei/Speichern unter* explizit durch den Code einzublenken. Dadurch kann die Benutzerhandlung direkt ausgewertet werden. Der Umgang mit den internen Dialogfeldern von Word ist in Kapitel 14 beschrieben.

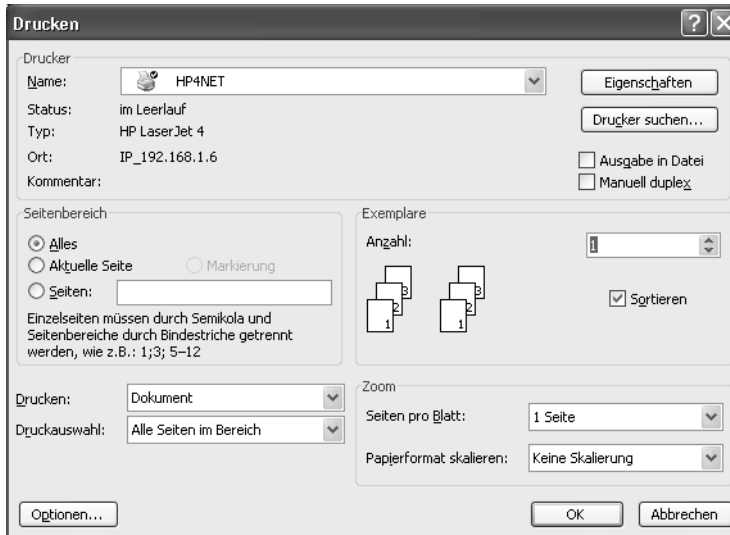
#### PrintOut

Die *PrintOut*-Methode stellt den Großteil der Funktionalität des Dialogfelds *Datei/Drucken* (Abbildung 6.5) zur Verfügung. Da die Zusammenhänge der verschiedenen Argumente gelegentlich Fragen aufwerfen, gehen wir hier etwas näher auf sie ein. Die allgemeine Syntax der Methode lautet:

```
PrintOut(Background, Append, Range, OutputFileName, From, To, Item, Copies, Pages, _
    PageType, PrintToFile, Collate, FileName, ActivePrinterMacGX, ManualDuplexPrint, _
    PrintZoomColumn, PrintZoomRow, PrintZoomPaperWidth, PrintZoomPaperHeight)
```

**Background.** Word hat zwei Druckmodi. Entweder muss der Benutzer warten, bis ein Druckauftrag vollständig an den Drucker gesandt wird, oder der Druckauftrag läuft im Hintergrund, während der Benutzer weiterarbeitet. Vorausgesetzt, das Drucken im Hintergrund verursacht keine Probleme (die korrekte Aktualisierung der Seitenzahlen etwa), ist diese letzte Einstellung (Menübefehl *Extras/Optionen/Drucken*) dem Benutzer natürlich lieber.

Für den Entwickler ist die Lage eher umgekehrt. Sendet sein Code einen Druckauftrag und soll die Ausführung erst weiterlaufen, nachdem der Auftrag erledigt ist, ist es wichtig, dass **nicht** im Hintergrund gedruckt wird. Im Objektmodell kann die Einstellung theoretisch an zwei verschiedenen Stellen vorgenommen werden: als Argument der *PrintOut*-Methode oder als Anwendungsoption *Application.Options.PrintBackground*. Nach Erfahrung der Autoren funktioniert nur die letztere zuverlässig.

Abbildg. 6.5 Das Dialogfeld *Datei/Drucken*

Hinzu kommt, dass Druckaufträge im Hintergrund asynchron bearbeitet werden. Schickt Ihr Code mehrere Dokumente an den Drucker, ist beim Drucken im Hintergrund nicht gewährleistet, dass sie in der erwarteten Reihenfolge gedruckt werden. Nehmen wir als Beispiel eine Anwendung, die Formulare erstellt, durchnummeriert und ausdruckt. Startnummer und Anzahl legt der Benutzer während der Ausführung fest. Würden diese Formulare asynchron gedruckt, müsste die Reihenfolge am Schluss kontrolliert und allenfalls von Hand nachsortiert werden.

Auf jedem Fall raten wir dringlichst, den Druck im Hintergrund während des Code-Ablaufs zu unterbinden, indem Sie das Background-Argument oder die Option `PrintBackground` auf `False` festlegen.

In Listing 6.11 bzw. in Listing 6.12 wird gezeigt, wie diese Option anzuwenden ist. Der Druckauftrag muss beendet werden, bevor das Dokument geschlossen wird. Beachten Sie, wie die ursprüngliche Einstellung in einer Variablen gespeichert und am Schluss wieder hergestellt wird.

#### HINWEIS

In einer neueren Version von Word wurde die Application-Eigenschaft `BackgroundPrintingStatus` eingeführt. Sie gibt die Anzahl anstehender Druckaufträge zurück und wird eingesetzt, um ausstehende Druckaufträge zu ermitteln, bevor die Word-Anwendung beendet wird. Da sie aber keine Informationen zu den einzelnen Aufträgen liefert, kann sie nicht verwendet werden, um festzustellen, ob ein bestimmtes Dokument schon gedruckt wurde.

**Listing 6.11** Drucken im Hintergrund unterbinden. Die Anwender-Einstellung wird am Schluss wieder hergestellt.

```
Sub DokAusdrucken()
    Dim doc As Word.Document
    Dim lDruckImHintergrund As Long

    lDruckImHintergrund = Application.Options.PrintBackground
    Application.Options.PrintBackground = False
```

**Listing 6.11** Drucken im Hintergrund unterbinden. Die Anwender-Einstellung wird am Schluss wieder hergestellt. (Fortsetzung)

```
Set doc = ActiveDocument
doc.PrintOut
Application.Options.PrintBackground = 1DruckImHintergrund
doc.Close SaveChanges:=wdSaveChanges
End Sub
```

**Listing 6.12** (.NET): Der C#-Code für das Unterbinden des Druckens im Hintergrund

```
private void DokAusdrucken_CS(wd.Document doc)
{
    object objMissing = System.Reflection.Missing.Value;
    bool druckImHintergrund = wdApp.Options.PrintBackground;
    wdApp.Options.PrintBackground = false;
    DokDrucken(doc)
    object objSaveChanges = wd.WdSaveOptions.wdSaveChanges;
    wdApp.Options.PrintBackground= druckImHintergrund;
    doc.Close(ref objSaveChanges, ref objMissing, ref objMissing);
}
private void DokDrucken(wd.Document doc)
{
    object objMissing = System.Reflection.Missing.Value;
    doc.PrintOut(ref objMissing, ref objMissing, ref objMissing, ref objMissing,
        ref objMissing, ref objMissing, ref objMissing, ref objMissing,
        ref objMissing, ref objMissing, ref objMissing, ref objMissing,
        ref objMissing, ref objMissing);
}
```

**Range, From, To, Pages.** Diese Argumente beziehen sich auf den Abschnitt *Seitenbereich* des Dialogfelds *Drucken*. Ausschlaggebend, ob From und To oder Pages zu verwenden ist, ist die Einstellung für Range, dessen mögliche Werte in Tabelle 6.7 aufgelistet sind.

**Tabelle 6.7** Die *WdPrintOutRange*-Werte

WdPrintOutRange-Enum	Wert	Beschreibung
wdPrintAllDocument	0	Das gesamte Dokument; entspricht der Option <i>Alles</i> im Dialogfeld.
wdPrintCurrentPage	2	Die aktuelle Seite.
wdPrintFromTo	3	Die angegebenen Seiten von <i>n</i> bis <i>m</i> drucken.
wdPrintRangeOfPages	4	Alle angegebenen Seiten drucken; entspricht dem Feld <i>Seiten</i> im Dialogfeld.
wdPrintSelection	1	Den markierten Text ausdrucken.

Wird Range auf wdPrintFromTo gesetzt, müssen den Argumenten From sowie To ganze Zahlen zugewiesen werden, die den Seitenzahlen im gegenwärtigen Dokument entsprechen. From und To werden ignoriert, wenn Range einen anderen Wert beträgt.

Das Argument Pages kommt nur zum Zug, wenn Range auf wdPrintRangeOfPages gesetzt wird. Es erwartet eine Zeichenkette, wie sie ins Feld *Seiten* einzugeben ist. Mehr Informationen zu diesem

Thema finden Sie in den Word-Hilfedateien unter »Drucken eines Dokuments« im Abschnitt »Drucken bestimmter Seiten und Abschnitte«.

Ein Beispiel, wie die erste Seite jedes Dokumentabschnitts gedruckt wird, sehen Sie in Listing 6.13 bzw. Listing 6.14. Das Resultat für ein Dokument mit drei Abschnitten könnte so aussehen: "P1/S1;P1/S2;P1/S3". Beachten Sie, wie das Trennzeichen (ein Semikolon) am Ende jeder For...Next-Schleife der Zeichenkette, worin wir die Bereichbestimmung aufbauen, hinzugefügt wird, außer beim letzten Mal.

**Listing 6.13** Als Druckbereich die erste Seite jedes Abschnitts festlegen

```
Sub DokErsteSeiteJedesAbschnitts()
    Dim lAbschnittZaehler As Long
    Dim lAnzahlAbschnitte As Long
    Dim doc As Word.Document
    Dim strDruckbereich As String

    Set doc = ActiveDocument
    lAnzahlAbschnitte = doc.Sections.Count
    For lAbschnittZaehler = 1 To lAnzahlAbschnitte
        strDruckbereich = strDruckbereich & "P1/S" & Trim(CStr(lAbschnittZaehler))
        If lAbschnittZaehler <> lAnzahlAbschnitte Then
            strDruckbereich = strDruckbereich & ";"
        End If
    Next
    doc.PrintOut Range:=wdPrintRangeOfPages, Pages:=strDruckbereich
End Sub
```

#### TIPP

Beachten Sie, wie in Listing 6.14 die PrintOut-Methode in eine getrennte Prozedur ausgelagert wird, und vergleichen Sie *DokDrucken* mit der Prozedur gleichen Namens in Listing 6.12. In C# dürfen mehrere Prozeduren gleichen Namens vorhanden sein, solange diese verschiedene »Signatures« haben (unterschiedliche Argumente oder Rückgabewert). Dies wird »overloading« genannt. .NET erkennt automatisch, welche Prozedur gemeint ist; so kommt C# ohne optionale Argumente aus. Wenn Sie häufig eine COM-Methode mit vielen Argumenten aufrufen müssen, lohnt es sich, eine solche »Bibliothek« anzulegen.

**Listing 6.14** (.NET): Die C#-Version von Listing 6.13

```
private void DokErsteSeiteJedesAbschnitts_CS()
{
    string druckbereich=null;
    wd.Document doc = wdApp.ActiveDocument;
    int anzahlAbschnitte = doc.Sections.Count;
    for(int abschnittZaehler = 1; abschnittZaehler<=anzahlAbschnitte; ++abschnittZaehler)
    {
        druckbereich += "P1S" + abschnittZaehler.ToString();
        if (abschnittZaehler != anzahlAbschnitte)
        {
            druckbereich += ";";
        }
    }
    object objMissing = System.Reflection.Missing.Value;
    object objPrintRange = (object) wd.WdPrintOutRange.wdPrintRangeOfPages;
    object objPages = (object) druckbereich;
```

**Listing 6.14** (.NET): Die C#-Version von Listing 6.13 (Fortsetzung)

```

    DokDrucken(doc, objPrintRange, objPages);
}
private void DokDrucken(wd.Document doc, string rangePages, wd.WdPrintOutRange _
    printOutRange)
{
    object objRange = (object) printOutRange;
    object objPages = (object) rangePages;
    object objMissing = System.Reflection.Missing.Value;
    doc.PrintOut(ref objMissing, ref objMissing, ref objRange, ref objMissing,
        ref objMissing, ref objMissing, ref objMissing, ref objMissing,
        ref objPages, ref objMissing, ref objMissing, ref objMissing,
        ref objMissing, ref objMissing, ref objMissing, ref objMissing,
        ref objMissing, ref objMissing);
}

```

**HINWEIS**

Denken Sie daran, dass die Seitennummerangaben im Pages-Argument den Seitenzahlen im Dokument entsprechen, wie sie ausgedruckt werden. Dies ist vor allem wichtig, wenn das Dokument mehrere Abschnitte enthält. Wird durchnummeriert, hat die Verwendung von Seitenzahlen zusammen mit Abschnittsnummern keine Bedeutung. Dies ist nur sinnvoll, wenn in jedem Abschnitt die Nummerierung neu gestartet wird.

**Item.** Unterhalb des Kastens *Seitenbereich* befinden sich zwei Dropdown-Felder, die oft nicht beachtet werden. Im ersten, das dem Argument *Item* entspricht, wird bestimmt, was anstelle des Dokuments auszudrucken ist: AutoText-Einträge, Änderungen und Kommentare, Tastaturbelegungen oder Formatvorlagen. Die gültigen Werte sind in Tabelle 6.8 aufgelistet.

**Tabelle 6.8** Werte für das Argument *Item*

WdPrintOutItem-Enum	Wert	Beschreibung
wdPrintAutoTextEntries	4	Druckt die in der verbundenen Vorlage gespeicherten AutoText-Einträge.
wdPrintComments	2	Druckt alle im Dokument vorhandenen Kommentare.
wdPrintDocumentContent	0	Druckt das Dokument aus.
wdPrintDocumentWithMarkup	7	Druckt den Dokumenttext mit Änderungen und Kommentaren in Sprechblasen im Rand (Word 2002, 2003).
wdPrintEnvelope	6	Druckt einen am Dokumentanfang angefügten Umschlag (erstellt über <i>Extras/Briefe und Sendungen/Umschläge und Etiketten</i> ).
wdPrintKeyAssignments	5	Druckt die im Dokument sowie in der verbundenen Vorlage gespeicherten Tastenbelegungen aus.
wdPrintMarkup	2	Druckt alle Änderungen und Kommentare in einer Liste aus, wie sie im Überarbeitungsfenster erscheinen (Word 2002, 2003).
wdPrintProperties	1	Druckt die im Dokument gespeicherten Dokumenteigenschaften.
wdPrintStyles	3	Druckt die im Dokument verwendeten Formatvorlagen.



**PageType.** Im zweiten Dropdown-Feld, das dem Argument `PageType` entspricht, wird bestimmt, ob alle Dokumentseiten (`wdPrintAllPages`), nur die geraden Seiten (`wdPrintEvenPagesOnly`) oder nur die ungeraden Seiten (`wdPrintOddPagesOnly`) auszudrucken sind. Diese Einstellung ist nützlich, wenn kein Duplex-Drucker verfügbar ist und trotzdem doppelseitig gedruckt werden soll.

**Append, OutputFilename, PrintToFile.** Im oberen Teil des Dialogfelds *Drucken* befindet sich das Kontrollkästchen *Ausgabe in Datei*. Für ein Word-Dokument wird es selten benutzt, wohl aber für die Erstellung von PDF-Dateien aus Word-Dokumenten. Es ist auch nützlich, wenn der Anwender eine Liste Formatvorlagen, AutoText-Einträge oder Tastaturbelegungen – was mit dem Argument `Item` bestimmt wird – bearbeiten möchte. Eine mit dieser Funktionalität erstellte Text-Datei kann in Word geöffnet werden.

Zuerst muss sichergestellt werden, dass die richtige Druckerart ausgewählt ist. Für eine PDF-Datei wäre dies ein Druckertreiber wie »Adobe PDF-Writer«. Für das Erstellen einer Text-Datei steht im Windows-Lieferumfang immer ein generischer Textdrucker zur Verfügung. Um ihn zu installieren, gehen Sie folgendermaßen vor:

1. Wählen Sie im Startmenü von Windows den Eintrag *Drucker und Faxgeräte*.
2. Führen Sie den Befehl *Drucker hinzufügen* aus.
3. Klicken Sie auf *Weiter*, bis das Dialogfeld *Drucker installieren* erscheint.
4. Aus der Liste *Hersteller* wählen Sie den Eintrag *Standard*.
5. Markieren Sie in der Liste *Drucker* den Eintrag *Generic / Text Only*.
6. Klicken Sie auf *Weiter*, bis die Schaltfläche *Fertig stellen* erscheint, und klicken Sie auf diese.

Das Listing 6.15 bzw. das Listing 6.16 enthält ein Code-Beispiel, das die AutoText-Einträge der aktuellen Umgebung in eine Textdatei exportiert.

**Listing 6.15** Ein Word-Dokument in eine Datei ausgeben, anstatt an den Drucker

```
'Haben Sie dem "TextDrucker" einen anderen Namen gegeben,
'muss der Code entsprechend angepasst werden.
Sub AutoTextListeErstellen()
    Dim doc As Word.Document
    Dim strPfad As String
    Dim strAktuellerDrucker

    Set doc = ActiveDocument
    strPfad = "C:\test\TastaturBelegung_" & doc.Name & Format(Date, "mmdd") _
        & "_" & Format(Time, "hh.ss") & ".prn"
    strAktuellerDrucker = Application.ActivePrinter
    Application.ActivePrinter = "TextDrucker"
    Application.Options.PrintBackground = False
    doc.PrintOut Item:=wdPrintAutoTextEntries, Append:=False, _
        OutputFileName:=strPfad, PrintToFile:=True
    Application.ActivePrinter = strAktuellerDrucker
    Application.Documents.Open strPfad
End Sub
```

**Listing 6.16** (NET): Aus Platzgründen wird die overloaded *DokDrucken*-Prozedur hier nicht nochmals aufgeführt

```
private void AutoTextListeErstellen_CS()
{
    wd.Document doc = wdApp.ActiveDocument;
    System.DateTime dt = System.DateTime.Now;
    string pfd = String.Format("C:\\test\\TastaturBelegung_{0}{1}_{2}.prn",
        doc.Name, dt.ToString("MMMdd"), dt.ToString("hh.ss"));
    string aktuellerDrucker = wdApp.ActivePrinter;
    wdApp.ActivePrinter = "TextDrucker";
    wdApp.Options.PrintBackground = false;
    bool append = false;
    wd.WdPrintOutItem printOutItem = wd.WdPrintOutItem.wdPrintAutoTextEntries;
    bool printToFile = true;
    DokDrucken(doc, printOutItem, append, pfd, printToFile);
    wdApp.ActivePrinter = aktuellerDrucker;
    WordDokumentOeffnen_CS(wdApp, pfd);
}
```

**HINWEIS**

Im Objektmodell befinden sich so gut wie keine Schnittstellen für die im Dialogfeld *Drucken* enthaltenen Drucker-Optionen und -Einstellungen. Einzig die Eigenschaft `Application.ActivePrinter` kann den ausgewählten Zieldrucker ändern. Dafür braucht Sie den genauen Druckernamen, wie er auf dem Rechner definiert ist. Alle weiteren Einstellungen müssen über die Windows-API-Schnittstelle stattfinden. Weitere Informationen hierzu finden Sie im Artikel »Controlling the Printer from Word VBA« <http://pubs.logicalexpressions.com/Pub0009/LPMArticle.asp?ID=101>.



Die Beispieldatei *Bsp06\_01\_Document.doc* finden Sie auf der CD-ROM im Ordner *\Kap06*.

## Dokumentvorlagen: Das *Template*-Objekt

Dokumentvorlagen sind ein wichtiger Bestandteil von Word. Erstens dienen sie als Schablone für neue Dokumente eines bestimmten Typs. Sie enthalten beispielsweise Briefkopf, Logo, Kopf- und Fußzeilen sowie Standardtexte. Ferner stellt eine gute Dokumentvorlage eine Reihe von Formatvorlagen zur Verfügung, um eine einheitliche Formatierung sämtlicher Dokumente dieses Typs zu gewährleisten. Dieser gesamte Inhalt wird an jedes von einer Vorlage neu erstellte Dokument weitergegeben.

Programmtechnisch wird ein neues Dokument durch die `Documents.Add`-Methode erstellt, die im Abschnitt »Der Kern der Sache: Das *Document*-Objekt« in diesem Kapitel vorgestellt wurde. Um dazu eine bestimmte Vorlage zu verwenden, übergeben Sie deren Pfadnamen als `FileName`-Argument.

**HINWEIS**

Nach dem Erstellen eines Dokuments aus einer Vorlage besteht für die erwähnten Inhalte keine Verbindung mehr zur Vorlage. Es ist also nicht möglich, die Kopfzeile in der Vorlage zu ändern, so dass die gleiche Änderung in allen verbundenen Dokumenten vorgenommen wird. Die einzige Ausnahme bilden Formatvorlagen. Falls in *Extras/Vorlagen und Add-Ins* das Kontrollkästchen *Dokumentformatvorlagen automatisch aktualisieren* (Abbildung 6.6) aktiviert ist, werden bei jedem Öffnen des Dokuments die Formatvorlagendefinitionen mit denen aus der Vorlage überschrieben. Dies ist jedoch mit der Verwendung der automatischen Nummerierung in Word meistens nicht vereinbar.

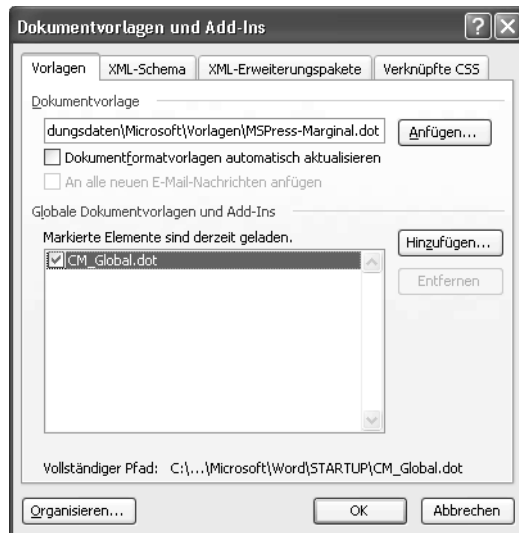
In zweiter Linie dienen Vorlagen als Behälter nützlicher Werkzeuge, die eine beliebige Kombination von Makros, Symbolleisten, AutoText-Einträgen und Tastaturbelegungen umfassen dürfen. In dieser Funktion gibt es zwei Arten von Vorlagen: die dokumentspezifische und das globale Add-In.

Die Werkzeuge einer dokumentspezifischen Vorlage sind nur für die damit verbundenen Dokumente sicht- und abrufbar.

Die Werkzeuge einer globalen Vorlage stehen jedem in der Word-Anwendung geöffneten Dokument zur Verfügung.

Von Word immer geladen wird eine globale Vorlage: die *Normal.dot*. Weitere Vorlagen können automatisch beim Starten von Word geladen werden, wenn sie sich im zugewiesenen *Startup*-Ordner befinden (siehe auch Kapitel 1). Zusätzlich können jederzeit weitere Vorlagen über *Extras/Vorlagen und Add-Ins* (Abbildung 6.6) geladen werden, indem man sie in der Liste aktiviert (oder hinzufügt und aktiviert).

Abbildg. 6.6 Dokumentspezifische und globale Add-Ins werden über dieses Dialogfeld verwaltet

**HINWEIS**

Add-Ins werden in Kapitel 13 eingehender behandelt.

Um programmtechnisch auf die dokumentspezifische Vorlage zuzugreifen, benutzen wir die Eigenschaft `AttachedTemplate` des `Document`-Objekts. Diese wurde im Abschnitt »Der Kern der Sache: Das *Document*-Objekt« in diesem Kapitel mit einem Beispiel vorgestellt.

Für die *Normal.dot* gibt es sogar ein eigenes Objekt: `NormalTemplate`. Als Beispiel dafür werden mit Listing 6.17 alle `AutoText`-Einträge der *Normal.dot* in einem neuen Dokument aufgelistet. Beachten Sie hierbei auch, wie das `Range`-Objekt eingesetzt wird. Näheres dazu steht im Abschnitt »Mit Bereichen arbeiten: Das *Range*-Objekt« in diesem Kapitel beschrieben.

**Listing 6.17** `AutoText`-Einträge der *Normal.dot* in neuem Dokument auflisten

```
Sub AlleAutoTextEintraegeAuflisten()
    Dim doc As Word.Document
    Dim rng As Word.Range
    Dim at As Word.AutoTextEntry

    Set doc = Documents.Add
    Set rng = doc.Content
    For Each at In NormalTemplate.AutoTextEntries
        rng.Text = at.Name & vbCr
        rng.Collapse Direction:=wdCollapseEnd
        Set rng = at.Insert(Where:=rng, RichText:=True)
        rng.InsertAfter vbCr & vbCr
        rng.Collapse Direction:=wdCollapseEnd
    Next
End Sub
```

**Listing 6.18** (.NET): Die C#-Version

```
private void btnListing6_18_Click(object sender, System.EventArgs e)
{
    object objCollapseEnd = (object) wd.WdCollapseDirection.wdCollapseEnd;
    object objTrue = (object) true;
    object objMissing = System.Reflection.Missing.Value;
    wd.Document doc = wdApp.Documents.Add(ref objMissing, ref objMissing,
        ref objMissing, ref objMissing);
    wd.Range rng = doc.Content;
    foreach (wd.AutoTextEntry at in wdApp.NormalTemplate.AutoTextEntries)
    {
        rng.Text = at.Name + "\n";
        rng.Collapse(ref objCollapseEnd);
        rng = at.Insert(rng, ref objTrue);
        rng.InsertAfter("\n\n");
        rng.Collapse(ref objCollapseEnd);
    }
    MessageBox.Show("Fertig!");
}
```

Um andere globale Vorlagen im Code anzusprechen, wird entweder der ganze Pfadname der Vorlage benötigt oder es muss durch die `Templates`-Auflistung geschleift werden. Das Listing 6.19 enthält hierzu ein Beispiel, das ein in der globalen Vorlage gespeichertes Makro ausführt.

**HINWEIS** Falls Sie den Pfadnamen zu einem der Vorlagenordner (Startup, Benutzervorlagen oder Arbeitsgruppenvorlagen) benötigen, gibt `Application.Options.DefaultFilePath` (siehe auch den Abschnitt »Die Anwendung: Das *Application*-Objekt« in diesem Kapitel) die Information zurück.

**Listing 6.19** Ist eine Vorlage nicht vorhanden, kann sie in den Speicher als Add-In geladen werden

```
Sub MakroInAddinAusführen()
    Dim tpl As Word.Template
    Dim strAddinName As String
    Dim strMakroName As String
    Dim bAddinGeladen As Boolean
    Dim adin As Word.AddIn

    strAddinName = ThisDocument.Path & "\Bsp06_02_Template.dot"
    strMakroName = "MakroInAddin"
    bAddinGeladen = False
    For Each tpl In Application.Templates
        If tpl.Name = strAddinName Then
            bAddinGeladen = True
            Application.Run strMakroName
        End If
    Next
    If Not bAddinGeladen Then
        'Die Vorlage als Add-In laden
        Set adin = Application.AddIns.Add(
            FileName:=strAddinName, Install:=True)
        Application.Run strMakroName
        'Und wieder aus dem Speicher entfernen
        adin.Delete
    End If
End Sub
```

**Listing 6.20** (.NET): Die C#-Version

```
private void Listing6_20_Click(object sender, System.EventArgs e)
{
    object objTrue = (object) true;
    string dateiName = System.IO.Directory.GetCurrentDirectory();
    System.IO.DirectoryInfo di = new System.IO.DirectoryInfo(dateiName);
    string pfad = (di.Parent.Parent.Parent.Parent.FullName + "\\Bsp06_02_Template.dot");
    string addinName = pfad;
    string makroName = "MakroInAddin";
    bool addinGeladen = false;
    foreach (wd.Template tpl in wdApp.Templates)
    {
        if (tpl.Name == addinName)
        {
            addinGeladen = true;
            RunWordMakro(makroName);
        }
    }
    if (!addinGeladen)
    {
        //Die Vorlage als Add-In laden
    }
}
```

**Listing 6.20** (.NET): Die C#-Version (*Fortsetzung*)

```

        wd.AddIn adin = wdApp.AddIns.Add(addinName, ref objTrue);
        RunWordMakro(makroName);
        //Und wieder aus dem Speicher entfernen
        adin.Delete();
    }
}

private void RunWordMakro(string makroName)
{
    object objMissing = System.Reflection.Missing.Value;
    wdApp.Run(makroName, ref objMissing, ref objMissing, ref objMissing,
        ref objMissing, ref objMissing, ref objMissing, ref objMissing,
        ref objMissing, ref objMissing, ref objMissing, ref objMissing,
        ref objMissing, ref objMissing, ref objMissing, ref objMissing,
        ref objMissing, ref objMissing, ref objMissing, ref objMissing,
        ref objMissing, ref objMissing, ref objMissing, ref objMissing);
}

```

Type Beim Schleifen durch die Templates-Auflistung können Sie über die Type-Eigenschaft feststellen, welche Art von Vorlage vorliegt. Die möglichen Werte sind in Tabelle 6.9 aufgelistet.

**Tabelle 6.9** Die Werte für *Template.Type*

WdTemplateType-Enum	Wert	Beschreibung
wdAttachedTemplate	2	dokumentspezifische Vorlage
wdGlobalTemplate	1	als globales Add-In geladene Vorlage
wdNormalTemplate	0	die Vorlage <i>Normal.dot</i>

Das Word-Objektmodell enthält sonst nur wenige Eigenschaften für das Template-Objekt, hauptsächlich die Dokumenteigenschaften (DocumentProperties) und die AutoText-Einträge. Über das Document-Objekt kann zusätzlich auf Makros (mit der Run-Methode), Symbolleisten (über die CommandBars-Auflistung) und Tastaturbelegungen (über die KeyBindings-Auflistung) in einer Vorlage zugegriffen werden.

OpenAs- Für alles andere – um etwa auf die Formatvorlagen zuzugreifen – müssen Sie die Vorlage als Dokument öffnen. Dafür steht die Methode OpenAsDocument bereit. Bitte beachten Sie, dass die Datei *in der Benutzerschnittstelle* geöffnet wird.

**WICHTIG**

Die OpenAsDocument-Methode bietet kein Argument, um die Vorlage unsichtbar zu öffnen oder sonst irgendwie zu schützen. Wenn Sie nicht wollen, dass der Benutzer darauf zugreift, muss nach dem Öffnen das Dokumentfenster unsichtbar gemacht werden. Es wird jedoch immer noch im Menü *Fenster* erscheinen und ansprechbar sein. Sie können nötigenfalls mit Hilfe von Anwendungsereignissen den Zugriff verweigern (mehr zum Thema Ereignisse steht in Kapitel 7 beschrieben).

**Listing 6.21** Eine geladene Vorlage als Dokument öffnen, um sie zu bearbeiten

```
Sub NormalDotOeffnen()
    Dim doc As Word.Document

    Set doc = NormalTemplate.OpenAsDocument
    doc.ActiveWindow.Visible = False
    doc.Close SaveChanges:=wdDoNotSaveChanges
End Sub
```

**Listing 6.22** (.NET): Die C#-Version

```
private void Listing6_22_Click(object sender, System.EventArgs e)
{
    wd.Document doc = wdApp.NormalTemplate.OpenAsDocument();
    doc.ActiveWindow.Visible = false;
    object objNoSaveChanges = (object) wd.WdSaveOptions.wdDoNotSaveChanges;
    object objMissing = System.Reflection.Missing.Value;
    doc.Close(ref objNoSaveChanges, ref objMissing, ref objMissing);
}
```



Die Beispieldatei *Bsp06\_01\_Template.doc* mit den Code-Beispielen sowie *Bsp06\_02\_Template.dot* mit dem Makro »MakroAddin« finden Sie auf der CD-ROM zum Buch im Ordner \Beispiele\Kap06.

## Mit Bereichen arbeiten: Das *Range*-Objekt

Was das Document-Objekt für die Word-Anwendung ist, ist das Range-Objekt für das Dokument. Ein Range ist ein zusammenhängender, fortlaufender Textbereich im Dokument, der alle in diesem Bereich enthaltenen Zeichen umfasst. Er kann unsichtbare Zeichen, z.B. Text mit verborgener Formatierung sowie Feldklammern und Feldcodes enthalten.

Jeder Range hat einen Start- und einen Endpunkt, die jeweils Zeichen im Textfluss sind. Befinden sich Start- und Endpunkt am gleichen Ort, sagen wir, dass der Bereich auf einen Punkt verkleinert ist. Am einfachsten zu begreifen ist das Konzept, wenn Sie sich eine virtuelle Markierung im Dokument vorstellen, die unabhängig von der sichtbaren Markierung manipuliert werden kann. Es ist möglich, im Code mehrere Range-Bereiche gleichzeitig zu definieren und damit zu arbeiten.

Die Zeichen in einem Dokumentteil (Story) werden intern von 1 bis *n* durchnummeriert, und es ist möglich, einen Range anhand von Start- und Endpunktwerten zu definieren. Da der Inhalt eines Word-Dokuments jedoch ständig im Fluss ist und zudem nicht alle Zeichen sichtbar sind, ist diese Methode nicht frei von Risiken. Allgemein empfiehlt es sich, Alternativen zur Arbeit mit numerischen Werten zu suchen. Im Folgenden stellen wir Ihnen einige nützliche Arbeitsweisen vor.

### Einen Bereich definieren

Range.  
Select

Weil ein Range unsichtbar ist, hat man manchmal das Gefühl, die Arbeit damit gleicht einer Lotterie. Während der Programmentwicklung ist es hilfreich, den Code schrittweise auszuführen und die

momentane Position des Bereichs anzuzeigen. Dies wird erreicht, indem man ihn mit der Select-Methode markiert und so im Dokument sichtbar macht:

```
rng.Select
```

Diese Methode wird auch genutzt, um die Markierung an die gewünschte Stelle zur Weiterbearbeitung durch den Benutzer zu setzen, bevor ihm die Kontrolle über das Dokument zurückgegeben wird.

Range  
fest-  
legen

Falls Sie im Code mit einer bestehenden Markierung im Dokument beginnen müssen, wird diese wie folgt einer Variablen des Typs Range zugewiesen:

```
Dim rng As Word.Range  
Set rng = Selection.Range
```

In C#:

```
Word.Range rng = wdApp.Selection.Range
```

Fangen Sie in einem neuen Dokument an, sieht's so aus:

```
Dim rng as Word.Range  
Dim doc as Word.Document  
Set doc = Documents.Add  
Set rng = doc.Range
```

In C#:

```
object objMissing = System.Reflection.Missing.Value;  
Word.Document doc = wdApp.Documents.Add(ref objMissing, ref objMissing,  
    ref objMissing, ref objMissing);  
Word.Range rng = doc.Content;
```

Ein Bereich mitten im Dokument kann ebenfalls einem Range zugewiesen werden. Die Frage ist nur, wie der Bereich auffindig gemacht wird. Unter Umständen genügt es, ein vorhandenes Objekt anzusprechen. Um beispielsweise mit dem ersten Absatz des zweiten Dokumentabschnitts zu arbeiten:

```
Set rng = doc.Sections(2).Paragraphs(1).Range
```

Um mit der letzten Tabelle im Dokument zu arbeiten:

```
Set rng = doc.Tables(doc.Tables.Count).Range
```

Einige Objekte geben einen Range zurück und haben daher keine Range-Eigenschaft, wie etwa: Word (Wort), Character (Zeichen), Field.Code und Field.Result:



```
Set rng1 = doc.Fields(3).Result 'das Ergebnis des dritten Feldes
Set rng2 = doc.Words(10) 'das zehnte Wort
```

Oft muss eine bestimmte Zeichenfolge gefunden und bearbeitet werden. Dazu bietet Word seine *Suchen und Ersetzen*-Funktionalität an. Diese ist so umfangreich, dass sie im eigenen Abschnitt »Die Nadel im Heuhaufen: *Find/Replace* einsetzen« in diesem Kapitel näher erklärt wird.

**HINWEIS**

Da das Range-Objekt eine zentrale Rolle bei der programmtechnischen Bearbeitung von Word-Dokumenten spielt, finden Sie in allen folgenden Abschnitten sowie in anderen Kapiteln reichlich Beispiele für seine Verwendung. Deshalb befassen wir uns hier hauptsächlich mit den Grundsätzen sowie den wichtigsten Eigenschaften und Methoden.

## Einen Bereich bearbeiten

Text und  
Forma-  
tierung

Sobald ein Range vorliegt, kann ihm Text zugewiesen und der Text formatiert werden:

```
rng.Text = "Dieser Text ist fett."
rng.Bold = True
```

Da die Werte True und False für die Eigenschaften Bold und Italic in Wirklichkeit numerisch und nicht boolesch sind, muss in C# diese Eigenschaft mit -1 bzw. 0 festgelegt werden.

```
rng.Text = "Dieser Text ist fett."
rng.Bold = -1
```

Absatzformatierungen werden über die ParagraphFormat-Eigenschaft zugewiesen. Um den Abstand nach einem Absatz festzulegen:

```
rng.ParagraphFormat.SpaceAfter = 3 'Maß in Points angeben
```

**HINWEIS**

Da die Syntax für Zeichen- und Absatzformatierungen problemlos mit dem Makrorekorder ermittelt werden kann, gehen wir hier nicht näher darauf ein. Die Befehle dafür befinden sich unter dem Menü *Format* in der Benutzerschnittstelle.

Collapse-  
Methode

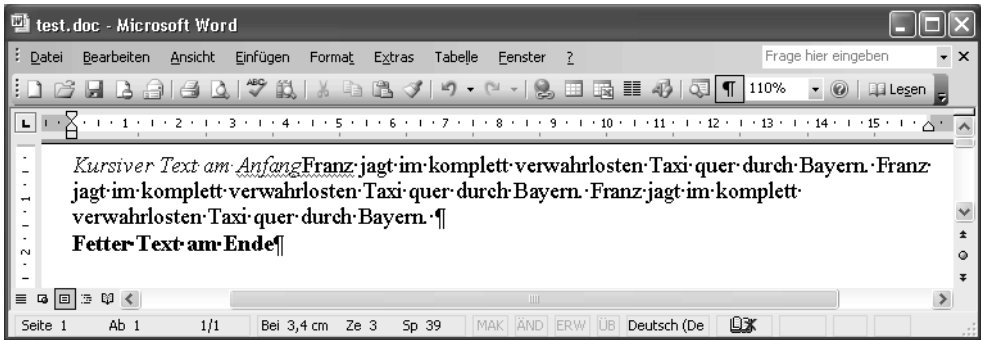
Wenn Sie ein Dokument, das schon Text enthält, öffnen und den Inhalt einem Range zuweisen, umfasst der Range den gesamten Inhalt des Dokument-Haupttextes (MainTextStory). Wird diesem Range dann Text zugewiesen, ersetzt er den im Dokument bereits vorhandenen Text.

Soll stattdessen der neue Text zusätzlich zum vorhandenen hinzugefügt und danach bearbeitet werden, sollte der Range zuerst auf einen Punkt verkleinert werden, wie in Listing 6.23 bzw. in Listing 6.24 ersichtlich. Dazu wird die Collapse-Methode verwendet, die über das optionale Direction-Argument einen von zwei Werten akzeptiert: Im Beispiel wird Text zuerst am Dokumentende, dann an dessen Anfang eingefügt und formatiert, ohne den bestehenden Text zu ändern.

- wdCollapseStart (1) verkleinert den Bereich auf den Startpunkt. Wird das Argument nicht angegeben, wird wdCollapseStart ausgeführt.

- wdCollapseEnd (0) verkleinert den Bereich auf den Endpunkt.

Abbildg. 6.7 Das Resultat von Listing 6.23



Listing 6.23 Den Bereich (Range) auf einen Punkt verkleinern, bevor Text eingefügt wird

```
Sub TextAmDokAnfangUndEnde()
    Dim doc as Word.Document
    Dim rng as Word.Range
    Dim strPfad as String

    strPfad = ThisDocument.Path & "\" & "Bsp06_01Range_Test.doc"
    Set doc = Documents.Open(strPfad)
    Set rng = doc.Range
    'Der neue Text erscheint am Dokumentende
    rng.Collapse wdCollapseEnd
    rng.Text = "Fetter Text am Ende"
    rng.Bold = True
    Set rng = doc.Range
    'Der neue Text erscheint am Dokumentanfang
    rng.Collapse wdCollapseStart
    rng.Text = "Kursiver Text am Anfang"
    rng.Italic = True
End Sub
```

Listing 6.24 (NET): Hier ist Code aus der Klasse für den Abschnitt über das *Document*-Objekt integriert, um das Dokument zu öffnen

```
private void Listing6_18_Click(object sender, System.EventArgs e)
{
    string dateiName = System.IO.Directory.GetCurrentDirectory();
    System.IO.DirectoryInfo di = new System.IO.DirectoryInfo(dateiName);
    string pfad = (di.Parent.Parent.Parent.Parent.FullName + "\\Bsp06_01Range_Test.doc");
    wd.Document doc = Kap06.Kap06Document.WordDokumentOeffnen_CS(wdApp, pfad);
    wd.Range rng = doc.Content;
    //Der neue Text erscheint am Dokumentende
    object objEndPunkt = (object) wd.WdCollapseDirection.wdCollapseEnd;
    rng.Collapse(ref objEndPunkt);
    rng.Text = "Fetter Text am Ende";
    rng.Bold = -1;
    rng = doc.Content;
    //Der neue Text erscheint am Dokumentanfang
```

**Listing 6.24** (NET): Hier ist Code aus der Klasse für den Abschnitt über das *Document*-Objekt integriert, um das Dokument zu öffnen

```
object objStartPunkt = (object) wd.WdCollapseDirection.wdCollapseStart;
rng.Collapse(ref objStartPunkt);
rng.Text = "Kursiver Text am Anfang";
rng.Italic = -1;
}
```



Die Beispieldatei *Bsp06\_01Range.doc* finden Sie auf der CD-ROM im Ordner \Kap06.

Insert-  
Metho-  
den

Es ist möglich, Text einem Bereich hinzuzufügen, ohne den Bereich vorab zu verkleinern. Es besteht jedoch keine Möglichkeit, auf diesen Text direkt wieder einzuwirken, um ihn beispielsweise zu formatieren; er »verschmilzt« mit dem im Bereich bestehenden Text. Zu diesem Zweck bietet das Word-Objektmodell einige Methoden an: *InsertAfter* (Text am Schluss des Bereichs einfügen), *InsertBefore* (Text am Anfang des Bereichs einfügen), *InsertBreak* (einen manuellen Umbruch (Wechsel) einfügen), *InsertParagraphAfter* (Absatz nach dem Bereich einfügen) und *InsertParagraphBefore* (Absatz vor dem Bereich einfügen).

```
rng.Text = "Text im Bereich."
rng.InsertAfter " Text dem Bereichende anfügen."
MsgBox rng.Text
'Ergebnis: Text im Bereich. Text dem Bereichende anfügen.
```

Die wichtigste dieser Methoden (weil es dafür keine Alternative gibt) ist *InsertBreak*. Damit werden Seiten- sowie Abschnittswchsel ins Dokument eingefügt. Ein Seitenwechsel ersetzt den Bereichsinhalt; ein Abschnittswchsel wird *vor* dem Bereich eingefügt und der Bereich auf den Punkt *zwischen* dem eingefügten Wechsel und dem ursprünglichen Bereich verkleinert (mehr über Abschnitte und Abschnittswchsel lesen Sie im Abschnitt »Abschnitte im Dokument: Das *Section*-Objekt« in diesem Kapitel) Um beispielsweise einen Abschnittswchsel des Typs *Nächste Seite* vor dem zweiten Absatz einzufügen:

```
Set rng = ActiveDocument.Paragraphs(2).Range
rng.InsertBreak Type:=wdSectionBreakContinuous
```

Und für C#:

```
wd.Document doc = wdApp.ActiveDocument;
wd.Range rng = doc.Paragraphs[2].Range;
object objBreakNextPage = (object) wd.WdBreakType.wdSectionBreakNextPage;
rng.InsertBreak(ref objBreakNextPage);
```

Die verschiedenen Umbruchtypen sind in Tabelle 6.10 aufgelistet.

**Tabelle 6.10** Die verschiedenen Umbruchtypen der Methode *InsertBreak*

WdBreakType-Enum	Wert	Beschreibung
wdColumnBreak	8	Fügt einen Spaltenwechsel (Zeitungsspalten) anstelle des gegenwärtigen Bereichs ein.
wdLineBreak	6	Fügt eine Zeilenschaltung anstelle des gegenwärtigen Bereichsinhalts ein.
wdLineBreakClearLeft wdLineBreakClearRight	9 10	Zeilenschaltungen für asiatische Dokumente.
wdPageBreak	7	Fügt eine neue Seite anstelle des gegenwärtigen Bereichsinhalts ein.
wdSectionBreakContinuous	3	Fügt einen fortlaufenden Abschnittswechsel vor dem gegenwärtigen Bereich ein und verkleinert ihn auf diesen Punkt.
wdSectionBreakEvenPage	4	Fügt einen geradeseitigen Abschnittswechsel vor dem gegenwärtigen Bereich ein und verkleinert ihn auf diesen Punkt.
wdSectionBreakNextPage	2	Fügt einen nächstseitigen Abschnittswechsel vor dem gegenwärtigen Bereich ein und verkleinert ihn auf diesen Punkt.
wdSectionBreakOddPage	5	Fügt einen ungeradeseitigen Abschnittswechsel vor dem gegenwärtigen Bereich ein und verkleinert ihn auf diesen Punkt.
wdTextWrappingBreak	11	Zeilenschaltungen für asiatische Dokumente.

Move-  
Metho-  
den

Ein Bereich kann schrittweise verkleinert oder erweitert werden. Dazu stehen mehrere Move-Methoden zur Verfügung: Move, MoveEnd, MoveEndUntil, MoveEndWhile, MoveStart, MoveStartWhile, MoveStartUntil, MoveUntil und MoveWhile.

Für Move, MoveEnd und MoveStart wird über das Argument Unit festgelegt, um welche Einheit dies geschieht. Das Argument Count bestimmt, um wie viele Einheiten der Bereich verschoben oder erweitert wird. Gültige Werte für Unit sind wdCharacter (Zeichen), wdWord (Wort), wdSentence (Satz), wdParagraph (Absatz), wdSection (Abschnitt), wdStory (Dokumentteil), wdCell (Zelle), wdColumn (Tabellenspalte), wdRow (Tabellenzeile) oder wdTable (Tabelle).

Um einen Bereich um einen Absatz in Richtung des Dokumentanfangs zu erweitern:

```
rng.MoveStart wdUnit:=wdParagraph, Count:=-1
```

In C#:

```
object objWordUnit = (object) wd.WdUnits.wdWord;
object objCountNeg1 = -1;
rng.MoveStart(ref objWordUnit, ref objCountNeg1);
```

#### ACHTUNG

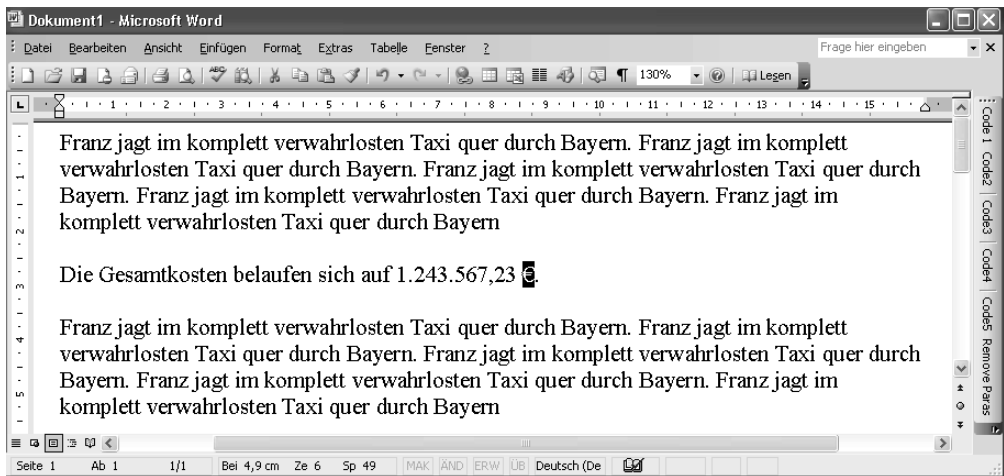
Denken Sie daran: Wenn eine MoveEnd-Methode mit einem positiven Unit-Wert verwendet wird, wird der Bereich erweitert; mit einem negativen Wert wird er verkleinert. Überschneiden sich dabei die End- und Startpunkte, wird der Bereich auf einen Punkt verkleinert und bleibt ein Punkt.

Für den Startpunkt ist es genau umgekehrt: negative Unit-Werte erweitern den Bereich; positive verkleinern ihn, bis er zum Punkt wird.

Die Methoden `MoveEndUntil`, `MoveEndWhile`, `MoveStartWhile`, `MoveStartUntil`, `MoveUntil` und `MoveWhile` ermöglichen eine bedingte Anpassung des Bereichsumfangs. Sie nehmen zwei Argumente: `CSet` sowie `Count`. `CSet` besteht aus einer Liste von Textzeichen und legt die Bedingung fest. Der Anfangs- bzw. Endpunkt wird verschoben, bis eines der Zeichen in der Liste angetroffen wird (»until«) bzw. bis keines der Zeichen angetroffen wird (»while«). `Count` bestimmt, um wie viele Zeichen maximal verschoben oder erweitert werden darf.

Das Ganze lässt sich nur schwer vorstellen, hier also ein kleines Beispiel, um das Prinzip zu veranschaulichen. Nehmen wir an, wir haben das Währungssymbol € gesucht und gefunden (Abbildung 6.8). Jetzt soll, wie in Listing 6.25 vorgestellt, die voranstehende Zahl (falls vorhanden) auch in den Bereich aufgenommen werden.

**Abbildg. 6.8** Die Kosten für Franz' teure Taxifahrt durch Bayern werden anhand des €-Zeichens im Dokument gefunden ...



Da das Euro-Symbol nach der Zahl steht, wird mit `MoveStartWhile` gearbeitet. Normalerweise steht ein Leerzeichen zwischen dem Euro-Symbol und der ersten Ziffer, wir wollen aber ein eventuell am Anfang der Zahl stehendes Leerzeichen *nicht* mit einbeziehen. Das bedeutet, der Bereich wird in zwei Schritten angepasst. Zuerst wird er um maximal eine Stelle erweitert, und zwar nur dann, wenn dieses Zeichen ein Leerzeichen ist.

Danach wird frei gegen Dokumentanfang erweitert, solange eine Ziffer, ein Komma (Dezimaltrennzeichen) oder Punkt (Tausendertrennzeichen) vorliegt. Am Schluss enthält der Bereich das Währungssymbol und die Zahl (Abbildung 6.9).

**Abbildg. 6.9** ... und der Bereich mit der `MoveStartWhile`-Methode erweitert, bis er die ganze Zahl enthält



**Listing 6.25** Einen Bereich bedingt erweitern

```
Sub ZahlPlusWaehrungssymbol()
    Dim rng As Word.Range
    Set rng = Selection.Range
    rng.MoveStartWhile CSet:= " ", Count:=-1
    rng.MoveStartWhile CSet:="1234567890,.", Count:=wdBackward
    MsgBox rng.Text
End Sub
```

**Listing 6.26** (.NET): Die C#-Version

```
private void ZahlPlusWaehrungssymbol_CS()
{
    //Markieren Sie zuerst ein Zeichen rechts neben einer Zahl
    wd.Range rng = wdApp.Selection.Range;
    object objCSet = (object) " ";
    object objCount = (object) -1;
    rng.MoveStartWhile(ref objCSet, ref objCount);
    objCSet="1234567890,.";
    objCount= (object) wd.WdConstants.wdBackward;
    rng.MoveStartWhile(ref objCSet, ref objCount);
    MessageBox.Show(rng.Text);
}
```



Die Beispieldatei *Bsp06\_02Range.doc* finden Sie auf der CD-ROM im Ordner *\Beispiele\Kap06*.

GoTo  
Methode

Die GoTo-Methode entspricht in etwa dem Dialogfeld *Bearbeiten/Gehe zu*. Im Allgemeinen finden die Autoren es besser, direkt mit dem Objektmodell zu arbeiten. Statt beispielsweise die erste Tabelle des Dokuments mit `Set rng = doc.Range.GoTo(What:=wdGoToTable, Which:=wdGoToAbsolute, Count:=1)` anzusprechen, würden wir eher `Set rng = doc.Tables(1).Range` einsetzen.

Hilfreich ist die Methode vor allem für Sachen, die im Objektmodell kein Gegenstück haben, wie etwa Seiten. Unglaublich, aber wahr: das Word-Objektmodell enthält kein »Page«-Objekt. Sie können nur mit GoTo einen Bereich für eine bestimmte Seite bekommen: `Set rng = doc.Range.GoTo(What:=wdGoToPage, Which:=wdGoToAbsolute, Count:=3)`. Mehr zu diesem Thema steht im Abschnitt »Zielscheibe Textmarke: Das Bookmark-Objekt« in diesem Kapitel.

Ebenfalls interessant sind die Objekttypen `wdGoToEquation` (Formel-Editor-Objekt) und `wdGoToLine` (Textzeile), die auch im Objektmodell kein Äquivalent haben und daher im Code sonst nicht direkt angesprochen werden können.

## Wo befindet sich der Bereich?

Bislang haben wir uns mit Methoden befasst, die einen Bereich festlegen, bearbeiten und erweitern oder verschieben. Oft, bevor wir eine Handlung ausführen, wäre es wichtig zu wissen, wo sich der Bereich befindet. Diese Frage ist besonders aktuell, wenn der Code mit dem Benutzer interaktiv agiert. Es wäre äußerst peinlich, würde der Code mit einer kryptischen VBA-Fehlermeldung abstürzen, nur weil sich die Markierung an einer ungültigen Stelle befindet!

**HINWEIS**

Lesen Sie zu diesem Thema auch über die Eigenschaft `Selection.Type` im Abschnitt »Die gegenwärtige Markierung: *Selection* und ähnliche Objekte« in diesem Kapitel.

Informa-  
tion

Das Word-Objektmodell stellt die Eigenschaft `Information` für die `Selection`- und `Range`-Objekte zur Verfügung. Diese ist etwas eigenartig, da sie eher einem Sammelsurium von Funktionen als einer Eigenschaft gleicht. Der Grund dafür liegt in der Urgeschichte der Anwendung, in der Word-Basic-Sprache. Um die einigermaßen problemlose Konvertierung von WordBasic-Makros in VBA zu ermöglichen, wurden viele Konstruktionen beibehalten, u.a. die von der Funktion `SelInfo()` – neu als `Information`-Eigenschaft – gelieferten Auskünfte.

Beim Aufruf dieser Eigenschaft müssen Sie ein Argument des Typs `wdInformation` übergeben, das festlegt, welche Art von Auskunft erfragt wird. Im Gegensatz zu vielen Enumerationen, sind diese in der VBA-Hilfe gut beschrieben, weshalb wir hier auf eine Auflistung verzichten.

Nehmen wir als Beispiel ein oft eingesetztes Argument: `wdWithinTable`. Damit wird festgestellt, ob sich der Bereich oder die Markierung innerhalb einer Tabelle befindet:

```
' Wenn »wahr«, befindet sich der Bereich innerhalb einer Tabelle
If rng.Information(wdWithinTable) Then
```

In C#:

```
//Wenn »wahr«, befindet sich der Bereich innerhalb einer Tabelle
if (rng.get_Information(wd.WdInformation.wdWithinTable))
```

Weitere nützliche Argumente sind: `wdHorizontalPositionRelativeToPage`, `wdHorizontalPositionRelativeToTextBoundary`, `wdVerticalPositionRelativeToPage`, und `wdVerticalPositionRelativeToTextBoundary`. Sie bieten die einzige Möglichkeit, herauszufinden, wo sich der Bereich oder die Markierung waagrecht und senkrecht auf der Seite befindet. Die Angabe erfolgt in Punkten (Points).

**HINWEIS**

Suchen Sie die Bildschirmkoordinaten einer Markierung oder eines Bereichs, schlagen Sie `GetPoint` in der allgemeinen VB-Hilfe nach.

InStory

Mit `Information` finden Sie auch heraus, auf welcher Seite sich der Bereich befindet, in welcher Textzeile er steht, ob ein Positionsrahmen markiert ist und vieles mehr.

Mit der Methode `InStory` stellen Sie fest, ob sich ein Bereich (oder eine Markierung) im gleichen Dokumentteil befindet wie ein anderer Bereich (oder eine andere Markierung).

Um das Prinzip zu veranschaulichen, nehmen wir an, der Benutzer hat über das Dialogfeld *Bearbeiten/Suchen* eine Zeichenfolge gefunden. Nun möchte er, dass an dieser Stelle eine Handlung ausgeführt wird. Was die Handlung tut, basiert darauf, ob das vorangegangene Suchergebnis sich im gleichen Dokumentteil befindet wie das aktuelle. Da *Suchen* in der Benutzerschnittstelle alle Dokumentteile durchsucht, könnte die Markierung im Haupttext stehen. Sie könnte sich aber auch in einer Kopfzeile, Fußnote oder in einem Kommentar befinden. Mit der folgenden Codezeile wird geprüft, ob sich der Markierungsbereich innerhalb des gleichen Dokumentteils befindet wie der Bereich der vorangegangenen Suche:

```
'Gibt »wahr« zurück, wenn die Markierung sich im gleichen Dokumentteil befindet,
'wie rngLetztesSuchErgebnis.
If Selection.InStory(rngLetztesSuchErgebnis) Then
```

In C#:

```
//Gibt »wahr« zurück, wenn die Markierung sich im gleichen Dokumentteil befindet,
//wie rngLetztesSuchErgebnis.
if (wdApp.Selection.InStory(rngLetztesSuchErgebnis))
```

**InRange** Ähnlich, aber nicht ganz gleich, ist die Methode `InRange`. Während `InStory` sich auf einen Dokumentteil bezieht, vergleicht `InRange` zwei Bereiche direkt miteinander. Befindet sich beispielsweise die Markierung in der zweiten Tabelle des Dokuments, gibt der folgende Test »Wahr« zurück:

```
If Selection.Range.InRange(ActiveDocument.Tables(2).Range) Then
```

**IsEqual** Es gibt noch eine dritte Methode dieser Art: `IsEqual`. Der Unterschied besteht darin, dass `IsEqual` kontrolliert, ob beide Bereiche genau die gleichen Start- und Endpunkte im gleichen Dokumentteil haben.

Diese Methode darf nicht mit dem Operator `Is` verwechselt werden. `Is` prüft, ob zwei Variablen auf das gleiche Objekt zeigen. Wenn wir Folgendes tun, müsste `rng1 Is rng2` »wahr« zurückgeben:

```
Set rng1 = ActiveDocument.Words(3)
Set rng1 = rng2
MsgBox (rng1 Is rng2)
MsgBox (rng1.IsEqual(rng2))
```

Im nachstehenden Fall aber wird »falsch« zurückgegeben. `IsEqual` hingegen gibt in beiden Fällen »wahr« zurück.

```
Set rng1 = ActiveDocument.Words(3)
Set rng2 = ActiveDocument.Words(3)
MsgBox (rng1 Is rng2)
MsgBox (rng1.IsEqual(rng2))
```

Der Unterschied ist subtil, aber wichtig und hat damit zu tun, wie Objekte im Hintergrund von VBA verwaltet werden. Wenn Sie mit `Set` eine Objektvariable ins Leben rufen, erstellt VBA das Objekt und die Variable »zeigt« darauf. Wird eine zweite Variable gleich einer bestehenden gesetzt, speichert sie einen zweiten »Zeiger« zum gleichen Objekt. Dies spart Ressourcen.

Benutzen Sie jedoch `Set` nochmals, um auf ein Objekt zu zeigen, erstellt VBA ein zusätzliches Objekt im Speicher. Es spielt keine Rolle, ob dafür schon ein Objekt im Speicher erstellt wurde, für VBA handelt es sich um ein völlig anderes.

Eine Alternative zum zweiten Listing oben wäre

```
Set rng1 = ActiveDocument.Words(3)
Set rng2 = rng1.Duplicate
MsgBox (rng1 Is rng2)
MsgBox (rng1.IsEqual(rng2))
```



Duplicate

Range.Duplicate kopiert das unterliegende Objekt selbst statt den Zeiger zum Objekt. Dieses Prinzip wird im Abschnitt »Die Nadel im Heuhaufen: *Find/Replace* einsetzen« in diesem Kapitel nochmals veranschaulicht.

## Text aus einem Bereich lesen

Vorhin haben Sie gesehen, wie Text einem Bereich zugewiesen wird. Auf ähnliche Art wird er aus einem Bereich geholt:

```
strBereichText = rng.Text
```

TextRetrievalMode

Soweit, so gut und einfach. Nur stellt sich die Frage, was ist da alles dabei? Wie steht's mit verborgenem Text oder Feldcodes? Sollen diese mit einbezogen oder beiseite gelassen werden?

Dafür stellt das Objektmodell TextRetrievalMode bereit, das selbst zwei Eigenschaften hat: IncludeFieldCodes und IncludeHiddenText. Wird eine dieser Eigenschaften auf True gesetzt, enthält die Zeichenkette die Feldcodes statt des Feldresultats bzw. den verborgenen Text. Das Listing 6.27 enthält eine Prozedur, die die Wirkung der verschiedenen Einstellungen demonstriert. Wird sie auf das Dokument in Abbildung 6.10 ausgeführt, ist Folgendes festzustellen:

- Standardmäßig ist IncludeHiddenText »wahr«, aber IncludeFieldCodes »falsch«.
- Auf Feldcodes, die automatisch verborgen sind (wie *XE* und *RD*), hat IncludeFieldCodes keine Wirkung, nur IncludeHiddenText.

### HINWEIS

Die Kästchen in Abbildung 6.10 entsprechen den Feldklammern und geben die ANSI-Zeichen 19 bzw. 21 zurück. Diese Zeichen sind aber nur der »sichtbare« Teil der Feldcodeklammer. Im Hintergrund enthalten sie noch viel mehr Information. Es ist also nicht möglich, durch Einfügen dieser Zeichen ein Feld ins Dokument einzufügen. Dies muss über den internen Befehl von Word geschehen (**[Strg]+[F9]** in der Benutzerschnittstelle oder über `Fields.Add` bei der Automatisierung). Mehr über die Arbeit mit Feldfunktionen lesen Sie im Abschnitt »Feldfunktionen« in diesem Kapitel.

Listing 6.27

Die verschiedenen Einstellungen von *TextRetrievalMode* vergleichen

```
Sub TextRetrievalModeTesten()
    Dim rng As Word.Range

    Set rng = ActiveDocument.Range
    MsgBox "Standardeinstellung:" & vbCrLf & rng.Text
    rng.TextRetrievalMode.IncludeFieldCodes = True
    rng.TextRetrievalMode.IncludeHiddenText = True
    MsgBox "Feldcodes ein; verborgener Text ein:" & vbCrLf & rng.Text
    rng.TextRetrievalMode.IncludeFieldCodes = False
    rng.TextRetrievalMode.IncludeHiddenText = True
    MsgBox "Feldcodes aus; verborgener Text ein:" & vbCrLf & rng.Text
    rng.TextRetrievalMode.IncludeFieldCodes = True
    rng.TextRetrievalMode.IncludeHiddenText = False
    MsgBox "Feldcodes ein; verborgener Text aus:" & vbCrLf & rng.Text
    rng.TextRetrievalMode.IncludeFieldCodes = False
    rng.TextRetrievalMode.IncludeHiddenText = False
    MsgBox "Feldcodes und verborgener Text aus:" & vbCrLf & rng.Text
End Sub
```

Abbildg. 6.10 *TextRetrievalMode* beeinflusst, wie Feldcodes und verborgener Text zurückgegeben werden



Format-  
tedText

Ferner hat *TextRetrievalMode* eine *View*-Eigenschaft, die es ermöglicht, den Text aus dem Bereich so zu lesen, wie er in einer bestimmten Ansicht wiedergegeben wird.

*Range.Text* gibt nur reinen Text zurück. Die meisten Word-Dokumente bestehen jedoch aus mehr als nur Text. Es gibt keinen Befehl, womit Sie alle Formatierungen auf einmal lesen können. Jede Eigenschaft muss einzeln abgefragt und in einer Variablen gespeichert werden, wenn Sie gezielt damit arbeiten möchten.

Geht es aber darum, formatierten Text von einer Stelle zu einer anderen zu kopieren, bietet das Word-Objektmodell die Eigenschaft *FormattedText* an. Damit können Sie schnell und bequem Text kopieren, ohne die Zwischenablage zu beanspruchen.

Listing 6.28 Formatierten Text von einem Bereich in einen anderen »kopieren«

```
Sub FormattedTextTesten()  
    Dim rng As Word.Range  
    Dim docNeu As Word.Document  
  
    Set rng = ActiveDocument.Content.Paragraphs(3).Range  
    Set docNeu = Documents.Add  
    docNeu.Content.FormattedText = rng.FormattedText  
End Sub
```



Die Beispieldatei *Bsp06\_03Range.doc* finden Sie auf der CD-ROM (siehe Einleitung) im Ordner *\Beispiele\Kap06*.

## Eine Formel berechnen

Calcu-  
late

Das *Range*-Objekt verfügt über eine interessante und unerwartete Methode, die es erlaubt, eine im Text stehende Formel zu berechnen. Einige, meist ältere, Programmiersprachen besitzen mit der Funktion *Eval* eine ähnliche Möglichkeit, die Visual Basic für Applikationen jedoch fehlt. Wir verdanken es dem alten WordBasic, dass die *Calculate*-Methode überhaupt existiert.

So kann beispielsweise der Benutzer in eine *InputBox* eine Formel eingeben, und wir können, ohne großen Aufwand, das Ergebnis zurückgeben. Das Listing 6.29 zeigt, wie es geht, und in Abbildung 6.11 sehen Sie einen Teil des Ergebnisses.

Abbildg. 6.11 Das passiert, wenn *ScreenUpdating* auf »falsch« gesetzt und ein Meldungsfeld verschoben wird

Listing 6.29 Mit Formeln im Text rechnen

```

Sub CalculateTest()
    Dim rng As Word.Range
    Dim strFormel As String

    Application.ScreenUpdating = False
    Set rng = ActiveDocument.Content.Paragraphs(5).Range
    MsgBox rng.Text & " = " & rng.Calculate
    rng.Collapse Direction:=wdCollapseEnd
    'Den Benutzer zur Eingabe einer Formel auffordern
    strFormel = InputBox("Bitte eine Formel eingeben:")
    'Abbrechen, wenn nichts eingegeben wurde
    If Len(strFormel) = 0 Then
        MsgBox "Keine Formel wurde eingegeben!", vbOKOnly
        Exit Sub
    End If
    'Diese am Ende des Bereichs einfügen
    rng.Text = strFormel
    'Die Berechnung ausführen
    MsgBox "Das Ergebnis ist: " & rng.Calculate
    'Die Formel wieder entfernen
    rng.Delete
End Sub

```

Listing 6.30 (NET): Eine Textbox in einer Windows-Form wird für die Benutzereingabe eingeblendet

```

private void CalculateTest_CS()
{
    try
    {
        wdApp.ScreenUpdating = false;
        string dateiName = System.IO.Directory.GetCurrentDirectory();
        System.IO.DirectoryInfo di = new System.IO.DirectoryInfo(dateiName);
        string pfad = (di.Parent.Parent.Parent.Parent.FullName + "\\Bsp06_04Range.doc");
        wd.Document doc = Kap06.Kap06Document.WordDokumentOeffnen_CS(wdApp, pfad);
    }
}

```

**Listing 6.30** (NET): Eine Textbox in einer Windows-Form wird für die Benutzereingabe eingeblendet (*Fortsetzung*)

```

        wd.Range rng = doc.Content.Paragraphs[2].Range;
        object objCollapseEnd = wd.WdCollapseDirection.wdCollapseEnd;
        MessageBox.Show(rng.Text + " = " + rng.Calculate());
        rng.Collapse(ref objCollapseEnd);
        //Den Benutzer zur Eingabe einer Formel auffordern
        lblFormel.Visible = true;
        txtFormel.Visible = true;
        btnFormel.Visible = true;
    }
    finally
    {
        wdApp.ScreenUpdating = true;
    }
}

private void btnFormel_Click(object sender, System.EventArgs e)
{
    try
    {
        wdApp.ScreenUpdating = false;
        wd.Document doc = wdApp.ActiveDocument;
        wd.Range rng = doc.Content.Paragraphs[1].Range;
        object objCollapseEnd = wd.WdCollapseDirection.wdCollapseEnd;
        rng.Collapse(ref objCollapseEnd);
        string formel = txtFormel.Text;
        //Diese am Ende des Bereichs einfügen
        rng.Text = formel;
        //Die Berechnung ausführen
        MessageBox.Show("Das Ergebnis ist: " + rng.Calculate());
        //Die Formel wieder entfernen
        object objMissing = System.Reflection.Missing.Value;
        rng.Delete(ref objMissing, ref objMissing);
    }
    finally
    {
        lblFormel.Visible = false;
        txtFormel.Visible = false;
        btnFormel.Visible = false;
        wdApp.ScreenUpdating = true;
    }
}

```



Die Beispieldatei *Bsp06\_04Range.doc* finden Sie auf der CD-ROM im Ordner *\Beispiele\Kap06*.

# Die Nadel im Heuhaufen: *Find/Replace* einsetzen

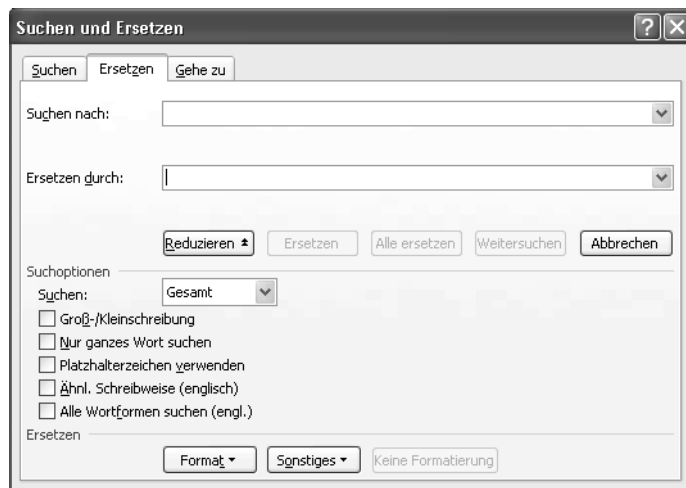
Die Word-Funktionalität *Suchen und Ersetzen* ist eine der nützlichsten der gesamten Anwendung. In diesem Teil werden wir einen Überblick über dessen Automatisierung vorstellen.

**HINWEIS** *Suchen und Ersetzen* in Word kann viel mehr als nur Zeichenketten im Text aufspüren, und diese, wenn gewünscht, mit einer anderen ersetzen. Es können auch Formatierungen gesucht und ersetzt sowie mit Platzhalterzeichen (die Regular Expressions (RegEx) ähnlich, aber nicht gleich sind) gearbeitet werden. Da eine Diskussion der Funktionalität der Benutzerschnittstelle die Grenzen dieses Buches sprengen würde, haben wir umfangreiche Informationen in der Datei *SuchenErsetzen.pdf* auf der CD-ROM zum Buch im Ordner `\Beilagen\SuchenErsetzen` bereitgestellt. Falls Sie mit den Möglichkeiten nicht schon vertraut sind, lohnt es sich, einen Blick darauf zu werfen, da überraschend viele Aufgaben ohne zusätzliches Programmieren lösbar sind.

## Vor- und Nachteile des Makrorekorder-Resultats

Im Allgemeinen liefert der Makrorekorder einen brauchbaren Code für die *Suchen und Ersetzen*-Funktionalität. Ein Beispiel hierfür sehen Sie in Listing 6.31. Mit nur wenigen Ergänzungen kann das Resultat problemlos als Automatisierungscode eingesetzt werden. Viele der Argumente stimmen mit den englischen Bezeichnungen der Dialogfeld-Steuerelemente überein, so dass es relativ einfach ist, die Handlung nachzuvollziehen. Die deutsche Version des Dialogfelds sehen Sie in Abbildung 6.12, die Tabelle 6.11 bietet eine Übersicht der entsprechenden englischen Begriffe.

**Abbildg. 6.12** Das Dialogfeld veranschaulicht einen Großteil der Funktionalität, die auch dem Entwickler zur Verfügung steht



**Tabelle 6.11** Übersicht der Parameter für das *Suchen und Ersetzen* mit VBA

Parameter	Beschriftung in der deutschen Umgebung
Text	<i>Suchen nach</i>
Replacement.Text	<i>Ersetzen durch</i>
Forward	<i>Suchen</i>
Wrap	[Kein entsprechendes Steuerelement. Legt fest, wie sich <i>Suchen und Ersetzen</i> am Ende einer Story verhält.]
Format	[Formatierungen werden gesucht]
MatchCase	<i>Groß-/Kleinschreibung</i>
MatchWholeWord	<i>Nur ganzes Wort suchen</i>
MatchWildcards	<i>Platzhalterzeichen verwenden</i>
MatchSoundsLike	<i>Ähnl. Schreibweise (englisch)</i>
MatchAllWordForms	<i>Alle Wortformen suchen (engl.)</i>
ClearFormatting	<i>Keine Formatierung</i>

**Listing 6.31** Ein aufgezeichnetes Makro, das die Zeichenfolge »updaten« durch »aktualisieren« ersetzt

```

Sub AlleUpdatenMitAktualisierenErsetzen()
    Selection.Find.ClearFormatting
    Selection.Find.Replacement.ClearFormatting
    With Selection.Find
        .Text = "updaten"
        .Replacement.Text = "aktualisieren"
        .Forward = True
        .Wrap = wdFindContinue
        .Format = False
        .MatchCase = True
        .MatchWholeWord = True
        .MatchWildcards = False
        .MatchSoundsLike = False
        .MatchAllWordForms = False
    End With
    Selection.Find.Execute Replace:=wdReplaceAll
End Sub

```

## Verhaltensunterschiede zur Benutzerschnittstelle

Der Beispielcode in Listing 6.31 beginnt so, wie in der Benutzerschnittstelle gearbeitet werden sollte: mit der Entfernung aller Formatierungseinstellungen. Das geschieht über die Methode `ClearFormatting`, wobei der Makrorekorder diese Zeilen zurückgibt, auch wenn die Schaltfläche nicht betätigt wurde.

Das aufgezeichnete Makro läuft einwandfrei, macht aber nicht genau das, was der gleiche Vorgang, ausgeführt in der Benutzerschnittstelle, tut. Es sucht nämlich nur den gegenwärtigen Textteil (StoryRange) ab, nicht das ganze Dokument mit sämtlichen Kopf- und Fußzeilen, Fußnoten, Endnoten und Autoformen. Anders ausgedrückt: Ein aufgezeichnetes Makro verhält sich ähnlich wie

das Dialogfeld, wenn Sie aus dem Dropdown-Feld *Suchen* entweder *Nach oben* oder *Nach unten* wählen.

Das Argument *Forward* entspricht dem Feld *Suchen*, es akzeptiert jedoch nur boolesche Werte, also entweder »Wahr« oder »Falsch«. *Forward* bedeutet soviel wie *Nach unten*. Wenn es auf *False* gesetzt wird, sucht Word *Nach oben*. Word-VBA hat keine Option, die der Einstellung *Gesamt* entspricht.

Wenn Sie in der Benutzerschnittstelle *Nach unten* oder *Nach oben* suchen, fragt Word am Ende (bzw. am Anfang) des Dokuments oder der Markierung, ob das restliche Dokument durchsucht werden soll. Je nach Wert des Arguments *Wrap* fragt Word dies bei der Ausführung des Makros nicht. In Listing 6.31 hat *Wrap* den Wert *wdFindContinue* (Word sucht weiter, ohne zu fragen). Die beiden anderen Möglichkeiten sind *wdFindAsk* (Word fragt) und *wdFindStop* (Word beendet die Suche).

### WICHTIG

Gehen Sie mit *wdFindContinue* sorgfältig um. Wenn die Suche für eine Zwischenhandlung unterbrochen wird, kann diese Einstellung zu einer Endlosschleife führen, die nur mit **Strg** + **Pause** abgebrochen werden kann.

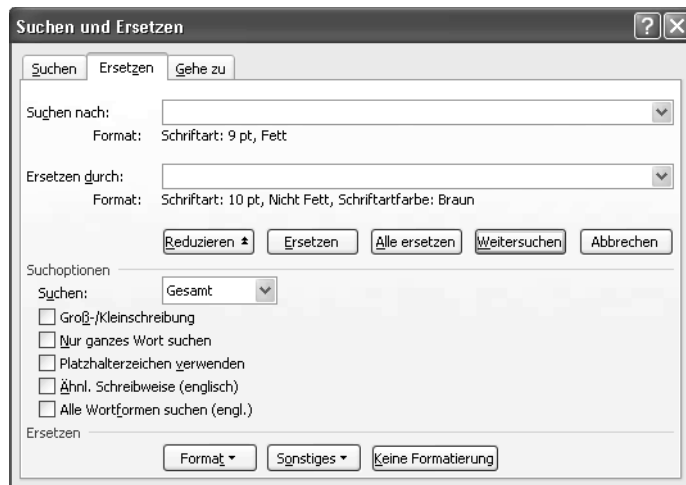
### HINWEIS

Im Word-Objektmodell gibt es kein Pendant zum Kontrollkästchen *Gefundene Elemente markieren in*, das sich im Dialogfeld *Bearbeiten/Suchen* der Word-Versionen 2002 und 2003 befindet.

## Formatierungen suchen und ersetzen

Es gibt einen Bereich, wo der Makrorekorder seinen Dienst versagt: Bei der Aufzeichnung bestimmter Schrifteinstellungen. Dieses Fehlverhalten erfordert eine manuelle Anpassung des Codes, wobei sich die »IntelliSense«-Funktion des VB-Editors als sehr hilfreich erweist. Vergleichen Sie die Abbildung 6.13 mit dem Code in Listing 6.32. Die fünf Zeilen mit dem Vermerk »hinzugefügt« wurden vom Makrorekorder nicht aufgezeichnet. Sie müssen die Anweisungen *.Font.Bold*, *.Font.Size* usw. selber eingeben.

**Abbildg. 6.13** Diese Kriterien für die Schriftformatierung werden vom Makrorekorder nicht aufgezeichnet



**Listing 6.32** Vom Makrorekorder nicht aufgezeichnete *Suchen und Ersetzen*-Kriterien müssen manuell hinzugefügt werden

```
Sub Fett9PunktSuchenMitRot10PunktErsetzen()
    Selection.Find.ClearFormatting
    Selection.Find.Replacement.ClearFormatting
    With Selection.Find
        .Text = ""
        .Font.Bold = True 'hinzugefügt
        .Font.Size = 9 'hinzugefügt
        .Replacement.Text = ""
        .Replacement.Font.Bold = False 'hinzugefügt
        .Replacement.Font.Color = wdColorDarkRed 'hinzugefügt
        .Replacement.Font.Size = 10 'hinzugefügt
        .Forward = True
        .Wrap = wdFindContinue
        .Format = True
        .MatchCase = False
        .MatchWholeWord = False
        .MatchWildcards = False
        .MatchSoundsLike = False
        .MatchAllWordForms = False
    End With
    Selection.Find.Execute Replace:=wdReplaceOne
End Sub
```

## Anpassung des aufgezeichneten Codes

Ein aufgezeichnetes Makro muss für einfache Aufgaben oft nicht oder nur unwesentlich geändert werden. Für komplexere Anwendungen hingegen sind gewisse Anpassungen notwendig, wenn beispielsweise die Suche durch andere Handlungen unterbrochen wird oder wenn die Suche in mehreren Dokumentteilen (StoryRanges) durchgeführt werden soll.

In diesem Abschnitt stellen wir einige der meist gebrauchten Szenarien vor.

### Suche mit einer Handlung unterbrechen (in einer Schleife ausführen)

Wie schon häufiger im Buch erwähnt, soll das Selection-Objekt möglichst gemieden und durch das Range-Objekt ersetzt werden. Diese Regel gilt grundsätzlich auch für die *Suchen und Ersetzen*-Funktion, ist aber für ein »reines« Suchen oder Ersetzen (»alles Ersetzen«) weniger zwingend. Für die folgenden Aufgaben ist ein Beibehalten des Selection-Objekts sinnvoll bzw. unbedenklich:

- Wenn die Suche die Markierung im Dokument verschiebt, weil der Anwender an der gefundenen Stelle eine Handlung ausführen soll.
- Wenn alle Vorkommen des Suchbegriffs durch den Ersatzbegriff ersetzt werden.

Range.  
Find

Das Range-Objekt wird also hauptsächlich gebraucht, wenn als Ziel einer Prozedur bei jedem gefundenen Vorkommen eines Suchbegriffs eine Handlung ausgeführt werden soll, die mit der *Ersetzen*-Funktionalität nicht erreichbar ist. Nehmen wir als Beispiel ein Dokument, in welches der Benutzer einige vordefinierte Begriffe eingibt. Unsere Anwendung sucht diese und fügt, anhand des letzten Wortes, einen AutoText-Eintrag ein. (Es könnte natürlich etwas viel Aufwändigeres sein, z.B. die Erstellung einer Tabelle mit aktuellen Daten aus einer Datenbank. Aber unsere Beispiele sollen überschaubar bleiben ...)



Dies bedeutet, dass die Suche nach jeder Handlung erneut aufgenommen werden muss. Wird mit dem Selection-Objekt gearbeitet, so wissen wir, dass

- zunächst der markierte Text durchsucht wird, bevor die Suche im restlichen Dokument fortgesetzt wird,
- die Markierung im Dokument herumspringt und nach jeder erfolgreichen Ausführung die Suche von dieser Stelle aus weiterläuft.

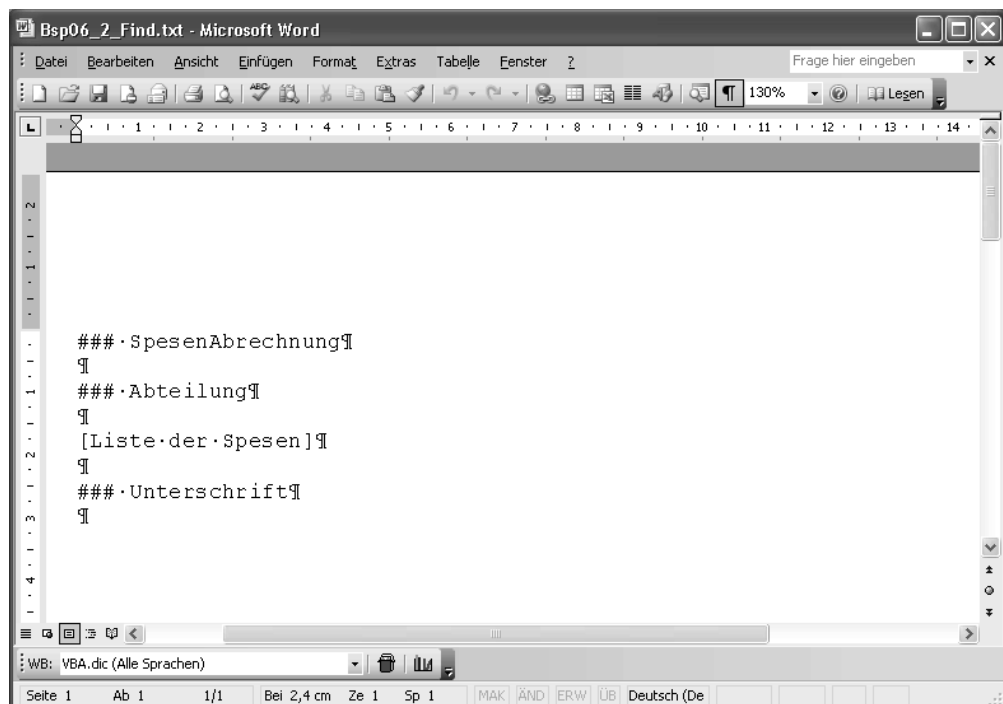
Ähnliches gilt auch für die Suche mit dem Range-Objekt, aber

- die Suche wird im angegebenen Bereich durchgeführt,
- der Bildschirm bleibt ruhig,
- bei erfolgreicher Suche verkleinert sich der Range auf den gefundenen Bereich.

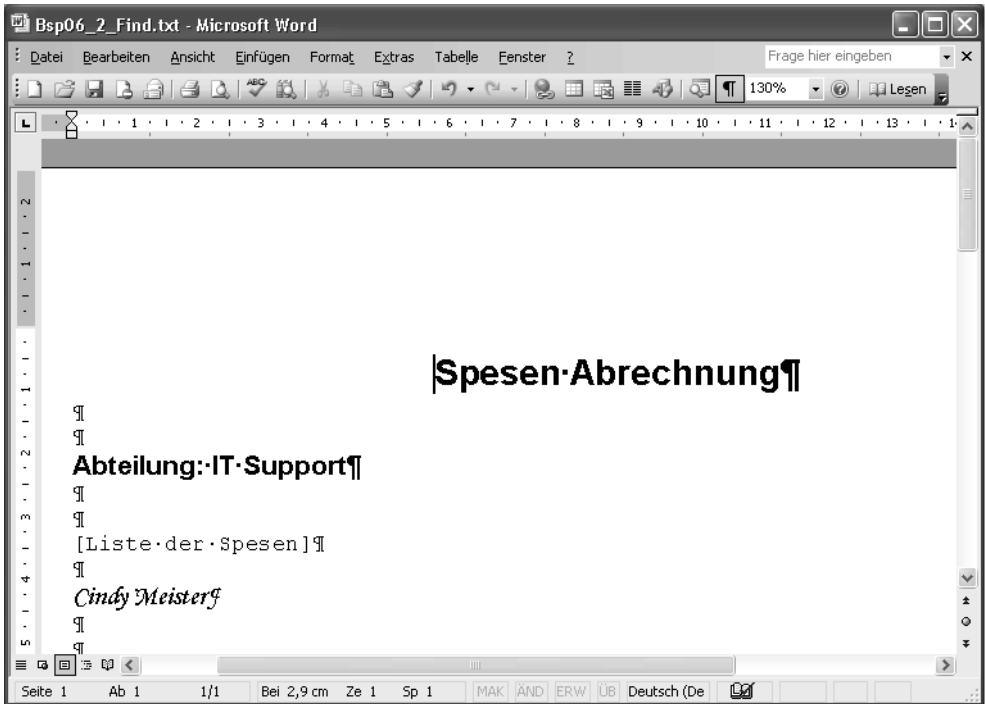
Der Hauptunterschied hier ist, dass die Weitersuche sich auf den *gefundenen Bereich* beschränkt, was meistens nicht erwünscht ist. Folglich muss vor Weiterführung der Suche der Bereich neu gesetzt werden.

Bei unserem Beispiel führt der Mitarbeiter im Außendienst seine Spesenabrechnung auf seinem kleinen Handgerät. Es handelt sich um eine einfache Textdatei, die später in Word geöffnet (Abbildung 6.14) und mit der Prozedur eines globalen Add-Ins – *SpesenrechnungVervollstaendigen* in Listing 6.33 bzw. Listing 6.34 – bearbeitet wird. Die Einträge, die die Suchbegriffe ersetzen, sind als AutoText-Einträge in einer anderen, benutzerspezifischen Vorlage gespeichert. Die globale Vorlage steht also allen Mitarbeitern der Firma zur Verfügung, aber jeder hat eine eigene, lokal gespeicherte Vorlage mit seinen persönlichen Angaben. Das Ergebnis ist in Abbildung 6.15 dargestellt.

Abbildg. 6.14 Eine einfache Spesenabrechnung. Den zu ersetzenden Begriffen stehen die Zeichen ### voran.



Abbildg. 6.15 Das Resultat von Listing 6.33. Die Formatierungen sind in den AutoText-Einträgen definiert.



In *SpesenrechnungVervollstaendigen* werden die Vorbereitungen getroffen. Der Suchbegriff wird festgelegt, zudem stellt die Prozedur sicher, dass wir es nicht mit einer Vorlage zu tun haben. Sonst würde das darauf folgende Anfügen der Vorlage mit den persönlichen Informationen fehlschlagen. Anschließend wird die Kernfunktion *BenutzerInfoEinfuegen* aufgerufen und ihr der Suchbegriff und das Document-Objekt als Argumente überreicht.

Range.  
Dupli-  
cate

Am Anfang dieser Prozedur werden zwei Bereiche definiert: *rngSuchBereich* und *rngErgebnis*. Der erste wird dem Haupttextbereich des Dokuments gleichgesetzt. Der zweite wird *nicht* dem ersten, sondern mittels der Eigenschaft *Duplicate* einer Kopie davon gleichgesetzt. Würde der zweite Bereich dem ersten gleichgesetzt, hätte jede Änderung am zweiten Range-Objekt zur Folge, dass diese auch für den ersten übernommen wird. Das Ziel ist es jedoch, nach jeder erfolgreichen Suche immer wieder auf den ursprünglichen Suchbereich zurückgreifen zu können.

Die  
Found-  
Eigen-  
schaft

Danach wird *Find* mit dem Range-Objekt *rngErgebnis* ausgeführt. Sie sehen, dass dieses nicht anders als ein *Selection*-Objekt verwendet wird. Dieser Teil steht innerhalb einer *Do...Loop*-Schleife. Da *While* sich in der Zeile mit *Loop* befindet, wird die Schleife mindestens einmal ausgeführt. Ist die Suche erfolgreich, gibt die *Found*-Eigenschaft »Wahr« zurück, und die Schleife wird wiederholt.

Ferner werden die Codezeilen zwischen *If* und *End If* ausgeführt: Der gefundene Text (###) wird gelöscht und der Bereich um das nächste Wort erweitert. Dieses dient dann als Name des AutoText-Eintrags, der an seiner Stelle eingefügt wird. Vor der Wiederholung der Schleife wird der Endpunkt von *rngErgebnis* auf den Endpunkt von *rngSuchBereich* gestellt, was heißt, dass die Suche nur von diesem Punkt an bis zum Ende des Dokuments durchgeführt wird. (Hätten wir es nochmals gleich

rngSuchBereich.Duplicate gesetzt, müsste der Code wieder das ganze Dokument durchsuchen, was bei einem großen Dokument deutlich langsamer wäre.)

**HINWEIS**

Um das Beispiel in Listing 6.33 auszuführen, kopieren Sie *Bsp06\_02\_Find.dot* in den *StartUp*-Ordner von Word. Wenn Word gestartet wird, wird die Vorlage als ein Add-In geladen. Öffnen Sie die Datei *Bsp06\_02\_Find.txt* in Word (befindet sich im Ordner *\Beispiele\Kap06*) und führen Sie das Makro *SpesenrechnungVervollstaendigen* aus, das Sie in *Extras/Makros* finden werden. (Die Datei *Bsp06\_01\_Find.dot*, die die AutoText-Einträge enthält, muss sich im gleichen Ordner mit *Bsp06\_02\_Find.txt* befinden.

Listing 6.33

Nach jeder erfolgreichen Suche findet eine Handlung statt

```
Sub SpesenrechnungVervollstaendigen()
    Dim doc As Word.Document
    Dim strSpesenVorlage As String
    Dim lAnzEintraege As Long
    Dim strSuchBegriff As String

    strSuchBegriff = "### "
    Set doc = ActiveDocument
    strSpesenVorlage = ActiveDocument.Path & "\Bsp06_01_Find.dot"
    If doc.Type = wdTypeDocument Then
        'Die Benutzer-spezifische Vorlage der Spesenrechnung anhängen
        doc.AttachedTemplate = strSpesenVorlage
        'Die mit AutoText zu ersetzenden Einträge suchen
        lAnzEintraege = BenutzerInfoEinfuegen(doc, strSuchBegriff)
    End If
    'Den Benutzer auffordern, das Dokument zu speichern
    With Dialogs(wdDialogFileSaveAs)
        'Speichern unter-Dialogfeld voreinstellen für den Dateityp "Word-Dokument"
        .Format = 0
        .Show
    End With
    Debug.Print lAnzEintraege & " AutoText-Einträge wurden eingefügt."
End Sub

Private Function BenutzerInfoEinfuegen(ByRef doc As Word.Document, _
    strSuchBegriff As String) As Long
    Dim lEintraegeZaehler As Long
    Dim rngSuchBereich As Word.Range
    Dim rngErgebnis As Range
    Dim tmplAutoTextBehaelter As Word.Template
    Dim bFound As Boolean

    Set rngSuchBereich = doc.Content
    Set rngErgebnis = rngSuchBereich.Duplicate
    Set tmplAutoTextBehaelter = doc.AttachedTemplate

    AllgemeineSuchkriterienFestlegen rngErgebnis
    Do
        With rngErgebnis.Find
            .Text = strSuchBegriff
            .Forward = True
            .Wrap = wdFindStop
            .Execute
            bFound = .Found
        End With
    Loop While bFound
```

**Listing 6.33** Nach jeder erfolgreichen Suche findet eine Handlung statt (*Fortsetzung*)

```

End With
If bFound Then
    'Das Suchergebnis soll nicht im Dokument bleiben
    rngErgebnis.Delete
    'Das Wort nach dem Ergebnis ist die Bezeichnung
    'des AutoText-Eintrags
    rngErgebnis.MoveEnd Unit:=wdWord, Count:=1
    'Nach dem Einfügen beinhaltet der Bereich den Text
    'des eingefügten AutoText-Eintrages
    'Falls der AutoText-Eintrag nicht vorhanden ist,
    'einfach fortfahren
    On Error Resume Next
    Set rngErgebnis = tmp1AutoTextBehaelter.AutoTextEntries(
    rngErgebnis.Text).Insert(Where:=rngErgebnis, RichText:=True)
    On Error GoTo 0
    rngErgebnis.End = rngSuchBereich.End
    lEintraegeZaehler = lEintraegeZaehler + 1
End If
Loop While bFound
BenutzerInfoEinfuegen = lEintraegeZaehler
End Function

Private Sub AllgemeineSuchkriterienFestlegen(ByRef rng As Word.Range)
    With rng.Find
        'Weitere Einstellungen wurden aus Platzgründen weggelassen.
        .ClearFormatting
        .MatchCase = False
        .MatchWholeWord = True
    End With
End Sub

```

**Listing 6.34** (.NET): In C# müssen alle Argumente der Methode *Find.Execute* angegeben werden

```

private void SpesenrechnungVervollstaendigen()
{
    //Sie müssen Bsp06_02_Find.txt öffnen, bevor das Makro ausgeführt wird.
    wd.Document doc = wdApp.ActiveDocument;
    string suchBegriff = "### ";
    string dateiName = System.IO.Directory.GetCurrentDirectory();
    System.IO.DirectoryInfo di = new System.IO.DirectoryInfo(dateiName);
    object spesenVorlage = (object)
        (di.Parent.Parent.Parent.Parent.FullName + "\\Bsp06_01_Find.dot");
    object objMissing = System.Reflection.Missing.Value;
    int anzEintraege = 0;
    if (doc.Type == wd.WdDocumentType.wdTypeDocument)
    {
        //Die Benutzer-spezifische Vorlage der Spesenrechnung anhängen
        doc.set_AttachedTemplate(ref spesenVorlage);
        //Die mit AutoText zu ersetzenden Einträge suchen
        anzEintraege = BenutzerInfoEinfuegen(doc, suchBegriff);
    }
    //Den Benutzer auffordern, das Dokument zu speichern
    wd.Dialog dlgSaveAs = wdApp.Dialogs[wd.WdWordDialog.wdDialogFileSaveAs];
    object[] parameters = new object[1];
    parameters[0] = (object) 0;
}

```

Listing 6.34 (.NET): In C# müssen alle Argumente der Methode *Find.Execute* angegeben werden (Fortsetzung)

```

//Speichern unter-Dialogfeld voreinstellen für den Dateityp "Word-Dokument"
dlgSaveAs.GetType().InvokeMember("Format", System.Reflection.BindingFlags.SetProperty,
    null, dlgSaveAs, parameters);
dlgSaveAs.Show(ref objMissing);
MessageBox.Show(anzEintraege + " AutoText-Einträge wurden eingefügt.");
}

private int BenutzerInfoEinfuegen(wd.Document doc, string suchBegriff)
{
    int eintraegeZaehler=0;
    bool bFound = false;
    wd.Range suchBereich = doc.Content;
    wd.Range suchErgebnis = suchBereich.Duplicate;
    wd.Template tmplAutoTextBehaelter = (wd.Template) doc.get_AttachedTemplate();
    object objTrue = true;
    object objFalse = false;
    object objMissing = System.Reflection.Missing.Value;
    do
    {
        object objFindText = (object) suchBegriff;
        object objFindWrap = (object) wd.WdFindWrap.wdFindStop;
        bFound = suchErgebnis.Find.Execute(ref objFindText, ref objFalse, ref objTrue,
            ref objFalse, ref objFalse, ref objFalse, ref objTrue, ref objFindWrap,
            ref objFalse, ref objMissing, ref objMissing, ref objFalse, ref objFalse,
            ref objFalse, ref objFalse);
        if (bFound)
        {
            //Das Suchergebnis soll nicht im Dokument bleiben
            suchErgebnis.Delete(ref objMissing, ref objMissing);
            //Das Wort nach dem Ergebnis ist die Bezeichnung des AutoText Eintrages
            object objWordUnit = wd.WdUnits.wdWord;
            object objCount1 = 1;
            suchErgebnis.MoveEnd(ref objWordUnit, ref objCount1);
            //Nach dem Einfügen beinhaltet der Bereich den Text
            //des eingefügten AutoText-Eintrags
            //Falls der AutoText-Eintrag nicht vorhanden ist, einfach fortzufahren
            try
            {
                object objAutoTextRange = (object) suchErgebnis;
                object objATName = (object) suchErgebnis.Text;
                wd.AutoTextEntry at = tmplAutoTextBehaelter.AutoTextEntries.get_Item(
                    ref objATName);
                suchErgebnis = at.Insert(suchErgebnis, ref objTrue);
            }
            catch {}
            suchErgebnis.End = suchBereich.End;
            eintraegeZaehler += 1;
        }
    } while (bFound);
    return eintraegeZaehler;
}

```



Die Beispieldateien *Bsp06\_01\_Find.dot* (enthält die AutoText-Einträge), *Bsp06\_02\_Find.txt* sowie *Bsp06\_02\_Find.dot* (enthält den Add-In-Code) finden Sie auf der CD-ROM im Ordner *\Beispiele\Kap06*.

Falls der Suchvorgang in einer Tabelle unterbrochen wird, gestaltet sich die Handlung etwas komplizierter, weil die Erweiterung eines Bereichs, der sich in einer Tabellenzelle befindet, automatisch ganze Tabellenreihen einschließt. Abbildung 6.16 und Listing 6.35 veranschaulichen das Problem. Zuerst wird der Bereich rng2 gleich der dritten Spalte der zweiten Tabellenzeile gesetzt. Dann wird der Bereich bis zum Dokumentende erweitert. Der Bereich erstreckt sich jedoch über die ganze zweite Zeile.

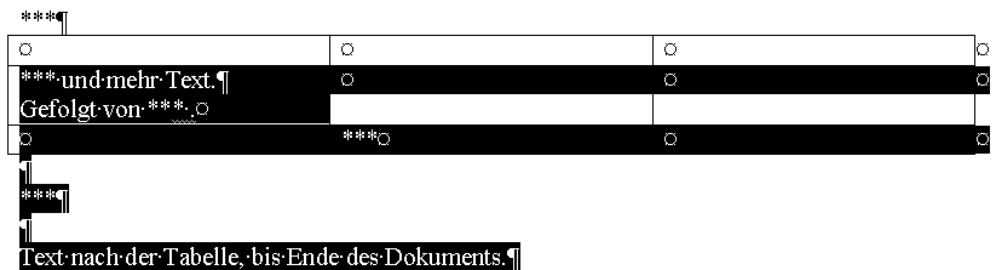
**Listing 6.35** Bereich in einer Tabelle bis zum Dokumentende erweitern

```
Sub RangeInTable()
    Dim rng1 As Word.Range
    Dim rng2 As Word.Range

    Set rng1 = ActiveDocument.Range
    Set rng2 = ActiveDocument.Tables(1).Cell(2, 3).Range
    rng2.Collapse wdCollapseStart
    rng2.Select

    rng2.End = rng1.End
    rng2.Select
End Sub
```

**Abbildg. 6.16** Das Resultat des Listing 6.35



Für das vorangehende Beispiel war das kein Problem, weil der Suchbegriff immer gelöscht wurde. Bleibt der Suchbegriff jedoch im Dokument bestehen, würde der Automatisierungscode sich in einer unendlichen Schleife verfangen.

In solchen Fällen muss das Suchen und Ersetzen innerhalb einer Tabelle Zelle für Zelle ausgeführt werden, wie das Listing 6.36 zeigt.

**Listing 6.36** Ein Dokument mit Tabelle durchsuchen

```
Sub SuchenMitTabelle()
    Dim strSuchBegriff As String
    Dim rngSuchBereich As Word.Range
    Dim rngErgebnisBereich As Word.Range
    Dim lngZaehler As Long
    Dim tbl As Word.Table

    strSuchBegriff = "****"
    Set rngSuchBereich = ActiveDocument.Content
    Set rngErgebnisBereich = rngSuchBereich.Duplicate
```

Listing 6.36 Ein Dokument mit Tabelle durchsuchen (Fortsetzung)

```

Do While BegriffSuchen(strSuchBegriff, rngErgebnisBereich)
    lngZaehler = lngZaehler + 1
    GefundeneStelleBearbeiten "Neuer Text" & CStr(lngZaehler), rngErgebnisBereich
    If rngErgebnisBereich.Information(wdWithInTable) Then
        'Die Suche in einer Tabelle erfordert, die Zellen einzeln zu bearbeiten.
        'Die Suche im aktuellen Zellenbereich ausführen.
        rngErgebnisBereich.End = rngErgebnisBereich.Cells(1).Range.End - 1
        Do While BegriffSuchen(strSuchBegriff, rngErgebnisBereich)
            lngZaehler = lngZaehler + 1
            GefundeneStelleBearbeiten "Neuer Text" & CStr(lngZaehler), rngErgebnisBereich
            If rngErgebnisBereich.Information(wdWithInTable) Then
                'Weiter in der gleichen Zelle suchen.
                rngErgebnisBereich.End = rngErgebnisBereich.Cells(1).Range.End - 1
            Else
                Exit Do
            End If
        Loop
        'Wenn die Suche innerhalb der Zelle erfolglos war, und der Ergebnisbereich
        'sich noch immer in einer Tabelle befindet...
        If rngErgebnisBereich.Information(wdWithInTable) Then
            '...und nicht in der letzten Zelle steht...
            Set tbl = rngErgebnisBereich.Tables(1)
            If Not rngErgebnisBereich.Cells(1) Is tbl.Range.Cells(tbl.Range.Cells.Count) Then
                'ihn in die folgende Zelle setzen,
                rngErgebnisBereich.MoveStart Unit:=wdCell, Count:=1
                rngErgebnisBereich.MoveEnd Unit:=wdCharacter, Count:=-1
            Else
                'sonst den Bereich bis ans Ende des Ursprungbereichs erweitern.
                rngErgebnisBereich.End = rngSuchBereich.End
            End If
        End If
    Else
        rngErgebnisBereich.End = rngSuchBereich.End
    End If
Loop
End Sub

Function BegriffSuchen(strSuchBegriff As String, ByRef rngSuchBereich As Word.Range) _
    As Boolean
    Dim bGefunden As Boolean
    With rngSuchBereich.Find
        .Text = strSuchBegriff
        bGefunden = .Execute
    End With
    BegriffSuchen = bGefunden
End Function

Private Sub GefundeneStelleBearbeiten(strNeuerText As String, ByRef rng As Word.Range)
    rng.InsertAfter strNeuerText
    rng.Collapse Direction:=wdCollapseEnd
End Sub

```



Die Beispieldatei *Bsp06\_03\_Find.doc* finden Sie auf der CD-ROM im Ordner *\Beispiele\Kap06*.

## Das ganze Dokument mit VBA durchsuchen

Die Frage steht noch offen, wie wir Word dazu bringen, das gesamte Dokument nach einem Suchbegriff zu durchsuchen und nicht nur die gegenwärtige Story (Dokumentteil). Es ist durchaus möglich, dass der Suchbegriff sich nicht nur im Dokumenttext, sondern auch in den Kopf- und Fußzeilen, Fußnoten oder Endnoten befindet. Dazu ist eine Schleife notwendig, die alle Stories im Dokument einbindet.

**HINWEIS** Die StoryRange-Auflistung wird im Abschnitt »Bereiche im Dokument: Das StoryRanges-Objekt« in diesem Kapitel eingehender diskutiert.

In Listing 6.37 bzw. Listing 6.38 sehen Sie in der Prozedur *GanzesDokumentDurchsuchen*, wie eine solche Schleife zusammengesetzt wird. Jede Story der Auflistung StoryRanges wird angesprochen: For Each sty In doc.StoryRanges. Das genügt jedoch nicht, da eine Story mehrere Unterbereiche umfassen kann. Dies ist der Fall, wenn ein Dokument aus mehreren Abschnitten mit eigenen Kopf- und Fußzeilen besteht. Jede Kopf- bzw. Fußzeile ist ein gesonderter StoryRange. Deshalb muss auch kontrolliert werden, ob ein NextStoryRange angesprochen werden kann. Dieser Test wird in einer Do-Schleife ausgeführt, bis in dieser Story kein StoryRange mehr vorhanden ist.

Beachten Sie den Einsatz der Eigenschaft TextRetrievalMode.IncludeFieldCodes. Damit können Pfadangaben in Feldcodes gesucht und ersetzt werden, ohne diese im Dokument anzeigen zu müssen. Diese Eigenschaft, sowie IncludeHiddenText, ermöglichen ein verfeinertes Durcharbeiten eines Bereichsinhalts.

**Listing 6.37** Alle Dokumentkomponenten durchsuchen

```
'Alle Vorkommen eines alten Pfadnamens durch den aktuellen ersetzen.
'Hauptsächlich nützlich für Hyperlink, IncludeText, IncludePicture
'und Link-Feldfunktionen
Sub GanzesDokumentDurchsuchen()
    Dim doc As Word.Document
    Dim sty As Word.Range
    Dim strSuchbegriff As String
    Dim strErsatzbegriff As String

    Set doc = ActiveDocument
    'Für Pfadnamen außerhalb einer Feldfunktion
    'strSuchbegriff = "\\AlterServer\Dokumente\Mein Projekt\"
    'strErsatzbegriff = "\\NeuerServer\Projekte\Projekt1\"
    'Für Pfadnamen innerhalb Feldfunktionen (doppelte Backslashes!)
    strSuchbegriff = "C:\\Test\\"
    strErsatzbegriff = Replace(ThisDocument.Path, "\", "\\") & "\\"

    For Each sty In doc.StoryRanges
        AlleInstanzenErsetzen sty, strSuchbegriff, strErsatzbegriff
        Do While Not (sty.NextStoryRange Is Nothing)
            Set sty = sty.NextStoryRange
            AlleInstanzenErsetzen sty, strSuchbegriff, strErsatzbegriff
        Loop
    Next sty
End Sub

Sub AlleInstanzenErsetzen(rng As Word.Range, strSuchbegriff As String, _
    strErsatzbegriff As String)
```



Listing 6.37 Alle Dokumentkomponenten durchsuchen (Fortsetzung)

```

rng.TextRetrievalMode.IncludeFieldCodes = True
rng.TextRetrievalMode.IncludeHiddenText = True
rng.Find.ClearFormatting
rng.Find.Replacement.ClearFormatting
With rng.Find
    .Text = strSuchbegriff
    .Replacement.Text = strErsatzbegriff
    .Forward = True
    .Wrap = wdFindContinue
    .Format = False
    .MatchCase = True
    .MatchWholeWord = True
    .MatchWildcards = False
    .MatchSoundsLike = False
    .MatchAllWordForms = False
End With
rng.Find.Execute Replace:=wdReplaceAll
rng.Fields.Update
End Sub

```

Listing 6.38 (NET): Auch für *Find.Execute* lohnt es sich, »Wrapper«-Funktionen bereitzustellen

```

//Alle Vorkommen eines alten Pfadnamens durch den aktuellen ersetzen
//Hauptsächlich nützlich für Hyperlink, IncludeText, IncludePicture
//und Link-Feldfunktionen.
private void GanzesDokDurchsuchen_CS()
{
    wd.Document doc = wdApp.ActiveDocument;
    //Für Pfadnamen außerhalb einer Feldfunktion.
    //string suchBegriff = "\\\\AlterServer\\Dokumente\\Mein Projekt\\"
    //string ersatzBegriff = "\\\\NeuerServer\\Projekte\\Projekt1\\"
    //Für Pfadnamen innerhalb Feldfunktionen (doppelte Backslashes!).
    string suchBegriff = "C:\\\\Test\\";
    string ersatzBegriff = @di.Parent.Parent.Parent.Parent.FullName + @"\\";
    ersatzBegriff = ersatzBegriff.Replace(@"\", @"\");

    foreach (wd.Range sty in doc.StoryRanges)
    {
        AlleInstanzenErsetzen_CS(sty, suchBegriff, ersatzBegriff);
        while (sty.NextStoryRange != null)
        {
            wd.Range styFolgender = sty.NextStoryRange;
            AlleInstanzenErsetzen_CS(styFolgender, suchBegriff, ersatzBegriff);
            styFolgender = null;
        }
    }
}

private void AlleInstanzenErsetzen_CS(wd.Range rng, string suchBegriff,
    string ersatzBegriff)
{
    rng.TextRetrievalMode.IncludeFieldCodes = true;
    rng.TextRetrievalMode.IncludeHiddenText = true;
    rng.Find.ClearFormatting();

```

**Listing 6.38** (NET): Auch für *Find.Execute* lohnt es sich, »Wrapper«-Funktionen bereitzustellen (*Fortsetzung*)

```
rng.Find.Replacement.ClearFormatting();
object objMissing = System.Reflection.Missing.Value;
object objTrue = (object) true;
object objFalse = (object) false;
object objFindText = (object) suchBegriff;
object objReplaceText = (object) ersatzBegriff;
object objWrapContinue = wd.WdFindWrap.wdFindContinue;
object objReplaceAll = wd.WdReplace.wdReplaceAll;
rng.Find.Execute(ref objFindText, ref objFalse, ref objFalse, ref objFalse, ref objFalse,
    ref objFalse, ref objTrue, ref objWrapContinue, ref objFalse, ref objReplaceText,
    ref objReplaceAll, ref objFalse, ref objFalse, ref objFalse, ref objFalse);
rng.Fields.Update();
}
```



Die Beispieldatei *Bsp06\_04\_Find.doc* finden Sie auf der CD-ROM im Ordner *\Beispiele\Kap06*.

### **Sicherstellen, dass alle Kopf- und Fußzeilen durchsucht werden**

Leider hat die Methode mit *NextStoryRange* einen kleinen Fehler, der einen vorzeitigen Abbruch des Suchvorgangs verursacht. Dies geschieht dann, wenn eine oder mehrere Kopf- oder Fußzeilen keinen Inhalt haben. Falls dies in einem Dokument, das Ihr Code bearbeiten muss, vorkommt, sorgt die Prozedur in Listing 6.39 dafür, dass Word alle Kopf- und Fußzeilen korrekt erkennt. Führen Sie diese Prozedur vor dem Suchvorgang aus.

**Listing 6.39** Kopf- und Fußzeilen für einen Suchvorgang durch das gesamte Dokument vorbereiten

```
Sub AlleKopfUndFusszeilenFuerInhaltTesten()
    Dim oSection As Word.Section
    Dim oHeader As Word.HeaderFooter
    Dim oFooter As Word.HeaderFooter

    'Durch alle Abschnitte Schleifen
    For Each oSection In ActiveDocument.Sections
        'Alle Kopfzeilen des Abschnitts bearbeiten
        For Each oHeader In oSection.Headers
            If Len(oHeader.Range.Text) = 0 Then
                'Falls die Kopfzeile leer ist, ein Leerzeichen eingeben
                oHeader.Range.Text = " "
            End If
        Next oHeader
        'Alle Fußzeilen des Abschnitts bearbeiten
        For Each oFooter In oSection.Footers
            If Len(oFooter.Range.Text) = 0 Then
                'Falls die Fußzeile leer ist, ein Leerzeichen eingeben
                oFooter.Range.Text = " "
            End If
        Next oFooter
    Next oSection
End Sub
```

Listing 6.40 (.NET): Die C#-Version

```
private void AlleKopfUndFusszeilenFuerInhaltTesten_CS()
{
    wd.Document doc = wdApp.ActiveDocument;
    //Durch alle Abschnitte schleifen
    foreach (wd.Section section in doc.Sections)
    {
        //Alle Kopfzeilen des Abschnitts bearbeiten
        foreach (wd.HeaderFooter header in section.Headers)
        {
            if (header.Range.Text.Length<=1)
            {
                //Falls die Kopfzeile leer ist, ein Leerzeichen eingeben
                header.Range.Text = " ";
            }
        }
        //Alle Fußzeilen des Abschnitts bearbeiten
        foreach (wd.HeaderFooter footer in section.Footers)
        {
            if (footer.Range.Text.Length<=1)
            {
                //Falls die Fußzeile leer ist, ein Leerzeichen eingeben
                footer.Range.Text = " ";
            }
        }
    }
}
```



Die Beispieldatei *Bsp06\_04\_Find.doc* finden Sie auf der CD-ROM im Ordner *\Kap06*.

## Text in AutoFormen der Kopf- und Fußzeilen bearbeiten

Obwohl die Prozeduren in Listing 6.37 und Listing 6.39 zuverlässig alle Kopf- und Fußzeilen durchsuchen, finden sie nicht immer alle darin verankerten Textfelder und AutoFormen mit Text. Da diese für sich separate StoryRanges sind, muss ein anderer Weg gefunden werden, um sie anzusprechen. Der Schlüssel hierzu ist die Tatsache, dass alle grafischen Objekte, die sich in der Zeichenebene einer Kopf- oder Fußzeile befinden, über den Bereich der normalen Kopfzeile des ersten Abschnitts des Dokuments zugänglich sind.

Der Code in Listing 6.41 veranschaulicht dieses Prinzip. Er ersetzt den Text »Entwurfsversion vom [beliebigen Datum]« mit »Finalversion vom [heutigen Datum]«. Alle Stories werden auf die übliche Weise in einer Schleife durchlaufen. Wenn der StoryRange der normalen Kopfzeile des ersten Abschnitts vorliegt, wird zusätzlich durch alle AutoFormen mit Text gesucht.

Listing 6.41 Einen Text auch in AutoFormen der Kopf- und Fußzeilen finden und ersetzen

```
Sub EntwurfDurchFinalErsetzen()
    Dim sty As Word.Range
    Dim rngShapeRange As Word.Range
    Dim shp As Word.Shape
    Dim strSuchText As String
    Dim strErsatzText As String
```

**Listing 6.41** Einen Text auch in AutoFormen der Kopf- und Fußzeilen finden und ersetzen *(Fortsetzung)*

```

Dim lAnzahlVorkommen As Long

strSuchText = "Entwurfsversion vom [0-9]{1;2}].[0-9]{1;2}].[0-9]{2;4}"
strErsatzText = "Finalversion vom " & Format(Date, "dd.mm.yyyy")

For Each sty In ActiveDocument.StoryRanges
    'Handelt es sich um die normale Kopfzeile des ersten Abschnitts,
    'werden alle grafischen Objekte mit Text bearbeitet
    If sty.StoryType = wdPrimaryHeaderStory Then
        For Each shp In ActiveDocument.Sections(1).Headers( _
            wdHeaderFooterPrimary).Shapes
            If shp.TextFrame.HasText Then
                Set rngShapeRange = shp.TextFrame.TextRange
                If AlleInstanzenErsetzenMitMustervergleich( _
                    rngShapeRange, strSuchText, strErsatzText) Then
                    lAnzahlVorkommen = lAnzahlVorkommen + 1
                End If
            End If
        Next shp
    End If
    'den StoryRange bearbeiten
    If AlleInstanzenErsetzenMitMustervergleich( _
        sty, strSuchText, strErsatzText) Then
        lAnzahlVorkommen = lAnzahlVorkommen + 1
    'mit allen Unterbereichen
    Do While Not (sty.NextStoryRange Is Nothing)
        Set sty = sty.NextStoryRange
        If AlleInstanzenErsetzenMitMustervergleich( _
            sty, strSuchText, strErsatzText) Then
            lAnzahlVorkommen = lAnzahlVorkommen + 1
        End If
    Loop
Next sty
MsgBox "Der Suchbegriff wurde " & CStr(lAnzahlVorkommen) & " Male ersetzt.", _
    vbInformation + vbOKOnly
End Sub

Function AlleInstanzenErsetzenMitMustervergleich(rng As Word.Range, _
    ByVal strSuchbegriff As String, ByVal strErsatzbegriff As String) As Boolean

    rng.Find.ClearFormatting
    rng.Find.Replacement.ClearFormatting
    With rng.Find
        .Text = strSuchbegriff
        .Replacement.Text = strErsatzbegriff
        .Forward = True
        .Wrap = wdFindContinue
        .Format = False
        .MatchCase = False
        .MatchWholeWord = False
        .MatchWildcards = True
        .MatchSoundsLike = False
        .MatchAllWordForms = False
    End With
    rng.Find.Execute Replace:=wdReplaceAll
    AlleInstanzenErsetzenMitMustervergleich = rng.Find.Found
End Function

```

Listing 6.42 (.NET): Die C#-Version

```

private void EntwurfDurchFinalErsetzen_CS()
{
    wd.Document doc = wdApp.ActiveDocument;
    string suchText = "Entwurfsversion vom [0-9]{1;2}.[0-9]{1;2}.[0-9]{2;4}";
    string ersatzText = "Finalversion vom " + DateTime.Now.ToLongDateString();
    int anzahlVorkommen = 0;

    foreach (wd.Range sty in doc.StoryRanges)
    {
        //Handelt es sich um die normale Kopfzeile des ersten Abschnitts,
        //werden alle grafischen Objekte mit Text bearbeitet
        if (sty.StoryType == wd.WdStoryType.wdPrimaryHeaderStory)
        {
            wd.WdHeaderFooterIndex hp = wd.WdHeaderFooterIndex.wdHeaderFooterPrimary;
            wd.HeaderFooter header = doc.Sections[1].Headers[hp];
            foreach (wd.Shape shp in header.Shapes)
            {
                if (shp.TextFrame.HasText!=0)
                {
                    wd.Range rngShapeTextRange = shp.TextFrame.TextRange;
                    if (AlleInstanzenErsetzenMitMustervergleich_CS(
                        rngShapeTextRange, suchText, ersatzText) )
                    {
                        anzahlVorkommen += 1;
                    }
                }
            }
        }
        //Den StoryRange bearbeiten
        if (AlleInstanzenErsetzenMitMustervergleich_CS(sty, suchText, ersatzText))
        {
            anzahlVorkommen += 1;
            //Mit allen Unterbereichen
            while (! (sty.NextStoryRange==null))
            {
                wd.Range styFolgender = sty.NextStoryRange;
                if (AlleInstanzenErsetzenMitMustervergleich_CS(sty, suchText, ersatzText))
                {
                    anzahlVorkommen += 1;
                }
            }
        }
    }
    MessageBox.Show(String.Format("Der Suchbegriff wurde {0} Male ersetzt.",
        anzahlVorkommen.ToString()));
}

private bool AlleInstanzenErsetzenMitMustervergleich_CS(wd.Range rng,
    string suchBegriff, string ersatzBegriff)
{
    rng.Find.ClearFormatting();
    rng.Find.Replacement.ClearFormatting();
    object objMissing = System.Reflection.Missing.Value;
    object objTrue = (object) true;
    object objFalse = (object) false;
    object objFindText = (object) suchBegriff;
    object objReplaceText = (object) ersatzBegriff;
    object objWrapContinue = wd.WdFindWrap.wdFindContinue;

```

**Listing 6.42** (.NET): Die C#-Version (*Fortsetzung*)

```
object objReplaceAll = wd.WdReplace.WdReplaceAll;
rng.Find.Execute(ref objFindText, ref objFalse, ref objTrue,
    ref objFalse, ref objFalse, ref objTrue, ref objWrapContinue, ref objFalse,
    ref objReplaceText, ref objReplaceAll, ref objFalse, ref objFalse, ref objFalse,
    ref objFalse);
return rng.Find.Found;
}
```



Die Beispieldatei *Bsp06\_05\_Find.doc* finden Sie auf der CD-ROM im Ordner *\Beispiele\Kap06*.

## Bekannte Probleme vermeiden oder beheben

Im Allgemeinen funktioniert *Suchen und Ersetzen* problemlos, wie in den vorangegangenen Abschnitten beschrieben. Es gibt jedoch einige Problemfälle, die nur unter bestimmten Umständen auftreten. Lesen Sie also die folgenden Beschreibungen sorgfältig durch und bauen Sie die vorgeschlagene Lösung gegebenenfalls in Ihre Anwendung ein.

### Word schließt die Suche mit Mustervergleich nicht ab

Bei einer Suche mit der Option *Platzhalterzeichen verwenden* kommt es ab und zu vor, dass Word den Suchvorgang nicht beenden kann. In der Benutzerschnittstelle wird weder Text markiert, noch die Meldung eingeblendet, dass der Suchvorgang beendet ist. Dialogfeld und Dokument verhalten sich offensichtlich trotzdem normal; der Benutzer kann weiterarbeiten.

Schuld daran ist eine Textstelle im Dokument, die aus einem unvollständigen Vergleich resultiert. Word findet den Anfang des Suchbegriffs, merkt ihn vor, erreicht das Dokumentende und ... kommt nicht weiter.

Das Ganze wäre nicht weiter schlimm und kaum erwähnenswert, wenn bei der Ausführung des Suchvorgangs als Teil eines Makros Word (und zwar alle Versionen) nicht abstürzen würde.

Um das Problem nachzuvollziehen und die Prozedur in Listing 6.43 zu testen, geben Sie einige »<>«-Paare mit Text dazwischen ein. Am Ende des Dokuments fügen Sie nur die öffnende Klammer »<« ein, ohne eine abschließende »>«. Die Suche mit Mustervergleich sucht eine öffnende Spitzenglammer, gefolgt von beliebigen Buchstaben (solange es sich nicht um eine schließende Spitzenglammer handelt) und einer schließenden Spitzenglammer.

**Listing 6.43** Beispielcode für die Suche mit Mustervergleich

```
Sub UnvollständigenVergleichAbfangen()
    Const errUnvollständigerVergleich As Long = vbObjectError + 1001
    Dim strErrUnvollständigerVergleich As String
    strErrUnvollständigerVergleich = "Word kommt wegen eines unvollständigen Mustervergleichs nicht weiter."
    On Error GoTo Err_Handler

    Selection.HomeKey wdStory
    With Selection.Find
        .Text = "<[!>]@"
    End With
End Sub
```

Listing 6.43 Beispielcode für die Suche mit Mustervergleich (Fortsetzung)

```

.Replacement.Text = ""
.Forward = True
.Wrap = wdFindStop
.Format = False
.MatchCase = False
.MatchWholeWord = False
.MatchWildcards = True
.MatchSoundsLike = False
.MatchAllWordForms = False
End With

Do While Selection.Find.Execute()
'Schleife beenden, wenn sich Word am Ende des Dokuments aufhängt
If Selection.End = 0 Then Err.Raise errUnvollständigerVergleich

'An dieser Stelle den gefundenen Text weiterbearbeiten. Beispiel:
Selection.Text = ">>"
Selection.Collapse wdCollapseEnd
Loop
Exit Sub

Err_Handler:
Select Case Err.Number
Case errUnvollständigerVergleich
MsgBox strErrUnvollständigerVergleich, vbCritical + vbOKOnly
'Alle Einstellungen zurücksetzen
With Selection.Find
.ClearFormatting
.Replacement.ClearFormatting
.Text = ""
.Replacement.Text = ""
.Forward = True
.Wrap = wdFindStop
.Format = False
.MatchCase = False
.MatchWholeWord = False
.MatchWildcards = False
.MatchSoundsLike = False
.MatchAllWordForms = False
End With
Case Else
MsgBox "Fehler: " & Err.Number & vbCr & _
Err.Description
End Select
End Sub

```

## Word nach erfolglosem Suchvorgang mit Mustervergleich zurücksetzen

Unter Umständen versagt das Suchen mit Platzhalterzeichen, wenn zuvor ein ähnlicher Vorgang ohne Erfolg durchgeführt wurde. Dieses Problem kann durch einen unmittelbar darauf folgenden, erfolgreichen Suchvorgang ohne Mustervergleich behoben werden.

Das Makro in Listing 6.44 setzt die Found-Eigenschaft ein, um zu kontrollieren, ob die Suche mit Platzhalterzeichen fehlgeschlagen ist. Falls ja, führt es sofort eine normale Suche nach dem Suchbegriff »^p« (eine Absatzmarke) aus. Da jedes Word-Dokument, ohne Ausnahme, mindestens einen

Absatz enthält, muss dieser Suchvorgang erfolgreich sein. Dadurch wird gewährleistet, dass weitere Suchvorgänge mit Platzhalterzeichen von Word korrekt ausgeführt werden.

**Listing 6.44** Die Suchfunktionalität zurücksetzen



```
Sub AbgesicherterSuchvorgangMitMustervergleich()
    Dim rng As Word.Range

    Set rng = ActiveDocument.Range
    'Suche mit Platzhalterzeichen ausführen
    'Alle Datumsangaben in USA-Format
    rng.Find.Execute
    FindText:="[0-9]{1;2}/[0-9]{1;2}/[0-9]{4;4}", _
    MatchWildcards:=True, Forward:=True
    'Bei Erfolg, weitere Handlungen ausführen, dann Makro abbrechen
    If rng.Find.Found Then
        'Weitere Handlungen kommen hier
        Exit Sub
    End If

    'Sonst Words Suchfunktionalität zurücksetzen
    rng.Find.Execute FindText:="^p", MatchWildcards:=False
End Sub
```

## Suche mit Platzhalterzeichen nach Benutzereingriff wieder ermöglichen

In Word bleibt ein Suchvorgang mit Platzhalterzeichen erfolglos, wenn der Benutzer zuvor folgende Handlungen ausgeführt hat:

- irgendeinen Suchvorgang, egal welcher Art,
- diesen Vorgang mit dem Browserobjekt (durch Betätigen von  + **F4**, **Strg** +  oder der Schaltflächen unterhalb der vertikalen Bildlaufleiste) wiederholt
- und am Ende des Dokuments die Frage von Word verneint, ob die Suche am Anfang weitergeführt werden soll.

Wenn Word auf diese Weise einen Suchvorgang abbrechen muss, bevor das ganze Dokument durchsucht wurde, bleiben anschließende Suchvorgänge mit Platzhalterzeichen erfolglos, auch wenn passende Textstellen im Dokument vorhanden sind.

Um dieses Problem zu umgehen, rufen Sie die Prozedur in Listing 6.45 auf, bevor Sie eine Suche mit Platzhalterzeichen in VBA ausführen.

**Listing 6.45** Suche mit Platzhalterzeichen nach Benutzereingriff wieder ermöglichen

```
Sub MustervergleichNachBenutzerEingriffBefaeighen()
    With Selection.Find
        .Execute MatchWildcards:=True, FindText:="?"
        Selection.Collapse Direction:=wdCollapseStart
        If Not .Found Then
            CommandBars.FindControl(ID:=313).Execute
            SendKeys String:="{ENTER}", Wait:=True
            SendKeys String:="{ESC}", Wait:=True
            DoEvents
        End If
        Selection.Collapse Direction:=wdCollapseStart
    End With
End Sub
```



Listing 6.45 Suche mit Platzhalterzeichen nach Benutzereingriff wieder ermöglichen (Fortsetzung)

```

        .MatchWildcards = False
        .Text = ""
    End With
End Sub

```

**HINWEIS**

Leider blendet diese Prozedur das Dialogfeld *Suchen und Ersetzen* für kurze Zeit ein. Wir wissen von keinem anderen Weg, dieses Problem zu beheben.

Mehr über den Einsatz des *CommandBar*-Objekts, um Word-Befehle auszuführen, finden Sie in Kapitel 15.

## Das Dialogfeld *Suchen und Ersetzen* zurücksetzen

Sofern Sie in VBA mit dem *Selection*-Objekt die Suchfunktionalität einsetzen, werden die Einstellungen aus Ihrem Code im Dialogfeld *Suchen und Ersetzen* festgehalten. Wenn der Benutzer dies nicht bemerkt, könnte dies zu unerwarteten Ergebnissen und Verärgerung führen.

Mit der Prozedur in Listing 6.46 können Sie am Ende Ihres Makros die Einstellungen des Dialogfelds zurücksetzen. Die Verwendung des *Range*-Objekts ändert die Einstellungen des Dialogfelds nicht.

Listing 6.46 Das Dialogfeld nach der Suchen und Ersetzen-Aktion mit dem *Selection*-Objekt zurücksetzen

```

Sub DialogfeldEinstellungenZuruecksetzen()
    With Selection.Find
        .ClearFormatting
        .Replacement.ClearFormatting
        .Text = ""
        .Replacement.Text = ""
        .Forward = True
        .Wrap = wdFindStop
        .Format = False
        .MatchCase = False
        .MatchWholeWord = False
        .MatchWildcards = False
        .MatchSoundsLike = False
        .MatchAllWordForms = False
    End With
End Sub

```



Der Code für alle Listings im Abschnitt »Bekannte Probleme vermeiden oder beheben« in diesem Kapitel befinden sich in der Beispieldatei *Bsp06\_06\_Find.doc* auf der CD-ROM im Ordner *\Beispiele\Kap06*.

## Browserobjekt zurückstellen

Bevor wir den Abschnitt über *Suchen und Ersetzen* abschließen, machen wir einen kurzen Abstecher, um eine etwas verwirrende, aber durchaus nützliche Folge einer erfolgreichen Suchaktion zu erklären. Plötzlich springen die Tastenkombinationen **[Strg]+[Bild↓]** und **[Strg]+[Bild↑]** nicht mehr die nächste bzw. vorherige Seite an, und die kleinen Doppelpfeile unter der vertikalen Lauf-



leiste sind blau statt schwarz gefärbt. Das liegt am *Browserobjekt*, das in Word 97 neu eingeführt wurde.

Wenn Sie auf das Bällchen zwischen den Doppelpfeilen klicken, blendet Word eine Palette der gültigen Objekttypen (Abschnitt, Grafik, Kommentar, Fußnote, Endnote, Feldfunktion, Tabelle und Überschrift) zur Auswahl ein. Nach einer Operation in einer der Registerkarten des Dialogfelds *Suchen und Ersetzen* wird der entsprechende Objekttyp automatisch ausgewählt, so dass `[Strg]+[Bild↓]` die nächste Instanz anspringt. Im Objektmodell entspricht diese Funktionalität der `GoTo`-Methode des `Selection`-Objekts, aber darum geht es hier nicht.

Oft würde der Anwender diese Funktionalität gerne ausschalten, um weiterhin problemlos mit der Tastatur im Dokument navigieren zu können. In Listing 6.47 zeigen wir, wie das zu erreichen ist. Die drei Prozeduren fangen die Word-eigenen Befehle *Suchen*, *Ersetzen*, und *Gehezu* ab, blenden die Dialogfelder ein und setzen anschließend das Browserobjekt zurück, so dass die Tastenkombinationen der Seitennavigation dienen.

Listing 6.47 Automatisches Einschalten des Browserobjekts verhindern

```
Sub EditFind()
    'BearbeitenSuchen
    On Error Resume Next
    Application.Dialogs(wdDialogEditFind).Show
    Application.Browser.Target = wdBrowsePage
End Sub

Sub EditReplace()
    'BearbeitenErsetzen
    Application.Dialogs(wdDialogEditReplace).Show
    Application.Browser.Target = wdBrowsePage
End Sub

Sub EditGoTo()
    'BearbeitenGeheZu
    Application.Dialogs(wdDialogEditGoTo).Show
    Application.Browser.Target = wdBrowsePage
End Sub
```

#### HINWEIS

Die Autoren danken Bill Coan, Word MVP, für den Inhalt dieses Kapitels.



Die Beispieldatei *Bsp06\_07\_Find.doc* auf der CD-ROM im Ordner *\Beispiele\Kap06*.

## Formatieren mit Stil: Das *Style*-Objekt

Nachdem die Grundlagen der Word-Anwendung und seiner Dokumente besprochen sind, wollen wir uns nun dem Dokumentinhalt widmen. Die weiteren Abschnitte dieses Kapitels befassen sich mit Aspekten des Objektmodells, die mit der Produktivität zu tun haben.

Ein Word-Dokument ist mehr als Text. Text kann man schließlich in jeden Text-Editor eingeben. Dieser jedoch kann Text nicht formatieren. Das Formatieren mit Schriftstilen (z.B. Fett oder Kursiv), Farben und Unterstreichungen beherrscht auch ein einfacheres Textverarbeitungsprogramm wie beispielsweise das im Lieferumfang von Windows enthaltene WordPad. Warum ist Word etwas Besonderes?

Denken wir über die Formatierung einer Überschrift nach. Sie muss sich vom Fließtext klar absetzen, hat also eine andere Schriftart, die größer und fett formatiert ist. Weiter soll der Abstand vor und nach dem Text deutlich sein. Hinzu kommt vielleicht, dass die Überschriften im Dokument durlaufend nummeriert sein müssen. Und letztlich sollen natürlich alle Überschriften im Dokument identisch formatiert sein.

Arbeitet der Benutzer mit WordPad, muss er jede Formatierung einzeln zuweisen. Und er müsste sich zudem an jede Einzelheit erinnern, wenn er die nächste Überschrift formatiert.

## Formatvorlagen

Microsoft Word bietet für diese Aufgabe Formatvorlagen an. In einer einzigen Formatvorlage werden mehrere Formatierungen vereint. Diese werden dann bei Bedarf in einem einzigen Arbeitsschritt dem Text zugewiesen. Formatvorlagen haben also folgende Vorteile:

- Die Formatierung erfolgt schneller.
- Es ist einfacher, im Dokument eine einheitliche Formatierung zu realisieren.

Vom Blickwinkel des Entwicklers gibt es noch weitere Vorteile:

- Vordefinierte Formatvorlagen bedeuten weniger Codezeilen.
- Die Ausführung des Codes ist schneller.
- Die Temp- und Scratch-Datei-Ressourcen von Word werden weniger strapaziert.

Word unterstützt für die Texterstellung dirj zwei Arten von Formatvorlagen: Absatz- sowie Zeichenformatvorlagen. Letztere definieren nur Schrifteigenschaften, während Absatzformatvorlagen nicht nur Schrift, sondern auch Absatz-, Rahmen-, Schattierungs-, Sprachen-, Tabstopp-, Nummerierungs- und Positionsrahmenformatierungen festlegen. Zeichenformatvorlagen überlagern Absatzformatvorlagen.

### HINWEIS

In Word 2002 wurden zwei neue Arten von Formatvorlagen eingeführt: Listen- (Nummerierungs-) und Tabellenformatvorlagen. Diese arbeiten in einem komplexen Zusammenspiel mit Absatz- und Zeichenformatvorlagen und werden eingehender in anderen Abschnitten behandelt.

Im Objektmodell werden Formatvorlagen durch die Styles-Auflistung sowie das Style-Objekt angesprochen, die in der Hierarchie direkt unter dem Document-Objekt stehen.

## Benutzerschnittstellen für Formatvorlagen



Es gibt vier wichtige Benutzerschnittstellen für die Arbeit mit Formatvorlagen in Word: die Drop-down-Liste in der *Format*-Symbolleiste (links in Abbildung 6.17), den Aufgabenbereich *Formatvorlagen und Formatierungen* (der über die Symbolschaltfläche links neben der Liste oder über *Format/Formatvorlagen und Formatierungen* eingeblendet wird) sowie benutzerdefinierte Symbolschaltflächen und Tastenkombinationen, denen über *Extras/Anpassen* auf der Registerkarte *Befehle* Formatvorlagen zugewiesen wurden.

### HINWEIS

Der Aufgabenbereich wurde in Word 2002 eingeführt.

Abbildg. 6.17 Die beiden Schaltflächen für Formatvorlagen befinden sich links in der Symbolleiste *Format*



Zunächst muss uns als Entwickler von Lösungen klar sein, dass der Durchschnitts-Anwender gar nicht realisiert, dass es Formatvorlagen gibt. Falls doch, kennt er nur einige wenige, die in Word mitgeliefert werden, wie *Überschrift 1* bis *Überschrift 3*. Auf die Idee, selbst welche zu erstellen, kommt er erst, wenn er ein Buch gelesen hat oder ihm jemand von der Funktionalität erzählt. Soll der Benutzer mit Formatvorlagen arbeiten, müssen wir es ihm einfach machen, sie zu verwenden, und ihm unter Umständen vielleicht sogar keine andere Wahl lassen.

Die Einführung des Aufgabenbereichs (siehe Kapitel 17) in Word 2002 hilft zu einem gewissen Grad, dem Anwender vordefinierte Formatierungen anzubieten. Da die Formatierungen sichtbar sind und beschreibend benannt werden können, wird der Anwender eher darauf zugreifen, anstatt sich der herkömmlichen Formatierungsbefehle zu bedienen.

Hilfreich sind auch Symbolleisten mit Symbolschaltflächen für Formatvorlagen. In Abbildung 6.18 sehen Sie eine solche, mit deren Hilfe dieses Buch erfasst wurde. Das Objektmodell bietet die Möglichkeit, Symbolleisten zu erstellen, zu positionieren und zu schützen. Mehr darüber lesen Sie in Kapitel 15.

Ihnen ist sicher aufgefallen, dass sich auf jeder Symbolschaltfläche der Abbildung 6.18 ein Tastaturkürzel befindet. Auch diese können programmtechnisch erstellt werden, über die *KeyBindings*-Auflistung, die in Kapitel 16 vorgestellt wird.

Abbildg. 6.18 Jede Symbolschaltfläche ist mit einer Formatvorlage verbunden und weist sie dem markierten Text zu

MsPress Absatzformate	
Standard	Strg+Umschalt+N
Überschrift 1	Alt+1
Überschrift 2	Alt+2
Überschrift 3	Alt+3
Überschrift 4	Alt+4
AbsatzBullet	Strg+D
AbsatzBullet 2.Ebene	Strg+J
AbsatzEinzug	Strg+M
AbsatzNummer 1	Strg+1
AbsatzNummer folgende	Strg+2
Listing mit Unterschrift	Strg+L
Listing ohne Unterschrift	Strg+Ä
Listing Einzug	Strg+Ö
Listingunterschrift	Strg+Alt+L
Tabellenkopf	Alt+Umschalt+K
Tabellentext	Alt+Umschalt+T
Einschub (Kasten)	Strg+E, K
Einschub Achtung	Strg+E, A
Einschub Hinweis	Strg+E, H
Einschub Tipp	Strg+E, T
Einschub Wichtig	Strg+E, W
Verweis auf CD-ROM	Strg+E, C
Marginal-Text	Strg+Alt+M
Marginal-Icon	Strg+Alt+I

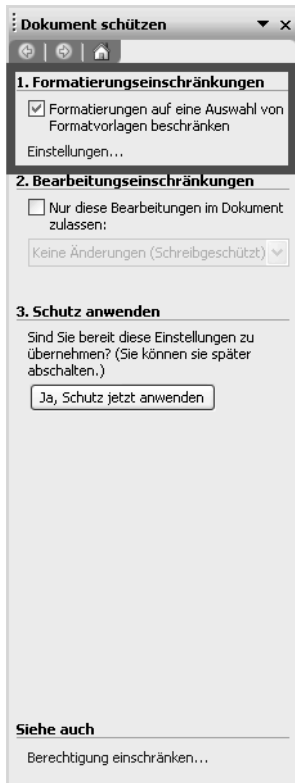
## Formatierungseinschränkungen

Bis einschließlich Word 2002 war es nicht möglich, den Anwender auf den Gebrauch gewisser Formatvorlagen zu beschränken. Wir konnten höchstens mit Code im Dokument nach »fremden« Formatvorlagen suchen und diese entfernen.

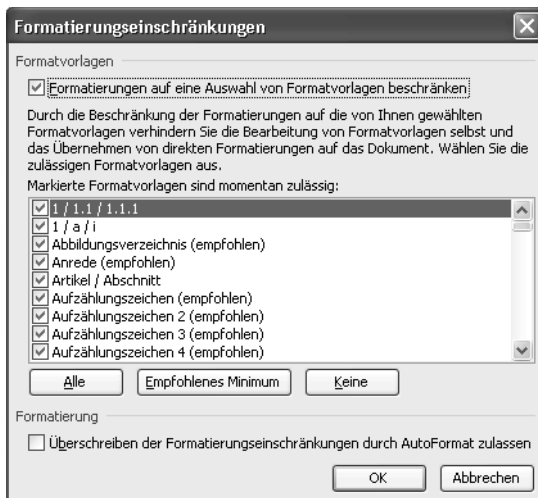
Word 2003 dagegen bietet eine Dokumentschutzoption, um die im Dokument erlaubten Formatierungen festzulegen. In der Benutzerschnittstelle befindet sie sich im Aufgabenbereich *Dokument schützen* (*Extras/Dokument schützen*), dargestellt in Abbildung 6.19. Durch Anklicken des Links *Einstellungen* wird das in Abbildung 6.20 ersichtliche Dialogfeld *Formatierungseinschränkungen* eingeblendet. Als Folge dieser Einschränkung werden direkte Formatierungsbefehle gesperrt, und nur die aktivierten Formatvorlagen-Einträge dürfen im Dokument benutzt werden. Zudem darf der Benutzer weder neue Formatvorlagen definieren, noch bestehende ändern.

Erst nach Aktivierung des Dokumentschutzes (Schritt 3 in Abbildung 6.19) wird die Einschränkung wirksam. Der Schutz kann mit einem Kennwort versehen werden.

Abbildg. 6.19 Über den Aufgabenbereich *Dokument schützen* gelangen wir an die Schutzoptionen für Formatierungen, ...



Abbildg. 6.20 ... und können hier festlegen, welche Formatvorlagen der Benutzer im Dokument einsetzen darf



**WICHTIG** In der Benutzerschnittstelle fragt Word nach, ob im Dokument vorhandene, von den Einstellungen abweichende Formatierungen entfernt werden sollen. Im Objektmodell gibt es dafür kein Gegenstück: bestehende Formatierungen bleiben im Dokument. Ferner ist zu beachten, dass in diesem Fall vorhandene Absatzformatierungen in Kraft bleiben, auch wenn ihre weitere Zuweisung gesperrt ist. Wird in einem solchen Absatz Text eingegeben, übernimmt er die Formatierung dieser Formatvorlage. Das Gleiche gilt jedoch nicht für Zeichenformatierungen; diese sind vollständig gesperrt, auch wenn sie auf einer Zeichenformatvorlage basieren. Wird Text an einer solchen Stelle eingegeben, nimmt er die Schriftarteigenschaften der darunter liegenden Absatzformatvorlage an.

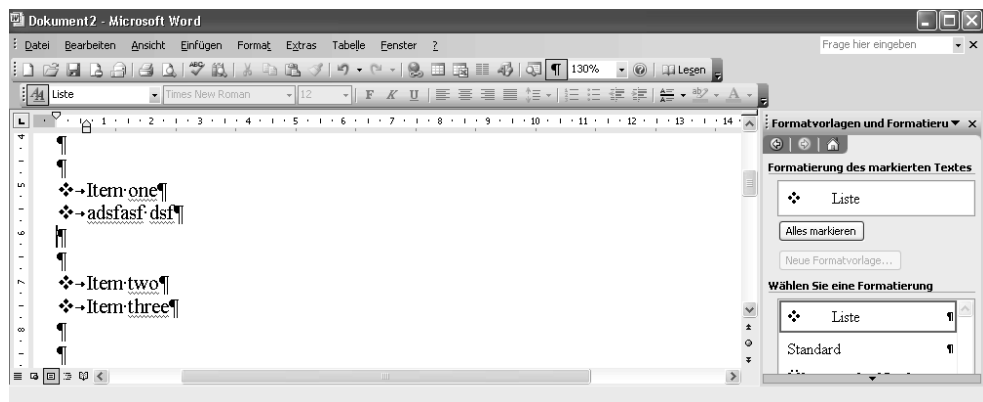
Locked

Im Objektmodell entspricht der obere Teil dieses Dialogfeldes der booleschen Eigenschaft `Locked` des `Style`-Objekts. Ist `Locked` »falsch« (die standardmäßige Einstellung), darf der Benutzer mit der Formatvorlage arbeiten. »Wahr« bedeutet, dass die Formatvorlage in diesem Kontext gesperrt ist.

Das untere Kontrollkästchen steht für die Dokumentoption `AutoFormatOverride`. Ist diese Einstellung aktiviert, dürfen die Optionen *Format/AutoFormat* und *AutoFormat während der Eingabe* ausgeführt werden.

**ACHTUNG** Das Ausschalten von `AutoFormat` funktioniert nicht einwandfrei. Es gibt Handlungen in der Benutzerschnittstelle, die es trotz Sperre aktivieren. Ein Beispiel ist in Abbildung 6.21 veranschaulicht. Obwohl die Formatvorlagen gesperrt sind, hat `AutoFormat` nach zweimaligem Drücken der `↵`-Taste das Symbol der Formatvorlage *Liste* aus dem Text entfernt. Der Drop-down-Liste *Formatvorlage* sowie dem Aufgabenbereich ist zu entnehmen, dass die Formatvorlage der Markierung noch *Liste* und nicht *Standard* ist.

Abbildg. 6.21 AutoFormat umgeht teilweise die Formatierungssperre



Weil `Locked` standardmäßig den Wert »falsch« hat, müssten Sie für alle Formatvorlagen, deren Gebrauch nicht erlaubt ist, die Eigenschaft auf »wahr« setzen oder sie unter Umständen aus dem Dokument löschen. Es ist jedoch einfacher, alle Formatvorlagen zu sperren und nur die erlaubten freizugeben, was am effizientesten in einer Schleife geschieht, wie das Listing 6.48 veranschaulicht.

Zudem entfernt diese Prozedur alle im Dokument vorhandenen Formatierungen, analog zum Verhalten in der Benutzerschnittstelle. Die Eigenschaft `InUse` verrät, ob eine Formatvorlage jemals im Dokument benutzt wurde. Diese wird in der Prozedur aus dem Dokument gelöscht. Ausnahmen

bilden die Word-internen Formatvorlagen »Überschrift 1«, »Überschrift 2«, »Überschrift 3«, »Standard«, »Normale Tabelle«, »Absatz-Standardschriftart« sowie »Keine Liste«. Diese dürfen nicht aus einem Dokument gelöscht werden. Zudem geben diejenigen, die die Grundform der vier Formatvorlagentypen darstellen – »Standard«, »Normale Tabelle«, »Absatz-Standardschriftart«, »Keine Liste« – für InUse immer »wahr« an.

**Listing 6.48** Nur eine Formatvorlage wird im geschützten Word 2003-Dokument freigegeben

```
Sub NurTextZulassen()
    Dim styl As Word.Style
    Dim doc As Word.Document

    Set doc = ActiveDocument
    For Each styl In doc.Styles
        styl.Locked = True
        If styl.InUse Then
            Select Case styl.NameLocal
                Case "Nur Text"
                    'Diese werden im Dokument beibehalten.
                Case "Überschrift 1", "Überschrift 2", "Überschrift 3"
                    'Können nicht gelöscht werden, also ersetzen.
                    Dim rng As Word.Range
                    Set rng = doc.Content
                    With rng.Find
                        .ClearFormatting
                        .Text = ""
                        .Wrap = wdFindStop
                        .Format = True
                        .Style = styl.NameLocal
                        .Replacement.ClearFormatting
                        .Replacement.Text = ""
                        .Replacement.Style = doc.Styles(wdStyleNormal)
                        .Execute Replace:=wdReplaceAll
                    End With
                Case "Standard", "Normale Tabelle", "Absatz-Standardschriftart", _
                    "Keine Liste"
                    'Können nicht gelöscht werden.
                Case Else
                    styl.Delete
            End Select
        End If
    Next
    doc.AutoFormatOverride = False
    'Nur die Formatvorlage 'Nur Text' darf im Dokument benutzt werden.
    doc.Styles(wdStylePlainText).Locked = False
    'direkte Formatierungen entfernen
    doc.Content.Font.Reset
    doc.Protect Password:="", NoReset:=False, Type:=wdNoProtection, _
        UseIRM:=False, EnforceStyleLock:=True
End Sub
```



Listing 6.49 (C#): Die C#-Version

```

private void NurTextZulassen_CS()
{
    object objTrue = (object) true;
    object objFalse = (object) false;
    wd.Document doc = wdApp.ActiveDocument;
    foreach (wd.Style styl in doc.Styles)
    {
        styl.Locked = true;
        if (styl.InUse)
        {
            switch (styl.NameLocal)
            {
                case "Nur Text":
                    //Diese werden im Dokument beibehalten.
                case "Überschrift 1":
                case "Überschrift 2":
                case "Überschrift 3":
                    //Können nicht gelöscht werden, also ersetzen.
                    wd.Range rng = doc.Content;
                    rng.Find.ClearFormatting();
                    rng.Find.Replacement.ClearFormatting();
                    rng.Find.Format = true;
                    object objStyl = (object) styl.NameLocal;
                    rng.Find.set_Style(ref objStyl);
                    object objReplaceStyle = (object)wd.WdBuiltinStyle.wdStyleNormal;
                    rng.Find.Replacement.set_Style(ref objReplaceStyle);
                    object objReplaceText= (object) "";
                    object objFindText = (object) "";
                    object objWrapStop = (object) wd.WdFindWrap.wdFindStop;
                    object objReplaceAll = (object) wd.WdReplace.wdReplaceAll;
                    rng.Find.Execute(ref objFindText, ref objFalse, ref objFalse, ref objFalse,
                        ref objFalse, ref objFalse, ref objTrue, ref objWrapStop, ref objTrue,
                        ref objReplaceText, ref objReplaceAll, ref objFalse,
                        ref objFalse, ref objFalse, ref objFalse);
                    break;
                case "Standard":
                case "Normale Tabelle":
                case "Absatz-Standardschriftart":
                case "Keine Liste":
                    //Können nicht gelöscht werden.
                    break;
                default:
                    styl.Delete();
                    break;
            }
        }
    }
    doc.AutoFormatOverride = false;
    //Nur die Formatvorlage 'Nur Text' darf im Dokument benutzt werden.
    object objStyleNurText = wd.WdBuiltinStyle.wdStylePlainText;
    doc.Styles.get_Item(ref objStyleNurText).Locked = false;
    //direkte Formatierungen entfernen
    doc.Content.Font.Reset();
    object objPassword = (object) "";
    doc.Protect(wd.WdProtectionType.wdNoProtection, ref objFalse, ref objPassword,
        ref objFalse, ref objTrue);
}

```

**WICHTIG** Das Sperren einer Formatvorlage gilt nur für die Benutzerschnittstelle. Über das Objektmodell können Sie weiterhin Formatvorlagen erstellen und ändern, ohne den Dokumentschutz aufzuheben.

## Word-interne Formatvorlagen

Ein von der unveränderten *Normal.dot* erstelltes Dokument enthält mehr als 100 vordefinierte Formatvorlagen. Diese decken eine Vielzahl an Bedürfnissen ab, wie etwa Überschriften, Fließtext, Listen, Kopf- und Fußzeile, Beschriftungen und Inhaltsverzeichnisse. Damit der Benutzer erkennt, wofür eine Formatvorlage vorgesehen ist, hat sie einen beschreibenden Namen. In jeder lokalisierten Version tragen die Word-eigenen Formatvorlagen einen Namen in der lokalen Sprache. Wird ein Dokument in einer anderen Sprachumgebung geöffnet (ausschlaggebend ist die Sprache der Menüs), ändert Word automatisch den Formatvorlagennamen in der Benutzerschnittstelle.

Dieses Verhalten ist nicht ohne Nachteile. Problematisch für den Benutzer sind Feldfunktionen, wie *StyleRef*, worin der Formatvorlagename als eine Zeichenkette erscheint. Der Entwickler muss zudem auf seinen Code aufpassen, wenn er sich mit Formatvorlagen befasst, und möglichst nah mit dem Objektmodell arbeiten, wie im Folgenden beschrieben wird.

### PROFITIPP

Um das Problem mit Feldfunktionen zu umgehen, können Dokumentvariablen oder Dokumenteigenschaften im Dokument gespeichert werden, denen der Name einer Formatvorlage als Wert zugewiesen ist. In der Feldfunktion wird dann statt des Namens eine *DocVariable*- bzw. *DocProperty*-Feldfunktion eingefügt, die den Namen übergibt. Beim Öffnen des Dokuments kann eine Prozedur den Umgebungsnamen feststellen und den Wert der Variablen bzw. Eigenschaft ändern. Somit muss die Änderung an nur einer Stelle stattfinden, auf eine Art, die für den Benutzer transparent ist. Mehr zu Dokumentvariablen und -Eigenschaften lesen Sie in Kapitel 12. Ein Beispiel für dieses Vorgehen ist im Abschnitt »Feldfunktionen« in diesem Kapitel beschrieben.

Name-  
Local

Eine bestimmte Formatvorlage wird mit einem Indexwert angesprochen. Dabei kann es sich um eine Ganzzahl (Position in der Auflistung) handeln, um den Namen, wie er in der Umgebung erscheint (*NameLocal*-Eigenschaft), sowie, für Words interne Formatvorlagen, um einen *WdBuiltinStyle*-Konstantwert. Eine Liste der *WdBuiltinStyle*-Konstantwerte finden Sie im Objektkatalog des VB-Editors; die Enumeration entspricht in ungefähr den englischen Namen. Um herauszufinden, welcher Konstantwert zu welcher Formatvorlage passt:

```
MsgBox ActiveDocument.Styles(wdStyleNormal).NameLocal
'Gibt "Standard" zurück in einer deutschen Umgebung
'Gibt "Normal" zurück in einer englischen Umgebung
```

Arbeiten Ihre Anwendung mit Words internen Formatvorlagen, sollten Sie wo immer möglich statt einer Zeichenkette die *WdBuiltinStyle*-Konstantwerte benutzen, um eine Formatvorlage zu identifizieren.

BuiltIn

Um festzustellen, welche Formatvorlagen in einem Dokument Word-interne sind, wird die *BuiltIn*-Eigenschaft gebraucht. Um alle nicht Word-internen Formatvorlagen aufzulisten:

Listing 6.50 Alle nicht Word-internen Formatvorlagen in einem neuen Dokument auflisten

```

Sub AlleNichtWordFVAuflisten()
    Dim doc As Word.Document
    Dim docNeu As Word.Document
    Dim styl As Word.Style
    Dim rng As Word.Range

    Set doc = ActiveDocument
    Set docNeu = Documents.Add
    Set rng = docNeu.Content
    For Each styl In doc.Styles
        If Not styl.BuiltIn Then
            rng.InsertAfter styl.NameLocal & vbCr
        End If
    Next
End Sub

```

Listing 6.51 Listing 6.51 (.NET): Die C#-Version

```

private void AlleNichtWordFVAuflisten_CS()
{
    object objMissing = System.Reflection.Missing.Value;
    wd.Document doc = wdApp.ActiveDocument;
    wd.Document docNeu = wdApp.Documents.Add(ref objMissing, ref objMissing,
        ref objMissing, ref objMissing);
    wd.Range rng = docNeu.Content;
    foreach (wd.Style styl in doc.Styles)
    {
        if(! styl.BuiltIn)
        {
            rng.InsertAfter(styl.NameLocal + "\n");
        }
    }
}

```



Die Beispieldatei *Bsp06\_01\_Style.doc* finden Sie auf der CD-ROM im Ordner *\Beispiele\Kap06*.

Die in einem Word-Dokument angezeigten Namen benutzerdefinierter Formatvorlagen bleiben im Gegensatz zu den Word-internen statisch. Wenn Sie für gemischte Sprachgebiete programmieren, kommt eher die Frage auf, wie allen Benutzern möglichst bedeutsame Namen zur Verfügung gestellt werden können.

Eine Möglichkeit ist, beim Öffnen eines Dokuments über die Eigenschaft `Application.Language` festzustellen, welche Sprache aktiv ist. Darauf basierend wird die `NameLocal`-Eigenschaft der Formatvorlagen angepasst, bevor das Dokument für die Bearbeitung freigegeben wird.

Handelt es sich um nur wenige Sprachen und Namen, können diese im Code geschrieben und verwaltet werden. Für größere Anwendungen empfiehlt es sich, eine »Äquivalenz-Tabelle« als Ressourcendatei hinzuzuziehen, ob als INI-Datei, Datenbank, Excel-Arbeitsmappe oder XML-Datei, überlassen wir dem Entwickler. Das Beispiel in Listing 6.52 führt die Angaben im Code auf. Die `AutoOpen`-Prozedur wird beim Öffnen des Dokuments automatisch ausgeführt.

**Listing 6.52** Die Namen von Formatvorlagen der Umgebungssprache anpassen

```

Private Const m_LANZV As Long = 2
Private Const m_LANZSPRACHEN As Long = 3
Private Const INDEX_DEUTSCH As Long = 0
Private Const INDEX_ENGLISCH As Long = 1
Private Const INDEX_FRANZ As Long = 2

Sub AutoOpen()
    Dim lSpracheNeu As Long
    Dim lSpracheAlt As Long
    Dim doc As Word.Document
    Dim vrb1Sprache As Word.Variable

    Set doc = ActiveDocument
    lSpracheNeu = Application.Language
    'Die zuletzt benutzte Sprache mit der der Umgebung vergleichen
    'Wenn anders, die Formatvorlagen anpassen
    Set vrb1Sprache = doc.Variables("Sprache")
    lSpracheAlt = CLng(vrb1Sprache.Value)
    If lSpracheNeu <> lSpracheAlt Then
        FVAnpassen doc, lSpracheNeu, lSpracheAlt
        vrb1Sprache.Value = CStr(lSpracheNeu)
    End If
End Sub

Private Sub FVAnpassen(ByRef doc As Word.Document,
    ByVal lSpracheNeu As Long, ByVal lSpracheAlt As Long)

    Dim aFVNamen(m_LANZSPRACHEN - 1, m_LANZV - 1) As Variant
    Dim lZaehler As Long
    Dim lIndexAlt As Long
    Dim lIndexNeu As Long
    Dim sFVAlt As String
    Dim sFVNeu As String

    'Ein Array mit den Namen der Formatvorlagen bestücken.
    SprachenListeFuellen aFVNamen()
    'Element der ersten Dimension des Arrays ermitteln.
    lIndexAlt = SprachIndexHolen(lSpracheAlt)
    lIndexNeu = SprachIndexHolen(lSpracheNeu)

    'Durch das Array schleifen und den Formatvorlagennamen
    'für alte und neue Sprache ermitteln und im Dokument umbenennen
    For lZaehler = LBound(aFVNamen, 2) To UBound(aFVNamen, 2)
        sFVAlt = aFVNamen(lIndexAlt, lZaehler)
        sFVNeu = aFVNamen(lIndexNeu, lZaehler)
        doc.Styles(sFVAlt).NameLocal = sFVNeu
    Next
End Sub

Private Sub SprachenListeFuellen(ByRef aFVNamen() As Variant)
    'Die erste Dimension gibt die Sprache an
    'Die zweite enthält den Formatvorlagennamen (Konstantwert)
    aFVNamen(INDEX_DEUTSCH, 0) = "TestFV1"
    aFVNamen(INDEX_DEUTSCH, 1) = "TestFV2"
    aFVNamen(INDEX_ENGLISCH, 0) = "TestStyle1"
    aFVNamen(INDEX_ENGLISCH, 1) = "TestStyle2"

```

**Listing 6.52** Die Namen von Formatvorlagen der Umgebungssprache anpassen (Fortsetzung)

```

aFVNamen(INDEX_FRANZ, 0) = "Test1"
aFVNamen(INDEX_FRANZ, 1) = "Test2"
End Sub

Private Function SprachIndexHolen(ByVal lSprache As Long) As Long
    Dim lIndex As Long

    Select Case lSprache
        Case 1031, 3079, 5127, 4103, 2055
            lIndex = INDEX_DEUTSCH
        Case 4108, 1036, 5132
            lIndex = INDEX_FRANZ
        Case 1033, 2057, 615
            lIndex = INDEX_ENGLISCH
        'Unerwartete Sprache
        Case Else
            End Select
        SprachIndexHolen = lIndex
    End Function

```



Die Beispieldatei *Bsp06\_02\_Style.doc* finden Sie auf der CD-ROM im Ordner *\Beispiele\Kap06*.

## Formatvorlagen erstellen und modifizieren

Meistens werden Formatvorlagen in Dokumentvorlagen erstellt und gespeichert. Alle von der Dokumentvorlage erstellten Dokumente erben automatisch deren Formatvorlagen. Deshalb kommt es im Entwickler-Alltag vergleichsweise selten vor, dass Formatvorlagen mit Code erstellt werden müssen, außer wenn die Anwendung ein Dokument ganz neu aufbauen soll.

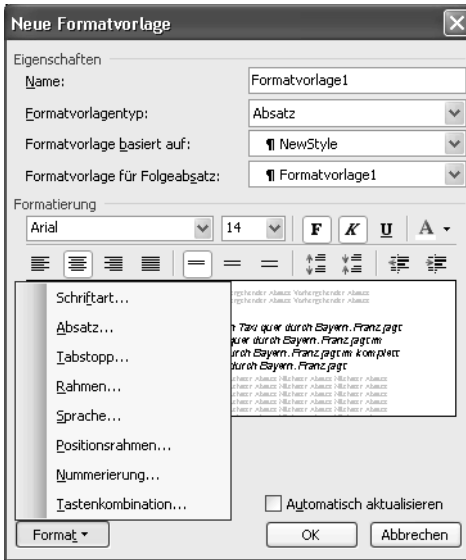
Add-  
Methode


Um einem Dokument eine neue Formatvorlage hinzuzufügen, setzen wir die Add-Methode des Style-Objekts ein: `styl = doc.Styles.Add(Name, [Type])`. Seit Word 2002 gibt es vier mögliche `WdStyleType`-Konstantwerte (Word 2000 erkennt nur die beiden ersten): `wdStyleTypeParagraph` (Absatz), `wdStyleTypeCharacter` (Zeichen), `wdStyleTypeList` (Nummerierung) oder `wdStyleTypeTable` (Tabelle).

In der Benutzeroberfläche übernimmt eine neue Formatvorlage die Eigenschaften der Markierung und basiert auf deren Formatvorlage. Dies ist anders als bei der Automatisierung, wo sie standardmäßig auf der Formatvorlage »Standard« basiert und deren Eigenschaften übernimmt.

Die weitere Festlegung findet im Dialogfeld *Formatvorlage erstellen* (Abbildung 6.22) mit den Menüs unter der Schaltfläche *Format* statt. Da diese den Dialogfeldern des Menüs *Format* in der Menüleiste entsprechen, ist das automatische Erstellen von Formatvorlagen ähnlich der Formatierung von Text, wie auch das Listing 6.53 zeigt.

Abbildg. 6.22 Das Dialogfeld, um eine Formatvorlage zu erstellen oder neu zu definieren



Darin werden zuerst alle Word-eigenen Formatvorlagen gesperrt und anschließend zwei neue Formatvorlagen erstellt. Bitte beachten Sie, dass die erste Formatvorlage, »Bericht Fliesstext«, automatisch der zweiten, »Bericht Überschrift«, folgt, wenn die -Taste gedrückt wird (NextParagraphStyle-Eigenschaft). Basiert eine Formatvorlage auf einer anderen oder folgt eine einer anderen, muss diese Formatvorlage schon vorhanden sein, sonst erfolgt eine Fehlermeldung. Deshalb werden diese Formatvorlagen in dieser Reihenfolge definiert.

Die Definitionen werden in einer getrennten Prozedur den Formatvorlagen zugewiesen. Aus Platzgründen haben wir für das Beispiel nur einige der möglichen Eigenschaften als Argumente von *NeueAbsatzFormatvorlage* aufgelistet, um eine Idee der Vorgehensweise zu liefern. Es ist erkennbar, dass die Eigenschaften auf genau die gleiche Art und Weise benutzt werden, als würden Sie mit einem Range- oder Selection-Objekt arbeiten. Sie dürfen alle Formatierungsbefehle benutzen, die unter dem Menü *Format* des Dialogfeldes *Formatvorlage erstellen* in Abbildung 6.22 ersichtlich sind.

Die Eigenschaften, die Sie nicht festlegen, übernimmt die neue Formatvorlage von der Formatvorlage, auf der sie basiert. Basiert sie auf keiner, beginnt sie mit denen der »Standard«-Formatvorlage.

Am Schluss des Beispiels zeigen wir in der Prozedur *BerichtSchreiben*, wie Text in das Dokument eingegeben sowie mit den Formatvorlagen formatiert und letztlich der Aufgabenbereich *Formatvorlagen und Formatierungen* eingeblendet wird, wie in Abbildung 6.23 zu sehen ist.

Listing 6.53 Formatvorlagen erstellen und definieren

```
Sub NeuerBerichtDok()
    Dim styl As Word.Style
    Dim doc As Word.Document

    Set doc = Documents.Add
    For Each styl In doc.Styles
        styl.Locked = True
    
```

Listing 6.53 Formatvorlagen erstellen und definieren (Fortsetzung)

```

Next
Set styl = Nothing
Set styl = doc.Styles.Add(Name:="Bericht FließText", Type:=wdStyleTypeParagraph)
NeueAbsatzFormatvorlage styl, "", 12, False, False, wdAlignParagraphLeft, _
    1.5, 1.5, , False, "", styl.NameLocal, False
Set styl = Nothing
Set styl = doc.Styles.Add(Name:="Bericht Überschrift 1", Type:=wdStyleTypeParagraph)
NeueAbsatzFormatvorlage styl, "Verdana", 16, True, , _
    wdAlignParagraphCenter, 6, 12, , True, "", "Bericht FließText", False
doc.Protect wdNoProtection, False, "", False, True
BerichtSchreiben doc
Application.TaskPanels(wdTaskPaneFormatting).Visible = True
End Sub

Private Sub NeueAbsatzFormatvorlage(styl As Word.Style, Optional fontName As String, _
    Optional fontSize As Single, Optional fontBold As Boolean, Optional fontItalic, _
    Optional paraAlignment As Long, Optional paraSpaceAfter As Single, _
    Optional paraSpaceBefore As Single, Optional paraLineSpacing As Single, _
    Optional paraBorders As Boolean, Optional baseStyle As Variant, _
    Optional nextStyle As Variant, Optional styleLocked As Boolean) _

    styl.baseStyle = baseStyle
    styl.NextParagraphStyle = nextStyle
    styl.Font.Name = fontName
    styl.Font.Size = fontSize
    styl.Font.Bold = fontBold
    With styl.ParagraphFormat
        .Alignment = paraAlignment
        .SpaceAfter = paraSpaceAfter
        .SpaceBefore = paraSpaceBefore
        .Borders.Enable = paraBorders
    End With
    styl.Locked = styleLocked
End Sub

Private Sub BerichtSchreiben(doc As Word.Document)
    Dim rng As Word.Range

    Set rng = doc.Content
    rng.Text = "Bericht Überschrift" & vbCrLf
    rng.Style = "Bericht Überschrift 1"
    rng.Collapse Direction:=wdCollapseEnd
    rng.Text = "Fliesstext im neuen Dokument."
    rng.Style = "Bericht Fliesstext"
End Sub

```

Listing 6.54 (.NET): Die C#-Version

```

private void Listing6_51_Click(object sender, System.EventArgs e)
{
    object objMissing = System.Reflection.Missing.Value;
    object objTrue = (object) true;
    wd.Style stylNeu;
    object objStyleAbsatz = (object) wd.WdStyleType.wdStyleTypeParagraph;
    wd.Document doc = wdApp.Documents.Add(ref objMissing, ref objMissing,

```

**Listing 6.54** (.NET): Die C#-Version (*Fortsetzung*)

```

        ref objMissing, ref objMissing);
    foreach (wd.Style styl in doc.Styles)
    {
        styl.Locked = true;
    }
    stylNeu = doc.Styles.Add("Bericht FließText", ref objStyleAbsatz);
    wd.WdParagraphAlignment paraLinks = wd.WdParagraphAlignment.wdAlignParagraphLeft;
    NeueAbsatzFormatvorlage_CS(stylNeu, "", 12, 0, 0, paraLinks,
        1.5f, 1.5f, 12, 0, "", stylNeu.NameLocal, false);

    stylNeu = null;
    stylNeu = doc.Styles.Add("Bericht Überschrift 1", ref objStyleAbsatz);
    wd.WdParagraphAlignment paraZentriert = wd.WdParagraphAlignment.wdAlignParagraphCenter;
    NeueAbsatzFormatvorlage_CS(stylNeu, "Verdana", 16, -1, 0, paraZentriert,
        6f, 12f, 16, -1, "", "Bericht FließText", false);
    doc.Protect(wd.WdProtectionType.wdNoProtection, ref objTrue,
        ref objMissing, ref objMissing, ref objTrue);
    BerichtSchreiben_CS(doc);
    wdApp.TaskPanes[wd.WdTaskPanes.wdTaskPaneFormatting].Visible = true;
}

private void NeueAbsatzFormatvorlage_CS(wd.Style styl, string fontName, float fontSize,
    int fontBold, int fontItalic, wd.WdParagraphAlignment paraAlignment,
    float paraSpaceAfter, float paraSpaceBefore, float paraLineSpacing,
    int paraBorders, object baseStyle, object nextStyle, bool styleLocked)
{
    object objBaseStyle = (object) baseStyle;
    styl.set_BaseStyle(ref objBaseStyle);
    object objNextStyle = (object) nextStyle;
    styl.set_NextParagraphStyle(ref objNextStyle);
    styl.Font.Name = fontName;
    styl.Font.Size = fontSize;
    styl.Font.Bold = fontBold;
    styl.ParagraphFormat.Alignment = paraAlignment;
    styl.ParagraphFormat.SpaceAfter = paraSpaceAfter;
    styl.ParagraphFormat.SpaceBefore = paraSpaceBefore;
    styl.ParagraphFormat.Borders.Enable = paraBorders;
    styl.Locked = styleLocked;
}

private void BerichtSchreiben_CS(wd.Document doc)
{
    wd.Range rng = doc.Content;
    rng.Text = "Bericht Überschrift\n";
    object stylÜberschrift1 = (object) "Bericht Überschrift 1";
    rng.set_Style(ref stylÜberschrift1);
    object objCollapseEnd = (object) wd.WdCollapseDirection.wdCollapseEnd;
    rng.Collapse(ref objCollapseEnd);
    object stylFließText = (object) "Bericht Fließtext";
    rng.Text = "Fließtext im neuen Dokument.";
    rng.set_Style(ref stylFließText);
}

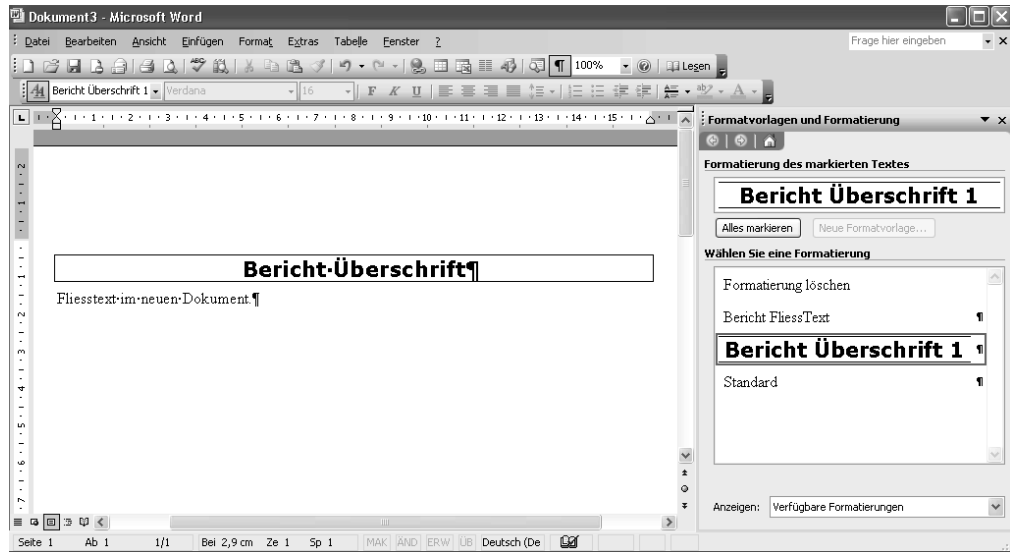
```





Die Beispieldatei *Bsp06\_03\_Style.doc* finden Sie auf der CD-ROM im Ordner *\Beispiele\Kap06*. Bitte beachten Sie, dass das Sperren von Formatvorlagen (die Locked-Eigenschaft sowie der Dokumentschutz für Formatvorlagen) nur in Word 2003 funktioniert. Das Erstellen und Anwenden von Formatvorlagen bleibt in allen Versionen von Word gleich.

**Abbildg. 6.23** Ein neues Dokument mit neuen Formatvorlagen wurde erstellt. Die übrigen Formatvorlagen wurden gesperrt.



Modifiziert wird eine Formatvorlage auf die gleiche Art und Weise wie bei der Erstellung, indem die Formatierungseigenschaften geändert werden.

## »Wilde« Formatvorlagen

In Word 2002 führte Microsoft einige Änderungen der Formatvorlagenverwaltung ein. Die Absicht war, die Anzahl verschiedener Formatierungen im Dokument zu reduzieren. Wie schon erwähnt, war ein Teil dieses Vorhabens der Aufgabenbereich, der sowohl Formatvorlagen als auch Formatierungen auflisten kann.

Eine weitere Maßnahme stellt eine Verbindung zwischen einer Formatvorlage und einer direkten Formatierung her, und zwar dann, wenn der Benutzer markiertem Text – ohne Absatzmarke – eine Absatzformatvorlage zuweist. Dieser Text nimmt die Schrifteigenschaften der Absatzformatvorlage an. Im Hintergrund erstellt Word eine neue Zeichenformatvorlage, die mit der Absatzformatvorlage fest verbunden ist. Seit einem Service Pack für Word 2002 sieht man diese zwar nicht mehr im Aufgabenbereich, wohl aber unter *Organisieren* (Abbildung 6.24). Sie erscheint auch, wenn ein Dokument in Word 2000 oder Word 97 geöffnet und gespeichert wird.

Wird die Definition der Absatzformatvorlage geändert, passt sich die Formatierung der damit verbundenen Zeichenformatvorlage an. Wird umgekehrt die Zeichenformatvorlage geändert, ändert sich die Absatzformatvorlage auch, was für den Benutzer sehr verwirrend sowie frustrierend ist.

**HINWEIS**

Dieses Verhalten ist nicht mit dem Problem zu verwechseln, das oft bei der Neuinstallation von Word 2002 auftritt. In dieser Version ist standardmäßig die Option *Automatisch aktualisieren* (Abbildung 6.22) für die Formatvorlage »Standard« eingeschaltet. Jede im Dokument vorgenommene Formatierung wirkt sich auf die Definition dieser Formatvorlage aus. Und da alle Word-internen Formatvorlagen auf »Standard« basieren, ändert sich die Formatierung im ganzen Dokument.

**Abbildg. 6.24** Eine von Word 2002 oder 2003 automatisch erstellte Zeichenformatvorlage, die mit der Absatzformatvorlage ähnlichen Namens fest verbunden ist



Es ist schwierig, vor allem für den Anwender, eine Verbindung endgültig aus dem Dokument zu entfernen. Probiert er, eine der Formatvorlagen zu löschen, wird die verbundene ebenfalls gelöscht.

Link-  
Style

Um festzustellen, ob zwischen zwei Formatvorlagen eine solche Verbindung besteht, oder um eine solche zu ändern oder herzustellen, benutzt der Entwickler die Eigenschaft `LinkStyle`.

1. Um eine Verbindung zwischen einer Absatz- und einer Zeichenformatvorlage herzustellen, markieren Sie einen Textbereich innerhalb eines Absatzes (ohne die Absatzmarke) und weisen ihm eine Absatzformatvorlage zu.
2. Um diese Verbindung zu sehen und zu testen,
  - ändern Sie die Definition der Absatzformatvorlage,
  - schauen Sie im Dialogfeld *Organisieren* auf der Registerkarte *Formatvorlagen* nach (Menübefehl *Extras/Vorlagen und Add-Ins*, Schaltfläche *Organisieren*),
  - führen Sie diese Codezeile aus:

```
MsgBox ActiveDocument.Styles("Name der Absatzformatvorlage").LinkStyle
```

3. Um die Verbindung aufzuheben, muss die Zeichenformatvorlage mit einer anderen Absatzformatvorlage verbunden werden, und zwar programmtechnisch. Es ist nicht möglich, die Verbindung einfach zu löschen.

```
ActiveDocument.Styles("Name Zeichenformatvorlage").LinkStyle = _
ActiveDocument.Styles(wdStyleNormal)
```


4. Wird eine Verbindung zur Formatvorlage *Standard* hergestellt, ist es jetzt möglich, die »wilde« Zeichenformatvorlage aus dem Dokument zu löschen, ohne dass eine Absatzformatvorlage verloren geht. Dies kann über *Extras/Vorlagen und Add-Ins* nach einem Klick auf die Schaltfläche *Organisieren* erfolgen oder programmtechnisch.

## Zielscheibe Textmarke: Das *Bookmark*-Objekt

Textmarken in Word-Dokumenten erfüllen zwei wichtige Aufgaben. Dem Anwender dienen sie hauptsächlich als Quellenbezeichner für Verweise. Der Entwickler benutzt sie, um Zielbereiche zu bezeichnen.

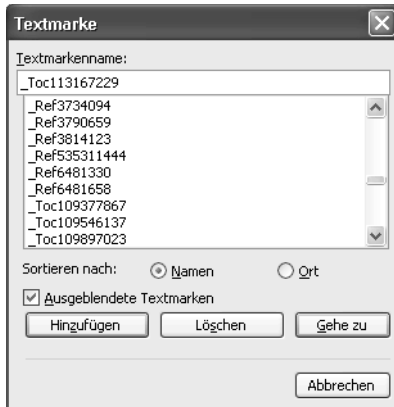
Ein Großteil der dynamischen Funktionalität in Word, wie Inhaltsverzeichnisse und Querverweise, basiert auf Feldfunktionen in Verbindung mit Textmarken, um ihren Textinhalt im Dokument zu identifizieren (siehe den Abschnitt »Feldfunktionen« in diesem Kapitel). Solche Textmarken bleiben unter normalen Umständen dem Benutzer verborgen, da ihre Namen mit einem Unterstrich anfangen. Wenn Sie jedoch nach Aufruf des Menübefehl *Einfügen/Textmarke* das Kontrollkästchen *Ausgeblendete Textmarken* ein-, aus- und nochmals einschalten, erscheinen diese in der Liste wie in Abbildung 6.25. Einträge für das Inhaltsverzeichnis beginnen mit »\_Toc«, solche für Querverweise mit »\_Ref«. Die lange darauf folgende Zahl wird von einem Algorithmus generiert und sorgt dafür, dass keine zwei Zahlen – in mehreren Dokumenten – gleich sein werden.

### TIPP

Steht anstelle eines Verweises die Meldung »Fehler! Verweisquelle konnte nicht gefunden werden.«, wurde eine Textmarke wahrscheinlich versehentlich gelöscht. Umgekehrt, wenn plötzlich unerwartet mehr Text im Verzeichnis oder Querverweis steht, hat der Benutzer wahrscheinlich am Anfang eines Absatzes die -Taste gedrückt, so dass der neue Text sich innerhalb der Textmarke befindet. Im ersten Fall muss der Querverweis neu erstellt werden. Im zweiten muss die Textmarke dem *Ref*-Feld entnommen, der korrekte Text markiert und die Textmarke neu hinzugefügt werden.

Das Dialogfeld in Abbildung 6.25 wird auch genutzt, um Textmarken in Dokumente und Vorlagen als Zielbereiche einzufügen. Automatisierungscode benutzt dann diese Textmarken, um Daten an die gegebene Stelle einzufügen, darin enthaltenen Text zu bearbeiten oder die Einfügemarke für den Benutzer zu positionieren. Textmarkennamen müssen mit einem Buchstaben anfangen, haben eine maximale Länge von 40 Zeichen und dürfen keine Leerzeichen oder Interpunktion enthalten. (Liegt kein gültiger Name im Feld *Textmarkenname* vor, ist die Schaltfläche *Hinzufügen* nicht wählbar.) Bei der Arbeit im Dialogfeld darf der Unterstrich nicht als Anfangsbuchstabe für die Benennung neuer Textmarken benutzt werden.

Abbildg. 6.25 Im Dialogfeld *Textmarke* werden Textmarken verwaltet



**TIPP**

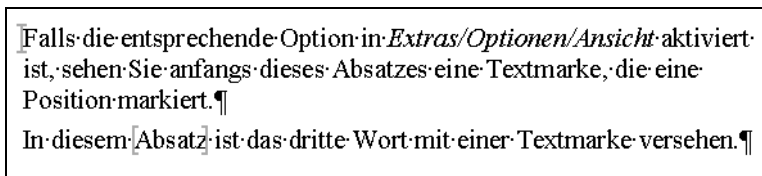
Textmarken können Sie nicht umbenennen. Es muss immer eine neue erstellt und die bestehende gelöscht werden.

Es gibt zwei Arten von Textmarken, wie in Abbildung 6.26 ersichtlich. Die erste markiert eine punktuelle Stelle und sieht wie ein »I « aus; die zweite umfasst ein oder mehrere Zeichen und ist ähnlich einem Paar eckiger Klammern.

**TIPP**

Um die Textmarken im Word-Dokument zu sehen, muss die entsprechende Option aktiviert sein (Menübefehl *Extras/Optionen*, Registerkarte *Ansicht*).

Abbildg. 6.26 Die zwei Arten von Textmarken



Text-  
marken  
einfügen

Wird eine große Word-Datei auf die Dateneingabe mit Automatisierung vorbereitet, ist es mühsam, das Dialogfeld immer wieder öffnen zu müssen. Zudem kann es vorteilhaft sein, die Textmarken zu verbergen. Ein nützliches Werkzeug ist ein kleines Makro mit Eingabeaufforderung für den Textmarkennamen wie in Listing 6.55. Über eine zugewiesene Symbolschaltfläche oder ein Tastaturkürzel ist es schnell aufrufbar. Und über das Objektmodell ist es kein Problem, einen Textmarkennamen mit einem Unterstrich zu beginnen.

Eine Textmarke wird einem Dokument mit der Methode `Document.Bookmarks.Add(Name, Range)` hinzugefügt. Das Argument `Name` stellt den Namen der Textmarke dar; `Range` ist der Bereich, der die Textmarke umfassen wird.

**TIPP**

Mögliche Varianten wären, den markierten Text als Textmarkenname zu übernehmen oder alle Vorkommen einer bestimmten Zeichenkette zu suchen und das nachfolgende Wort als Textmarkenname zu übernehmen.

Listing 6.55

Ein Makro, um das Bestücken eines Dokuments mit Textmarken zu beschleunigen

```

Sub TextmarkeEinfuegen()
    Dim strTextmarkenname As String
    Dim strMarkierungTyp As String
    Dim strText As String
    Dim rng As Word.Range

    On Error GoTo Fehlerbehandlung

    Set rng = Selection.Range
    'Maximal 30 Zeichen werden in der Eingabeaufforderung angezeigt.
    'Sind es mehr, schneiden wir sie ab und fügen ... hinzu.
    If Len(rng) <= 30 Then
        strText = rng.Text
    Else
        strText = Left(rng.Text, 30) & "..."
    End If

    'Die Eingabeaufforderung unterscheidet, ob Zeichen markiert sind.
    If Selection.Type = wdSelectionIP Then
        strMarkierungTyp = "die markierte Position im Dokument"
    ElseIf Selection.Type = wdSelectionNormal Then
        strMarkierungTyp = "den markierten Text " & """" & strText & """"
    Else
        MsgBox "Sie können an dieser Stelle keine Textmarke einfügen"
        Exit Sub
    End If

    Eingabeaufforderung:
    strTextmarkenname = InputBox("Bitte geben Sie der Textmarke für " & _
        strMarkierungTyp & " einen Namen:")

    If Len(strTextmarkenname) > 0 Then
        ActiveDocument.Bookmarks.Add Name:=strTextmarkenname, Range:=rng
    End If

    Exit Sub

Fehlerbehandlung:
    Select Case Err.Number
        Case 5828
            MsgBox "Die Textmarkenname - " & strTextmarkenname & " - ist ungültig." & _
                " Bitte probieren Sie nochmals.", vbOKOnly + vbCritical
            Resume Eingabeaufforderung
        Case Else
            MsgBox Err.Description & vbCr & Err.Number, vbOKOnly + vbCritical
    End Select
End Sub

```

**HINWEIS** Da C# kein Gegenstück zur VBA-InputBox hat, wird in der Prozedur *BenutzerEingabe* ein Formular dynamisch erstellt und eingeblendet. Aus Platzgründen befindet sich dieser Code nur im Beispielprojekt.

Listing 6.56 (.NET): C#-Version

```
private void TextmarkeEinfuegen_CS()
{
    string text = "";
    string markierungTyp = "";
    wd.Selection sel = wdApp.Selection;
    wd.Range rng = sel.Range;
    //Maximal 30 Zeichen werden in der Eingabeaufforderung angezeigt.
    //Sind es mehr, schneiden wir ihn ab und fügen ... hinzu.
    string rngText = "";
    rngText = rng.Text;
    //In C# wird keine Markierung als Null interpretiert.
    if (rngText == null || rngText.Length <= 30)
    {
        text = rng.Text;
    }
    else
    {
        text = rng.Text.Substring(1, 30) + "...";
    }

    //Die Eingabeaufforderung unterscheidet, ob Zeichen markiert
    if (sel.Type == wd.WdSelectionType.wdSelectionIP)
    {
        markierungTyp = "die markierte Position im Dokument";
    }
    else if (sel.Type == wd.WdSelectionType.wdSelectionNormal)
    {
        markierungTyp = "den markierten Text \"" + text + "\"";
    }
    else
    {
        forms.MessageBox.Show("Sie können an diese Stelle keine Textmarke einfügen");
        goto Ende;
    }

    //Formular für die Eingabeaufforderung einblenden
    string textmarkenname = BenutzerEingabe(markierungTyp);
    try
    {
        if (textmarkenname.Length > 0)
        {
            object objRange = rng;
            wdApp.ActiveDocument.Bookmarks.Add(textmarkenname, ref objRange);
        }
    }
    catch (System.Exception ex)
    {
        forms.MessageBox.Show(ex.Message + "\n" + ex.Source + "\n" + ex.InnerException);
    }
    Ende: text="";
}
```



Daten  
schreiben

Die Beispieldatei *Bsp06\_01\_Bookmark.doc* finden Sie auf der CD-ROM im Ordner *\Beispiele\Kap06*.

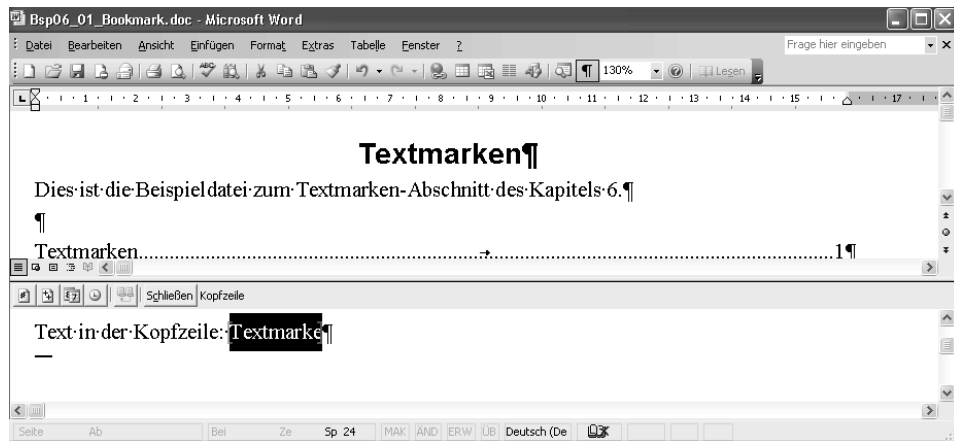
Liegt ein Dokument mit Textmarken vor, soll der Automatisierungscode Daten eingeben oder die Stelle anspringen.

Das Ergebnis des Makrorekorders liefert Code, der das Selection-Objekt einsetzt. Damit wird zur Textmarke gesprungen, was auch geht, solange sich alle Textmarken im gleichen Dokumentteil befinden (obwohl der Bildschirm unruhig wirkt und die Ausführung langsamer ist). Stehen Zielpunkte jedoch in einer Kopf- oder Fußzeile, melden Word 2000 und 2002, dass die Textmarke nicht vorhanden sei.

#### HINWEIS

In Word 2003 wurde das Verhalten geändert. Um Textmarken in Kopf- oder Fußzeilen anzuspringen, wenn diese Ansicht nicht aktiv ist, wechselt Word in die *Normalansicht* und blendet dort das alte Kopf- und Fußzeilen-Fenster aus Word 6.0 ein (Abbildung 6.27).

Abbildg. 6.27 Das alte Kopf- und Fußzeilen-Fenster in der Normalansicht



Besser ist es, Textmarken über das Range-Objekt anzusprechen. Der Bildschirm bleibt ruhig und die Ausführung ist schneller:

```
ActiveDocument.Bookmarks("Name").Range.Text = "Der neue Text."
```

In C# ist es etwas komplizierter:

```
object objName = "test";
wd.Bookmark bkm = doc.Bookmarks.get_Item(ref objName);
bkm.Range.Text = "Der neue Text.";
```

Was genau passiert, wenn Daten in einen Textmarkenbereich geschrieben werden, kommt auf die Art Textmarke an. Markiert sie eine Stelle im Text (sieht wie ein »I« aus), werden die Daten unmittelbar rechts daneben eingefügt. Eventuell vorhandener Text wird nach rechts verschoben. Die Textmarke bleibt am gleichen Ort bestehen (siehe Textmarke »TM2« in Abbildung 6.28).

Umfasst die Textmarke Text, wie die Textmarke »TM3« in Abbildung 6.28, ersetzen die Daten diesen. Zudem wird dadurch die Textmarke gelöscht.

Eine Alternativmethode besteht darin, den neuen Text vor dem Textmarkenbereich einzufügen. In diesem Fall beinhaltet die Textmarke am Schluss den ursprünglichen sowie den neuen Text: Das wurde mit Textmarke »TM1« gemacht; das Ergebnis sehen Sie im unteren Teil der Abbildung 6.28.

Unter Umständen soll die Textmarke nur den neuen Text umfassen. Um dies zu erreichen, muss sie nach Einfügen der Daten neu erstellt werden, wie mit der Textmarke »TM3« in Abbildung 6.28. (Werden mit dieser Methode Daten in eine Positionstextmarke wie »TM2« geschrieben, umfasst am Schluss die Textmarke den neuen Text.)

Die drei Methoden können Sie in Listing 6.57 bzw. Listing 6.58 vergleichen. Diese veranschaulichen zudem, dass es in Word möglich ist, mit der Exists-Eigenschaft zu prüfen, ob eine Textmarke im Dokument vorhanden ist. Die Bookmarks-Auflistung ist die einzige, die über eine Exists-Eigenschaft verfügt.

**Abbildg. 6.28** Daten werden den drei Textmarken mit verschiedenen Methoden hinzugefügt. In jedem Fall bleibt die Textmarke bestehen.

TM1 mit Inhalt TM2 als Position TM3 mit Inhalt

NEUER TEXT TM1 mit Inhalt NEUER TEXT TM2 als Position  
NEUER TEXT

**Listing 6.57** Eine Textmarke nach Einfügen von Daten wieder erstellen, so dass sie den neuen Text umfasst

```
Sub DatenEingeben()
    Dim rng As Word.Range
    Dim doc As Word.Document
    Dim bkm As Word.Bookmark
    Dim strTextmarkenname As String
    Dim strText As String

    strText = "NEUER TEXT"
    Set doc = ActiveDocument
    strTextmarkenname = "TM1"
    'Neuen Text am Anfang des Textmarkeninhalts einfügen.
    doc.Bookmarks(strTextmarkenname).Range.InsertBefore strText

    strTextmarkenname = "TM2"
    'Neuen Text an die Position der Textmarke einfügen.
    doc.Bookmarks(strTextmarkenname).Range.Text = strText

    strTextmarkenname = "TM3"
    'Den gegenwärtigen Inhalt mit neuem Text ersetzen und Textmarke beibehalten.
    'Prüfen, ob die Textmarke vorhanden ist
    If doc.Bookmarks.Exists(strTextmarkenname) Then
        Set bkm = doc.Bookmarks(strTextmarkenname)
        Set rng = bkm.Range
        rng.Text = strText
        'Die Textmarke ist weg
        Set bkm = doc.Bookmarks.Add(strTextmarkenname, rng)
    End If
End Sub
```



Listing 6.58 (.NET): Die C#-Version

```
private void DatenEingeben_CS()
{
    object objMissing = System.Reflection.Missing.Value;
    string text = "NEUER TEXT";
    object objDateiName = "C:\\WordBuch\\Beispiele\\Kap06\\Bsp06_01_Bookmark.doc";
    wd.Document doc = wdApp.Documents.Open(ref objDateiName,
        ref objMissing, ref objMissing, ref objMissing, ref objMissing,
        ref objMissing, ref objMissing, ref objMissing, ref objMissing,
        ref objMissing, ref objMissing, ref objMissing, ref objMissing);
    string textmarkenName = "TM1";
    //Neuen Text am Anfang des Textmarkeninhalts einfügen.
    object objBookmarkName = textmarkenName;
    wd.Bookmark bkm = doc.Bookmarks.get_Item(ref objBookmarkName);
    bkm.Range.InsertBefore(text);

    textmarkenName = "TM2";
    objBookmarkName = textmarkenName;
    bkm = doc.Bookmarks.get_Item(ref objBookmarkName);
    //Neuen Text an die Position der Textmarke einfügen.
    bkm.Range.Text = text;

    bkm=null;
    textmarkenName = "TM3";
    objBookmarkName = textmarkenName;
    //Den gegenwärtigen Inhalt mit neuem Text ersetzen und Textmarke beibehalten.
    //Prüfen, ob die Textmarke vorhanden ist
    if (doc.Bookmarks.Exists(textmarkenName))
    {
        bkm = doc.Bookmarks.get_Item(ref objBookmarkName);
        wd.Range rng = bkm.Range;
        rng.Text = text;
        //Die Textmarke ist weg und muss ersetzt werden
        object objRange = rng;
        bkm = doc.Bookmarks.Add(textmarkenName, ref objRange);
    }
}
```



Die Beispieldatei *Bsp06\_01\_Bookmark.doc* finden Sie auf der CD-ROM im Ordner *\Beispiele\Kap06*.

Daten aus  
Textmar-  
ken lesen

Der Inhalt einer Textmarke wird auf ähnliche Art und Weise gelesen, wie Text hineingeschrieben wird: über die *Range.Text*-Eigenschaft:

```
strTextmarkenInhalt = ActiveDocument.Bookmarks("Name").Range.Text
```

Häufig werden Textmarkeninhalte gelesen und in eine Datenbanktabelle geschrieben. Dies geht besonders gut, wenn Datenfelder und Textmarken die gleichen oder ähnlichen Namen haben. Dann kann entweder durch die *Bookmarks*-Auflistung oder die *Datensatz-Felder* geschleift und mit dem Gegenstück der anderen Auflistung gearbeitet werden. Ein Beispiel hierfür finden Sie in Kapitel 10 im Abschnitt über die Interoperabilität mit Microsoft Access.

**ACHTUNG** Word sortiert die Bookmarks-Auflistung auf zwei verschiedene Weisen: alphabetisch oder entsprechend der Reihenfolge der Textmarken im Dokument. Wenn die Auflistung über das Document-Objekt durchschleift wird, repräsentiert der Indexwert die alphabetische Reihenfolge. Wird sie über das Range-Objekt angesprochen, repräsentiert der Indexwert die Reihenfolge der Textmarken im angegebenen Textbereich.

Vordefinierte Textmarken

Eine Abhandlung dieses Themas ist unvollständig, ohne die vordefinierten Textmarken von Word zu erwähnen. Diese stammen aus den Ur-Zeiten der Word-Anwendung und ermöglichen den Zugriff auf Teile des Dokuments, die im Objektmodell kein Gegenstück haben, wie etwa: Seiten, Zeilen, Textinhalt einer Überschrift. Eine Liste aller vordefinierten Textmarken enthält die Tabelle 6.12. Die meisten davon haben gleichwertige VBA-Anweisungen ersetzt, interessant sind vor allem `\Page`, `\Line` sowie `\HeadingLevel`.

**Tabelle 6.12** Liste der vordefinierten Textmarken, mit gleichwertiger Anweisung des Objektmodells, wo vorhanden

Vordefinierte Textmarke	Beschreibung	Objektmodell-Anweisung
<code>\Sel</code>	Gegenwärtige Markierung	<code>Selection.Range</code>
<code>\PrevSel1</code>	Die zuletzt bearbeitete Stelle	<code>Application.GoBack</code>
<code>\PrevSel2</code>	Die vorletzte bearbeitete Stelle	<code>Application.GoBack</code> zweimal
<code>\StartOfSel</code>	Startpunkt der gegenwärtigen Markierung	<code>Range.Start</code>
<code>\EndOfSel</code>	Endpunkt der gegenwärtigen Markierung	<code>Range.End</code>
<code>\Line</code>	Die aktuelle Zeile bzw. die erste Zeile der aktuellen Markierung. Wenn die Einfügemarke sich am Ende einer Zeile befindet, bei der es sich nicht um die letzte Zeile eines Absatzes handelt, schließt die Textmarke die gesamte nächste Zeile ein.	(Keine gleichwertige Anweisung)
<code>\Char</code>	Aktuelles Zeichen, bei dem es sich um das Zeichen nach der Einfügemarke handelt, wenn nichts markiert ist oder das Zeichen am Anfang der Markierung steht.	<code>Range.Characters(1)</code>
<code>\Para</code>	Aktueller Absatz, bei dem es sich um den Absatz handelt, der die Einfügemarke enthält. Wenn mehrere Absätze markiert sind, handelt es sich um den ersten Absatz der Markierung. Wenn sich die Einfügemarke oder Markierung im letzten Absatz des Dokuments befindet, schließt die Textmarke <code>\Para</code> die Absatzmarke nicht ein.	<code>Range.Paragraphs(1)</code>
<code>\Section</code>	Aktueller Abschnitt, einschließlich des Umbruchs am Ende des Abschnitts, falls vorhanden. Der aktuelle Abschnitt enthält die Einfügemarke oder die Markierung. Enthält die Markierung mehrere Abschnitte, ist die Textmarke <code>\Section</code> der erste Abschnitt in der Markierung.	<code>Range.Sections(1)</code>
<code>\Doc</code>	Gesamter Inhalt des aktiven Dokuments, mit Ausnahme der letzten Absatzmarke	<code>ActiveDocument</code>

**Tabelle 6.12** Liste der vordefinierten Textmarken, mit gleichwertiger Anweisung des Objektmodells, wo vorhanden (*Fortsetzung*)

Vordefinierte Textmarke	Beschreibung	Objektmodell-Anweisung
<code>\Page</code>	Aktuelle Seite, einschließlich des Umbruchs am Ende der Seite, falls vorhanden. Die aktuelle Seite enthält die Einfügemarke. Enthält die aktuelle Markierung mehrere Seiten, ist die Textmarke <code>\Page</code> die erste Seite der Markierung. Befindet sich die Einfügemarke oder Markierung auf der letzten Seite des Dokuments, enthält die Textmarke <code>\Page</code> nicht die letzte Absatzmarke.	(keine gleichwertige Anweisung)
<code>\StartOfDoc</code>	Der Anfang des Dokuments	<code>ActiveDocument.Content.Start</code>
<code>\EndOfDoc</code>	Das Ende des Dokuments	<code>ActiveDocument.Content.End</code>
<code>\Cell</code>	Aktuelle Zelle in einer Tabelle, bei der es sich um die Zelle handelt, welche die Einfügemarke enthält. Sind eine oder mehrere Zellen einer Tabelle in die aktuelle Markierung eingeschlossen, ist die Textmarke <code>\Cell</code> die erste Zelle in der Markierung.	<code>Range.Cells(1)</code>
<code>\Table</code>	Aktuelle Tabelle, bei der es sich um die Tabelle handelt, welche die Einfügemarke oder die Markierung enthält. Schließt die Markierung mehrere Tabellen ein, ist die Textmarke <code>\Table</code> die gesamte erste Tabelle der Markierung. Dies ist auch der Fall, wenn nicht die gesamte Tabelle markiert ist.	<code>Range.Tables(1)</code>
<code>\HeadingLevel</code>	Die Überschrift, welche die Einfügemarke oder die Markierung mit untergeordneten Überschriften und Text enthält. Handelt es sich bei der aktuellen Markierung um Textkörper, schließt die Textmarke <code>\HeadingLevel</code> die vorhergehende Überschrift mit allen Überschriften und allem Text ein, die der Überschrift untergeordnet sind.	(keine gleichwertige Anweisung)

Wichtig beim Umgang mit vordefinierten Textmarken ist, dass sie sich immer auf die gegenwärtige Markierung beziehen. Mit einer gewissen »Unruhe« auf dem Bildschirm muss also gerechnet werden.

Um mit der Textmarke `\Page` den Text irgendeiner Seite einem Bereich zuzuweisen, muss sich die Einfügemarke zuerst auf dieser Seite befinden. Mit den folgenden Codezeilen wird der Text, der sich auf der gleichen Seite wie die erste Tabelle im Dokument befindet, einem Bereich zugewiesen:

```
ActiveDocument.Tables(1).Range.Select
Set rng = Selection.Bookmarks("\Page").Range
```

Der Beispielcode in Listing 6.59 sucht das gegenwärtige Dokument nach der Formatvorlage *Überschrift 3* ab und übernimmt jedes Vorkommen – samt dem folgenden formatierten Text bis zur nächsten höheren Überschriftenebene – in ein neues Dokument.

**Listing 6.59** Da mit vordefinierten Textmarken gearbeitet wird, wird ausnahmsweise mit dem *Selection*- statt dem *Range*-Objekt gesucht

```
Sub AlleEbene3Text()
    Dim rngSuchBereich As Word.Range
    Dim rngZielBereich As Word.Range
    Dim docNeu As Word.Document
    Dim bFound As Boolean

    Set rngSuchBereich = ActiveDocument.Content
    Set docNeu = Application.Documents.Add
    Set rngZielBereich = docNeu.Content
    'Am Dokumentanfang beginnen
    rngSuchBereich.Collapse Direction:=wdCollapseStart
    rngSuchBereich.Select
    Application.ScreenUpdating = false
    With Application.Selection.Find
        .ClearAllFuzzyOptions
        .ClearFormatting
        .MatchAllWordForms = False
        .MatchCase = False
        .MatchSoundsLike = False
        .MatchWholeWord = False
        .MatchWildcards = False
        .Format = True
        .Forward = True
        .Text = ""
        .Wrap = wdFindStop
        .Style = wdStyleHeading3
    End With
    Do
        bFound = Selection.Find.Execute
        If bFound Then
            rngZielBereich.FormattedText = _
                Selection.Bookmarks("\HeadingLevel").Range.FormattedText
            rngZielBereich.Collapse Direction:=wdCollapseEnd
            Selection.Collapse Direction:=wdCollapseEnd
        End If
    Loop While bFound
End Sub
```



Die Beispieldatei *Bsp06\_01\_Bookmark.doc* finden Sie auf der CD-ROM im Ordner *\Beispiele\Kap06*.

## Inhalt mit Tabellen strukturiert darstellen

Tabellen erfüllen in Word zwei wichtige Aufgaben: Sie helfen bei der grafischen Strukturierung und dem Layout des Dokumentinhalts und dienen, wenn auch mit etwas unvollständiger Funktionalität, für Datenauflistungen und Kalkulationen.

Im Word-Objektmodell wird eine Tabelle mit dem *Table*-Objekt dargestellt. Eine Tabelle besteht aus Zeilen (*Rows*-Auflistung sowie das *Row*-Objekt), Spalten (*Columns*-Auflistung sowie das *Column*-Objekt) und Zellen (*Cells*-Auflistung sowie das *Cell*-Objekt). Mit Ausnahme von Spalten stellen

alle diese Objekte eine Range-Eigenschaft zur Verfügung, womit die Formatierung und der Textinhalt manipuliert werden.

Die Tabellenfunktionalität wurde in jeder neuen Word-Version seit Word 97 durch erweiterte Möglichkeiten ergänzt. Hauptziel der meisten dieser Änderungen war, das Verhalten von Webseiten-Tabellen anzubieten. In Laufe der Zeit wurden eingeführt:

- Automatische Anpassung der Zellenbreite an den Textinhalt.
- Schriftgröße anpassen, so dass sich Text in eine bestimmte Breite einpasst.
- Drehung von Text um 90 oder 270 Grad in Tabellenzellen.
- Senkrechte Ausrichtung von Text in einer Tabellenzelle.
- Tabellenbreite als Prozent der Seitenbreite festlegen.
- Abstand zwischen Zellen.
- Verschachtelung mehrerer Tabellen.
- Textfluss um die Tabelle.
- Freie Positionierung von Tabellen auf der Seite.
- Weiterführung einer frei positionierten Tabelle auf der nächsten Seite.
- Textfluss um grafische Objekte, die über einer Tabellenzelle liegen.
- Tabellenformatvorlagen

Diese stufenweise Einführung neuer Funktionalität stellt sowohl Entwickler als auch Benutzer vor einige Probleme, sobald Dokumente in verschiedenen Word-Versionen bearbeitet werden. Logischerweise unterstützen ältere Word-Versionen die neue Funktionalität nicht, so dass Dokumente, die in einer früheren Version geöffnet werden, anders aussehen könnten. Bei Automatisierungscode kommt es gar zu Kompilierungsfehlern, die die Fehlerbehandlung nicht abfangen kann.

Es ist also sehr wichtig, Anwendungen, die für mehrere Word-Versionen vorgesehen sind, in der ältesten dieser Versionen zu entwickeln und gründlich zu testen.

#### PROFITIPP

Für den Entwickler, der VBA-Code in einem Word-Projekt schreibt, bietet sich die Möglichkeit, versionspezifischen Code in ein eigenes Modul zu schreiben. Da VBA den Code eines Moduls erst kompiliert, wenn erstmals eine darin stehende Prozedur aufgerufen wird, ist dies ein probates Mittel, um Unterschiede im Objektmodell in einer Anwendung zu handhaben.

Entwickler, die Word von einer anderen Umgebung aus automatisieren, müssen mit »Late Binding« arbeiten.

In beiden Fällen wird die Eigenschaft `Application.Version` abgefragt, um herauszufinden, welche Word-Version vorliegt.

## Tabellen erstellen

#### Add-Methode

Eine Tabelle wird einem Dokument mit der Add-Methode hinzugefügt. Die Syntax der Add-Methode lautet:

- `Tables.Add(Range, NumRows, NumColumns, DefaultTableBehavior, AutoFitBehavior)`

Range ist der Bereich im Dokument, wo die Tabelle eingefügt wird. NumRows und NumColumns geben die Anzahl der Zeilen und Spalten an. Diese drei Argumente sind erforderlich. Zwei weitere kamen erst in Word 2000 hinzu und sind daher optional. Trotzdem sind sie für den Entwickler von Bedeutung, weil sie das Verhalten der Tabelle in Bezug auf die automatische Spaltenbreite festlegen. DefaultTableBehavior hat zwei mögliche Werte: wdWord8TableBehavior (verhält sich wie Word 97) sowie wdWord9TableBehavior (verhält sich wie in Word 2000 und später). Für AutoFitBehavior gibt es deren drei: wdAutoFitContent (Spaltenbreiten passen sich dem Textinhalt an), wdAutoFitFixed (Spaltenbreiten sind statisch) oder wdAutoFitWindow (Spaltenbreiten passen sich der Breite des Fensters an). Diese treten jedoch nur dann in Kraft, wenn DefaultTableBehavior auf wdWord9TableBehavior festgelegt ist.

Die automatische Anpassung der Zellen- und Spaltenbreite an den Textinhalt wurde in Word 2000, Version 9, eingeführt und ist standardmäßig aktiv. Diese Funktionalität verlangsamt das Layout einer langen Tabelle erheblich, zudem ist das Verhalten oft nicht erwünscht. Sollen die Spalten in einer Tabelle statische Breiten haben, muss DefaultTableBehavior auf wdWord8TableBehavior festgelegt werden.

Auch sonst beklagen sich Entwickler, dass die Erstellung von langen Tabellen relativ langsam ist – egal ob die Zeilen einzeln, nach Bedarf, hinzugefügt und mit Daten gefüllt werden oder ob die Tabelle von vornherein mit der benötigten Anzahl an Zeilen ausgestattet und anschließend mit Daten gefüllt wird.

Die schnellste Methode, eine neue Tabelle zu erstellen und mit Daten zu füllen, ist, die Daten zuerst in einer zeichengetrennten Zeichenkette zusammenzufügen, diese einem Bereich zuzuweisen und den Bereich danach in eine Tabelle umzuwandeln.

Alle drei Methoden können dem Listing 6.60 bzw. dem Listing 6.61 entnommen werden; das Resultat ist in Abbildung 6.29 ersichtlich. Zudem veranschaulichen die Listings folgende Objekte, Eigenschaften und Methoden des Word-Objektmodells:

- Tables.Add
- Row.Cells(index)
- Row.Index
- Rows.Add
- Table.Cell(Zeilenindex, Spaltenindex)
- Range.ConvertToTable

Abbildg. 6.29 Die drei Tabellen wurden mit unterschiedlichen Methoden erstellt. Das Resultat ist das gleiche, nur die Effizienz für längere Tabellen ist unterschiedlich.

Zeile 1, Spalte 1	Zeile 1, Spalte 2	Zeile 1, Spalte 3
Zeile 2, Spalte 1	Zeile 2, Spalte 2	Zeile 2, Spalte 3
Zeile 3, Spalte 1	Zeile 3, Spalte 2	Zeile 3, Spalte 3
¶		
Zeile 1, Spalte 1	Zeile 1, Spalte 2	Zeile 1, Spalte 3
Zeile 2, Spalte 1	Zeile 2, Spalte 2	Zeile 2, Spalte 3
Zeile 3, Spalte 1	Zeile 3, Spalte 2	Zeile 3, Spalte 3
¶		
Zeile 1, Spalte 1	Zeile 1, Spalte 2	Zeile 1, Spalte 3
Zeile 2, Spalte 1	Zeile 2, Spalte 2	Zeile 2, Spalte 3
Zeile 3, Spalte 1	Zeile 3, Spalte 2	Zeile 3, Spalte 3
¶		

**Listing 6.60** Drei mögliche Methoden, um mit Daten gefüllte Tabellen zu erstellen. Die dritte ist mit Abstand die schnellste, wenn die Tabellen lang werden.

```

Sub TabellenEinfuegen1()
    Const lDIM_1 As Long = 2
    Const lDIM_2 As Long = 2
    Dim tbl As Word.Table
    Dim doc As Word.Document
    Dim rng As Word.Range
    Dim row As Word.Row
    Dim lAnzZeilen As Long
    Dim lAnzSpalten As Long
    Dim lZaehlerZeile As Long
    Dim lZaehlerSpalte As Long

    Set doc = Documents.Add
    Set rng = doc.Content
    lAnzZeilen = lDIM_1 + 1
    lAnzSpalten = lDIM_2 + 1

    Dim aDaten(lDIM_1, lDIM_2) As String
    DatenZuweisen aDaten()

    'Zeilen nach Bedarf erstellen und Daten einfügen
    Set tbl = doc.Tables.Add(rng, 1, lAnzSpalten, wdWord8TableBehavior)
    Set row = tbl.Rows(1)
    For lZaehlerZeile = LBound(aDaten, 1) To UBound(aDaten, 1)
        'Daten in die letzte Zeile eingeben.
        For lZaehlerSpalte = LBound(aDaten, 2) To UBound(aDaten, 2)
            row.Cells(lZaehlerSpalte + 1).Range.Text = _
                aDaten(lZaehlerZeile, lZaehlerSpalte)
        Next
        'Neue Zeile nach Bedarf einfügen.
        If row.Index <> lAnzZeilen Then
            Set row = Nothing
            Set row = tbl.Rows.Add
        End If
    Next

    Set rng = BereichNachTabelleFestlegen(tbl)
    'Alle Zeilen erstellen, dann die Daten einfügen.
    Set tbl = Nothing
    Set tbl = doc.Tables.Add(rng, lAnzZeilen, lAnzSpalten, wdWord8TableBehavior)
    For lZaehlerZeile = LBound(aDaten, 1) To UBound(aDaten, 1)
        For lZaehlerSpalte = LBound(aDaten, 2) To UBound(aDaten, 2)
            tbl.Cell(lZaehlerZeile + 1, lZaehlerSpalte + 1).Range.Text = _
                aDaten(lZaehlerZeile, lZaehlerSpalte)
        Next
    Next

    Set rng = BereichNachTabelleFestlegen(tbl)

    'Aus den Daten eine zeichengetrennte Zeichenkette erstellen
    'Diese einfügen und in eine Tabelle umwandeln.
    Dim strDaten As String
    For lZaehlerZeile = LBound(aDaten, 1) To UBound(aDaten, 1)
        For lZaehlerSpalte = LBound(aDaten, 2) To UBound(aDaten, 2)
            strDaten = strDaten & aDaten(lZaehlerZeile, lZaehlerSpalte)
        
```

**Listing 6.60** Drei mögliche Methoden, um mit Daten gefüllte Tabellen zu erstellen. Die dritte ist mit Abstand die schnellste, wenn die Tabellen lang werden. (*Fortsetzung*)

```

        If lZaehlerSpalte <> UBound(aDaten, 2) Then
            strDaten = strDaten & vbTab
        End If
    Next
    If lZaehlerZeile <> UBound(aDaten, 1) Then
        strDaten = strDaten & vbCr
    End If
Next
rng.Text = strDaten
Set tbl = rng.ConvertToTable(vbTab)
End Sub

Private Sub DatenZuweisen(ByRef aDaten() As String)
    Dim lZaehlerZeilen As Long
    Dim lZaehlerSpalten

    For lZaehlerZeilen = LBound(aDaten, 1) To UBound(aDaten, 1)
        For lZaehlerSpalten = LBound(aDaten, 2) To UBound(aDaten, 2)
            aDaten(lZaehlerZeilen, lZaehlerSpalten) = _
                "Zeile " & CStr(lZaehlerZeilen + 1) & _
                ", Spalte " & CStr(lZaehlerSpalten + 1)
        Next
    Next
End Sub

Public Function BereichNachTabelleFestlegen(tbl As Word.Table) As Word.Range
    'Den Bereich nach der Tabelle und eine neue Absatzmarke festlegen
    Dim rng As Word.Range

    Set rng = tbl.Range
    rng.Collapse Direction:=wdCollapseEnd
    rng.InsertParagraphAfter
    rng.Collapse Direction:=wdCollapseEnd

    Set BereichNachTabelleFestlegen = rng
End Function

```

Da sich bei der C#-Automatisierung die Word-Anwendung ohne Dokument öffnet, wird in Listing 6.61 ein neues Dokument erstellt, und dieses an die Prozedur *TabellenEinfuegen\_CS* übergeben. Die Variablen *DIM\_1*, *DIM\_2*, *AnzZeilen* und *AnzSpalten* haben hier andere Werte, als im VBA-Codebeispiel, weil das .NET Framework Datenfelder (Arrays) anders handhabt.

**Listing 6.61** (.NET): Die C#-Version

```

private void Listing6_53_Click(object sender, System.EventArgs e)
{
    object objMissing = System.Reflection.Missing.Value;
    wd.Document doc = wdApp.Documents.Add(ref objMissing, ref objMissing,
        ref objMissing, ref objMissing);
    TabellenEinfuegen_CS(doc);
}

private void TabellenEinfuegen_CS(wd.Document doc)

```



Listing 6.61 (.NET): Die C#-Version (Fortsetzung)

```

const int DIM_1 = 3;
const int DIM_2 = 3;
wd.Range rng = doc.Content;
int AnzZeilen = DIM_1;
int AnzSpalten = DIM_2;
int zaehlerZeile;
int zaehlerSpalte;
string tab = "\t";
string[,] aDaten = new string[DIM_1, DIM_2];
object objMissing = System.Reflection.Missing.Value;

DatenZuweisen(ref aDaten);
//Zeilen nach Bedarf erstellen und Daten einfügen
object objTableBehavior8 = wd.WdDefaultTableBehavior.wdWord8TableBehavior;
object objTableAutoFit = wd.WdAutoFitBehavior.wdAutoFitWindow;
wd.Table tbl = doc.Tables.Add(rng, 1, AnzSpalten, ref objTableBehavior8,
    ref objTableAutoFit);
wd.Row row = tbl.Rows[1];
for(zaehlerZeile = aDaten.GetLowerBound(0);
    zaehlerZeile<= aDaten.GetUpperBound(0); zaehlerZeile++)
{
    //Daten in die letzte Zeile eingeben.
    for (zaehlerSpalte = aDaten.GetLowerBound(1);

        zaehlerSpalte<= aDaten.GetUpperBound(1); zaehlerSpalte++)
    {
        row.Cells[zaehlerSpalte + 1].Range.Text = aDaten[zaehlerZeile, zaehlerSpalte];
    }
    //Neue Zeile nach Bedarf einfügen.
    if (row.Index != AnzZeilen)
    {
        row = null;
        row = tbl.Rows.Add(ref objMissing);
    }
    rng=null;
}
rng = BereichNachTabelleFestlegen(tbl);

//Alle Zeilen erstellen, dann die Daten einfügen.
tbl = null;
tbl = doc.Tables.Add(rng, AnzZeilen, AnzSpalten, ref objTableBehavior8,
    ref objMissing);
for (zaehlerZeile = aDaten.GetLowerBound(0);
    zaehlerZeile <= aDaten.GetUpperBound(0);zaehlerZeile++)
{
    for (zaehlerSpalte = aDaten.GetLowerBound(1);
        zaehlerSpalte <= aDaten.GetUpperBound(1); zaehlerSpalte++)
    {
        tbl.Cell(zaehlerZeile + 1, zaehlerSpalte + 1).Range.Text =
            aDaten[zaehlerZeile, zaehlerSpalte];
    }
}

rng = null;
rng = BereichNachTabelleFestlegen(tbl);

```

**Listing 6.61** (.NET): Die C#-Version (*Fortsetzung*)

```

tbl = null;
//Aus den Daten eine zeichengetrennte Zeichenkette erstellen
//Diese einfügen und in eine Tabelle umwandeln.
string daten = "";
for (zaehlerZeile = aDaten.GetLowerBound(0);
    zaehlerZeile <= aDaten.GetUpperBound(0); zaehlerZeile++)
{
    for (zaehlerSpalte = aDaten.GetLowerBound(1);
        zaehlerSpalte <= aDaten.GetUpperBound(1); zaehlerSpalte++)
    {
        daten += aDaten[zaehlerZeile, zaehlerSpalte];
        if (zaehlerSpalte != aDaten.GetUpperBound(1))
        {
            daten += tab;
        }
    }
    if (zaehlerZeile != aDaten.GetUpperBound(1))
    {
        daten += "\n";
    }
}
rng.Text = daten;
object objTab = (object) tab;
tbl = rng.ConvertToTable(ref objTab, ref objMissing, ref objMissing, ref objMissing,
    ref objMissing, ref objMissing, ref objMissing, ref objMissing, ref objMissing,
    ref objMissing, ref objMissing, ref objMissing, ref objMissing, ref objMissing);
}

private void DatenZuweisen(ref string[,] daten)
{
    for (int zaehlerZeilen = daten.GetLowerBound(0);
        zaehlerZeilen <= daten.GetUpperBound(0); zaehlerZeilen++)
    {
        for (int zaehlerSpalten = daten.GetLowerBound(1);
            zaehlerSpalten <= daten.GetUpperBound(1); zaehlerSpalten++)
        {
            daten[zaehlerZeilen, zaehlerSpalten] = String.Format("Zeile {0}, Spalte {1}",
                (zaehlerZeilen + 1), (zaehlerSpalten + 1));
        }
    }
}

private wd.Range BereichNachTabelleFestlegen(wd.Table tbl)
{
    object objCollapseEnd = wd.WdCollapseDirection.wdCollapseEnd;
    wd.Range rng = tbl.Range;
    rng.Collapse(ref objCollapseEnd);
    rng.InsertParagraphAfter();
    rng.Collapse(ref objCollapseEnd);
    return rng;
}

```

## PROFITIPP

Eine alternative, noch schnellere Methode, um lange Tabellen zu erstellen, wäre, sie als RTF-, HTML- oder (nur in Word 2003) XML-Datei zu erstellen und in Word zu importieren. Unter Umständen wäre es sogar möglich, das ganze Dokument so zu erstellen, um es dann in Word zu öffnen und als Word-Dokument zu speichern. Mehr zum XML-Dateiformat finden Sie in Teil VI dieses Buches. Weitere Informationen zu allen Dateiformaten befinden sich auf der MSDN-Webseite bei Microsoft.com.



Die Beispieldatei *Bsp06\_01\_Table.doc* finden Sie auf der CD-ROM im Ordner *\Beispiele\Kap06*.

## Tabellen formatieren

Die Tabellenformatierung umfasst zwei Aspekte: das Aussehen und das grafische Layout. Unter Aussehen verstehen wir sowohl Schrift- und Absatzzeigenschaften sowie den Textfluss innerhalb von Tabellenzellen als auch Rahmen und Schattierungen. Diese werden mit den gleichen Befehlen ausgeführt, wie sonst für die Formatierung von Text in der Word-Umgebung und dürfen in einer Tabellenformatvorlage festgehalten und damit einer Tabelle zugewiesen werden.

Als zum grafischen Layout gehörend betrachten wir alles, was mit der Größe der Tabelle und ihrer Positionierung auf der Seite zu tun hat. Hierzu gehören Spaltenbreite, Zeilenhöhe, Spalten- und Zeilenanzahl, Zellenverbindungen und die Textflussformatierung um die Tabelle. Diese Eigenschaften können nicht als Teil einer Tabellenformatvorlage definiert werden.

### Die nicht-grafische Formatierung

Die Formatierung von Text innerhalb einer Tabelle wird genau gleich vorgenommen wie außerhalb der Tabelle. Die Formatierung wird einer Markierung (Selection) oder einem Bereich (Range) zugewiesen. Hier einige Beispiele:

Schrift-  
formatie-  
rung

Um den gesamten Text einer Tabelle mit der Schrift »Verdana« zu formatieren:

```
tbl.Range.Font.Name = "Verdana"
```

Um das zweite Wort der ersten Zelle einer Tabelle rot zu formatieren:

```
tbl.Cell(1,1).Range.Words(2).Font.Color = wdColorRed
```

Nur vereinzelte Codebeispiele für C# werden aufgeführt, um zu veranschaulichen, wie mit Zeilen und Zellen umzugehen ist. In C# sieht die obige Codezeile so aus (beachten Sie, dass die Words-Auflistung als ein Array behandelt wird):

```
tbl.Cell(1,1).Range.Words[2].Font.Color = wd.WdColor.wdColorRed;
```

Absatz-  
formatie-  
rung

Um alle Absätze der letzten Zelle einer Tabelle mit einem Erstzeilen-Einzug von 0,3 cm zu versehen:

```
row.Cells(row.Cells.Count).Range.ParagraphFormat.FirstLineIndent = _
CentimetersToPoints(0.3)
```

C# behandelt auch die Rows-Auflistung wie ein Array:

```
row.Cells[row.Cells.Count].Range.ParagraphFormat.FirstLineIndent = _
    wdApp.CentimetersToPoints(0.3f);
```

Rahmen-  
formatie-  
rung

Um einen Rahmen mit den standardmäßigen Einstellungen um den ersten Absatz einer Zelle zu zeichnen:

```
cel.Range.Paragraphs(1).Borders.Enable = True
```

In C# erwartet die Borders.Enable-Eigenschaft eine Ganzzahl und nicht einen booleschen Wert:

```
cel.Range.Paragraphs[1].Borders.Enable = -1;
```

Und so wird ein Rahmen um die Zelle selber gezeichnet, der hier definierte Rahmen ist blau gepunktet und 300 Pt breit:

```
With cel.Borders
    .Enable = True
    .OutsideColor = wdColorBlue
    .OutsideLineStyle = wdLineStyleDot
    .OutsideLineWidth = wdLineWidth300pt
End With
```

Schattie-  
rung

Um das letzte Wort der letzten Zelle der ersten Spalte mit einer gelben Schattierung zu hinterlegen:

```
Set cel = tbl.Columns(1).Cells(tbl.Rows.Count)
'Das letzte Wort ist das "Ende-der-Zelle"-Zeichen, daher können wir .Last nicht brauchen.
'Das zweitletzte ist die verborgene Absatzmarke, also müssen wir zwei Wörter "zurück".
Set rng = cel.Range.Words(cel.Range.Words.Count - 2)
rng.Shading.ForegroundPatternColor = wdColorYellow
```

Auch hier keine Überraschungen in der C#-Version. Die Columns-Auflistung wird wie ein Array behandelt.

```
cel = tbl.Columns[1].Cells[tbl.Rows.Count];
wd.Range rng = cel.Range.Words[cel.Range.Words.Count - 2];
rng.Shading.ForegroundPatternColor = wd.WdColor.wdColorYellow;
```

Und um der Zelle eine Schattierung zuzuweisen (die Zeichenschattierung bleibt sichtbar):

```
cel.Shading.ForegroundPatternColor = wdColorBrightGreen
```

Format-  
vorlage

Um die erste Zeile mit der Formatvorlage »Tabellenkopf« zu formatieren:

```
tbl.Rows(1).Range.Style = "Tabellenkopf"
```

Und schließlich, um die erste Spalte mit einer Formatvorlage zu formatieren:

```
tbl.Columns(1).Select  
Selection.Style = "ErsteSpalte"
```

**ACHTUNG** Bitte beachten Sie, dass es nicht möglich ist, eine Spalte einem Range zuzuweisen. Soll sie als Einheit formatiert werden, muss sie zuerst markiert werden.

Tabellen-  
formatie-  
rung

Zudem gibt es einige Tabelleneigenschaften, die den Textfluss in der Tabelle beeinflussen. Viele davon sind eher unbekannt, weshalb in diesem Abschnitt neben dem Codebeispiel auch der Menübefehl in der Benutzerschnittstelle erwähnt wird. Sie befinden sich mit einer Ausnahme alle im Dialogfeld *Tabelle/Tabelleneigenschaften*. Die Abbildung 6.30 veranschaulicht die Wirkung der vorgestellten Anweisungen.

Links sehen Sie die Tabelle, bevor die Textflussformatierung geändert wurde; rechts die Folgen dieser Formatierungen:

- Die ersten zwei Zeilen werden auf den folgenden Seiten wiederholt.
- Der Text innerhalb einer Zelle darf nicht über die Seite umbrechen.
- Der erste Absatz der vierten Zelle in Spalte 1 darf nicht in die nächste Zeile umbrechen.
- Der Tabellenrand (statt dem Text in der Tabelle) steht linksbündig mit dem Dokumenttext. Dies fällt weniger auf, weil der Abstand zwischen Text und Zellrand auf Null gesetzt wurde.

Abbildg. 6.30

Auswirkung der Befehle, die den Textumbruch und seine Position in der Tabellenzelle festlegen

Spalte 1	Spalte 2	Spalte 1	Spalte 2
Zeile 2		Zeile 2	
	Franz jagt im komplett- verwahrlosten Taxi quer- durch Bayern. Franz jagt im		Franz jagt im komplett- verwahrlosten Taxi quer durch Bayern. Franz jagt im komplett- verwahrlosten
	komplett- verwahrlosten		
Mehr Text als Platz in der Zelle ist. Noch eine Zeile, die umbricht.		Mehr Text als Platz in der Zelle ist. Noch eine Zeile, die umb	

Tabellen-  
einzug

Eine Tabelle wird üblicherweise so erstellt, dass der darin enthaltene Text links mit dem übrigen Text im Dokument bündig steht. Das bedeutet, der Tabellenrahmen ragt in den linken Seitenrand hinein, was aber nicht gepflegt aussieht, wenn die Tabelle mit Rahmen formatiert ist. Auf der Registerkarte *Tabelle* kann die Position der Tabelle, relativ zum linken Rand, mit dem Feld *Einzug von links* festgelegt werden. Im Word-Objektmodell sieht die Anweisung so aus:

```
tbl.Rows.LeftIndent = CentimetersToPoints(0)
```

#### WICHTIG

Rahmen sind nicht mit Gitternetzlinien zu verwechseln; Rahmen werden ausgedruckt, Gitternetzlinien sind nur auf dem Bildschirm sichtbar und werden über den Menübefehl *Tabelle/Gitternetzlinien ausblenden* (bzw. *anzeigen*) gesteuert. Das Gegenstück im Word-Objektmodell ist

```
ActiveWindow.View.TableGridlines = False '( bzw. True)
```

Zellen-  
wechsel

Standardmäßig wird eine Tabellenzelle umbrochen, wenn das Ende der Seite erreicht wird; ein Teil des Textes ist auf einer Seite, der Rest auf der nächsten. Soll der gesamte Zelleninhalt zwingend auf einer Seite bleiben (solange er nicht länger als eine Seite ist), wird das Kontrollkästchen *Zeilenwechsel auf Seiten zulassen* der Registerkarte *Zeile* deaktiviert. Der Beispielcode:

```
tbl.Rows.AllowBreakAcrossPages = False
```

Auch diese Eigenschaft verlangt in C# eine Ganzzahl statt eines booleschen Werts.

```
tbl.Rows.AllowBreakAcrossPages = 0;
```

Kopfzei-  
len wie-  
derholen

Bei langen Tabellen wird die Übersicht besser bewahrt, wenn die Kopfzeilen (Spaltenüberschriften) auf jeder Seite wiederholt werden. In der Benutzerschnittstelle müssen die Zeilen am Tabellenanfang markiert und dann das Kontrollkästchen *Gleiche Kopfzeile auf jeder Seite wiederholen* auf der Registerkarte *Zeile* aktiviert werden (oder Menübefehl *Tabelle/Überschriftenzeilen wiederholen*). Bei der Verwendung von Bereichen in der Automatisierung muss der Bereich entsprechend festgelegt werden:

```
Set rng = tbl.rows(1).Range
rng.MoveEnd Unit:=wdRow, Count:=1
'Die erste Zeile wird anfangs jeder Seite automatisch wiederholt.
Set rows = rng.rows
rows.HeadingFormat = True
```

Und noch eine Eigenschaft, die in C# statt eines booleschen Wertes eine Ganzzahl verlangt:

```
rows.HeadingFormat = -1;
```

**HINWEIS**

Diese Option wird ausgesetzt, wenn ein manueller Wechsel den Tabellenfluss unterbricht.

Word bietet keine Möglichkeit, einen Zusatztext für die wiederholten Spaltenüberschriften, wie etwa »Fortsetzung«, festzulegen. Auch Feldfunktionsergebnisse erscheinen nur statisch. Wenn Sie diese Funktionalität brauchen, müssen Sie *HeadingFormat* ausschalten und die Zeilen zu Beginn jeder Seite mit der *Add*-Methode einfügen.

Senk-  
rechte  
Ausrich-  
tung

Der Text in Tabellenzellen kann sowohl senkrecht als auch waagrecht ausgerichtet werden. Der Benutzer findet diese Option in der Symbolleiste *Tabellen und Rahmen* sowie als Menübefehl *Format/Absatzrichtung*. Die entsprechende Eigenschaft im Objektmodell ist *VerticalAlignment*:

```
For Each row In rows
    row.Cells.VerticalAlignment = wdCellAlignVerticalCenter
Next
```

**ACHTUNG**

Wird ein grafisches Objekt mit Textflussformatierung in einer Tabellenzelle verankert, wird die senkrechte Ausrichtung ausgesetzt. Der Text wird dann am oberen Zellenrand stehen.

Zellenbe-  
grenzung

Meistens besteht ein Abstand zwischen dem Text und den Zellenbegrenzungen. Für die gesamte Tabelle wird dieser im Dialogfeld *Tabellenoptionen* festgelegt, das über die gleichnamige Schaltfläche in der Registerkarte *Tabelle* zu finden ist. Dieser Abstand heißt im Objektmodell *LeftPadding* (Abstand links), *RightPadding* (Abstand rechts), *TopPadding* (Abstand oben) sowie *BottomPadding* (Abstand unten), erwartet wird ein Wert in Points.

```
tbl.LeftPadding = CentimetersToPoints(0)
tbl.RightPadding = CentimetersToPoints(0)
```

Es ist auch möglich, auf der Registerkarte *Zelle* abweichende Abstände für einzelne Zellen festzulegen. Im Automatisierungscode werden statt der Tabelle die Zellen direkt angesprochen:

```
Set cel = tbl.Columns(2).Cells(2)
cel.LeftPadding = CentimetersToPoints(1)
cel.RightPadding = CentimetersToPoints(0.5)
```

Zellen-  
umbruch  
Text  
anpassen

Letztlich kann der Textumbruch am Zellenrand über die Kontrollkästchen *Zeilenumbruch* und *Text anpassen* im Dialogfeld *Zellenoptionen* (aufrufbar über die Schaltfläche *Optionen* auf der Registerkarte *Zelle*) angepasst werden. Ersteres ist standardmäßig eingeschaltet und ermöglicht den Textumbruch in die nächste Textzeile, wenn die Zelle nicht breit genug ist, um ihn auf einer Textzeile anzuzeigen. Das zweite Kontrollkästchen verringert die Breite der Zeichen, so dass der erste Absatz der Zelle in die Zellenbreite passt. Im Objektmodell heißen die beiden *WordWrap* bzw. *FitText*.

```
Set cel = tbl.Cell(4, 1)
cel.FitText = True
cel.WordWrap = False
```

Diese zwei Eigenschaften des Cell-Objekts erwarteten in C# boolesche Werte:

```
cel = tbl.Cell(4, 1);
cel.FitText = true;
cel.WordWrap = false;
```



Tabellen-  
format-  
vorlage

Die Beispieldatei *Bsp06\_02\_Table.doc*, woraus die obigen Codeschnipsel stammen, finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap06*.

Eine Tabellenformatvorlage darf alle oben erwähnten Formatierungen beinhalten. Sie kann im Dokument bereits vorhanden sein oder im Code im Dokument erstellt werden. Eine Tabellenformatvorlage wird wie folgt einer Tabelle zugewiesen:

```
doc.Tables(1).Style = "Name der Formatvorlage"
```

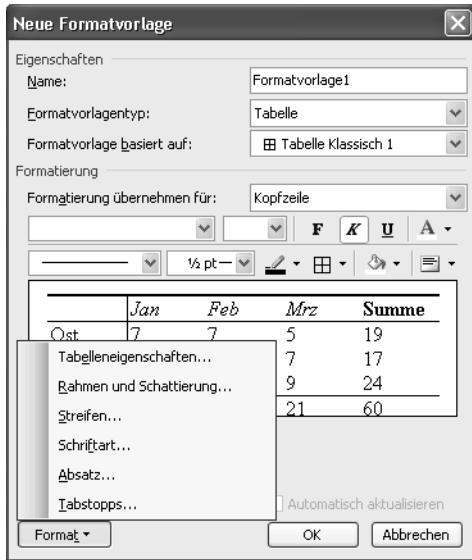
**ACHTUNG** Tabellenformatvorlagen wurden in Word 2002 eingeführt. Falls ein Dokument, worin Tabellenformatvorlagen für die Formatierung benutzt wurden, in einer früheren Word-Version geöffnet wird, gehen die Tabellenformatvorlagen verloren. Die Konvertierung behält alle Tabellenformatierungen wie Rahmenlinien und Schattierungen. Schrift- und Absatzformatierungen werden jedoch auf die Formatierung der Formatvorlage »Standard« des Zieldokuments zurückgesetzt.

In der Benutzerschnittstelle wird eine Tabellenformatvorlage wie folgt erstellt:

1. Rufen Sie den Menübefehl *Format/Formatvorlagen und Formatierung* aus.
2. Im Aufgabenbereich des gleichen Namens klicken Sie auf die Schaltfläche *Neue Formatvorlage*. Das Dialogfeld in Abbildung 6.31 wird eingeblendet.
3. Geben Sie im obersten Feld des Dialogfelds einen Namen ein.
4. Legen Sie als *Formatvorlagentyp* »Tabelle« fest.
5. Falls die Formatvorlage auf einem bestehenden Tabellen-AutoFormat oder einer anderen Tabellenformatvorlage basieren soll, wählen Sie den entsprechenden Eintrag aus der Liste *Formatvorlage basiert auf*.
6. Nun wählen Sie aus der Liste *Formatierung übernehmen für* den Teil der Tabelle, dessen Formatierung festzulegen ist. Zur Verfügung stehen »Gesamte Tabelle«, »Kopfzeile« (wdFirstRow), »Letzte Zeile« (wdLastRow), »Linke Spalte« (wdFirstColumn), »Rechte Spalte« (wdLastColumn), »Gerade Zeilen« (wdEvenRowBanding), »Ungerade Zeilen« (wdOddRowBanding), »Ungerade Spalten« (wdOddColumnBanding), »Gerade Spalten« (wdEvenColumnBanding), »Linke obere Zelle« (wdNWCell), »Rechte obere Zelle« (wdNECell), »Linke untere Zelle« (wdSWCell) sowie »Rechte untere Zelle« (wdSECell).  
(Die Ausdrücke in Klammern sind die WdConditionCode-Konstantwerte der Condition-Methode. Sie werden beim Erstellen einer Tabellenformatvorlage gebraucht.)
7. Legen Sie die gewünschte Formatierung fest. Zur Verfügung stehen alle Einträge hinter der Schaltfläche *Format* (Abbildung 6.31): *Tabelleneigenschaften, Rahmen und Schattierungen, Streifen, Schriftart, Absatz* und *Tabstopps*.



Abbildg. 6.31 Wählen Sie den Formatvorlagentyp *Tabelle*, um eine Tabellenformatvorlage zu definieren



**HINWEIS** Ein Codebeispiel für das Erstellen einer Tabellenformatvorlage finden Sie in der Lösung in Kapitel 20.

Tabellenformatvorlagen weisen einige Besonderheiten auf, die nachfolgend zusammengefasst sind.

### Schrift- und Absatzformatierungen

Tabellenformatvorlagen können nicht mit Zeichen- oder Absatzformatierungen verknüpft werden. Dieser Umstand schränkt ihren Nutzen erheblich ein, wenn die Formatierungen differenzierter sein sollen, als die Funktionalität vorsieht. Sie dürfen durchaus Text in den Tabellen mit Zeichen- und Absatzformatvorlagen zusätzlich formatieren; diese »überlagern« die Einstellungen der Tabellenformatvorlage.

Aus diesem Grund ist es auch wichtig, wenn die Einstellungen der Tabellenformatvorlage zur Geltung kommen sollen, dass bei der Erstellung der Tabelle der aktuelle Absatz mit der Formatvorlage »Standard« formatiert ist.

**ACHTUNG** Aus irgendeinem uns unbekannten Grund ist es nicht möglich, eine Tabellenformatvorlage mit Arial 10 als Schriftart und -grad zu definieren. Die Einstellung wird schlicht ignoriert. Jede andere Größe wird anerkannt und umgesetzt. Sie können entweder als Schriftgröße 9,5 oder 10, 5 eintippen oder die Schriftgröße der *Standard*-Formatvorlage auf 10 festlegen (und eine andere Formatvorlage für den Fließtext benutzen), wenn die Schriftgröße der Tabelle unbedingt 10 sein soll.

### Streifen

Streifen erhöhen die Lesbarkeit einer langen Tabelle. Wenn Sie eine Schattierung für *Ungerade Zeilen*, *Gerade Zeilen*, *Ungerade Spalten* oder *Gerade Spalten* festlegen, wird die Tabelle automatisch mit alternierenden Streifen formatiert. Wurde für die Formatvorlage eine *Kopfzeile* oder *Linke Spalte*

definiert, ist die erste ungerade Zeile oder Spalte die darauf folgende (also die zweite der gesamten Tabelle). Die Anzahl der Zeilen und/oder Spalten in einem Verbund bestimmen Sie über den Eintrag *Streifen* unter der Schaltfläche *Format*.

### Diagonale Rahmenlinien

Diagonale Rahmenlinien stehen in Tabellenformatvorlagen zur Verfügung. Ihre eigentliche Anwendung ist etwas kompliziert, weil sie dazu neigen, alle anderen Rahmeneinstellungen durcheinander zu bringen. Es erwies sich als unmöglich, die Tabellenformatvorlage in Abbildung 6.32 in der Benutzeroberfläche zu erstellen. In VBA gelingt es, aber der richtige Weg war nicht sofort zu erkennen und variierte je nach individueller Zelle und Rahmenformatierung. Meist musste der Befehl für die Diagonale und gelegentlich auch für andere Rahmenlinien wiederholt werden, bis alle Einstellungen richtig interpretiert wurden. »Probieren, probieren, probieren« heißt die Devise, bis es klappt.

Abbildg. 6.32 Diagonale Rahmenlinien sind sehr heikel zu realisieren, da sie evtl. andere Rahmenformatierungen ausschalten

	2000	2001	2002	Durchschnitt
Orangen	1980	2300	2000	1990
Bananen	1650	1800	1900	1775
Ananas	850	900	740	795
Summen	6480	7001	6642	

### Eckzellen

Falls Sie einer Eckzelle eine besondere Formatierung zuweisen, werden Sie unter Umständen feststellen, dass die Zelle sich scheinbar weigert, diese anzunehmen. Und zwar kommen Eckzellenformatierungen erst zum Vorschein, wenn der Zeile und der Spalte, die sich an diesem Punkt kreuzen, auch eine Formatierung zugeteilt wurde. Diese Einschränkung können Sie in der Benutzeroberfläche umgehen, indem Sie der Zeile und der Spalte den Schriftschnitt *Fett* zuweisen und umgehend wieder ausschalten. Somit haben diese Elemente die Formatierung »nicht Fett« und die Eckzelle erscheint mit ihrer Formatierung.

### Tabelleneigenschaften

Obwohl unter der *Format*-Schaltfläche der Eintrag *Tabelleneigenschaften* erscheint, stehen viele der darin enthaltenen Attribute den Tabellenformatvorlagen nicht zur Verfügung. Es ist unmöglich, einen Textfluss für die Tabelle oder eine bevorzugte Spaltenbreite oder Zeilenhöhe zu bestimmen. Die Tabellen- und Zellenausrichtung können hingegen festgelegt werden. Auch lässt sich der Seitenumbruch innerhalb von Zellen unterbinden.

Das Kontrollkästchen *Gleiche Kopfzeile auf jeder Seite wiederholen* auf der Registerkarte *Zeile* kann ebenfalls aktiviert werden – aber aufgepasst! Wenn Sie den Eintrag *Kopfzeile* im Dropdown-Feld *Formatierung übernehmen für* nicht gewählt haben, übernehmen bei aktiviertem Kontrollkästchen alle Tabellenteile die Formatierung der Kopfzeile. Denken Sie also daran, diese Option nur für die Kopfzeile zu aktivieren oder setzen Sie diese Einstellung für jede Tabelle einzeln, über *Tabelle/Tabelleneigenschaften/Zeile*.

### Tabellenformatvorlage als Standard-Tabellenformatierung

Zusätzlich zur erhöhten Effizienz bei der Arbeit ist die wichtigste Aufgabe einer Tabellenformatvorlage, für ein einheitliches Aussehen der Tabellen in einem Dokument zu sorgen. Deshalb kann eine Tabellenformatvorlage als Standard-Tabellenformatierung für die Tabellen eines Dokuments oder

für die gesamte Word-Umgebung festgelegt werden. Klicken Sie im Aufgabenbereich *Formatvorlagen und Formatierung* rechts auf den entsprechenden Eintrag und wählen Sie *Als Standard-Tabellenformatvorlage festlegen*. Anschließend erscheint ein Dialogfeld, in dem bestimmt wird, ob diese Formatvorlage als Standard nur für dieses Dokument oder für alle neuen, auf dieser Vorlage basierenden Dokumente zu übernehmen ist.

Das standardmäßige Tabellenformat von Word ist *Tabellengitternetz*. Wenn Sie keine andere Tabellenformatvorlage als Standard festgelegt haben und im Dialogfeld *Tabelle einfügen* kein AutoFormat für neue Tabellen wählen, weist Word neuen Tabellen diese Formatvorlage zu. Es steht Ihnen frei, diese Tabellenformatvorlage zu ändern. Durch Aktivierung des Kontrollkästchens *Zur Vorlage hinzufügen* im Dialogfeld *Formatvorlage ändern* übernimmt Word Ihre Einstellungen für die ganze Word-Umgebung, wenn das Dokument auf der *Normal.dot* basiert.

## PROFITIPP

Wollen Sie mehr als die von der Tabellenformatvorlage unterstützten Formatierungen festlegen (wie etwa Spaltenbreite oder Zeilen- und Spaltenanzahl), sollten Sie in Betracht ziehen, die Tabelle als AutoText-Eintrag zu speichern. Wir machen aber darauf aufmerksam, dass beim Einfügen eines solchen AutoText-Eintrags die Rahmenlinien verändert werden könnten, wenn die ursprüngliche Tabelle mit der standardmäßigen Tabellenformatvorlage »Tabellengitternetz« formatiert wurde. Die Rahmenlinien werden der Formatvorlagendefinition im Zieldokument angepasst.

## Das grafische Layout einer Tabelle

Unter den grafischen Aspekten der Tabellenformatierung verstehen wir die Eigenschaften und Methoden, die die Positionierung und Größe einer Tabelle und ihrer Komponenten festlegen, die in einer Tabellenformatvorlage nicht festgehalten werden können.

Tabelle  
positionieren

In allen Versionen von Word kann festgelegt werden, ob eine Tabelle, ähnlich wie ein Absatz, linksbündig, zentriert oder rechtsbündig relativ zu den Texträndern zu positionieren ist. In der Benutzeroberfläche befindet sich diese Einstellung im Dialogfeld *Tabelleneigenschaften* auf der Registerkarte *Tabelle*. Erstreckt sich eine Tabelle über die gesamte Seitenbreite, zeigt die Einstellung logischerweise keine Wirkung. Diese Option ermöglicht keinen Textfluss um die Tabelle. Im Objektmodell stellt die *Alignment*-Eigenschaft der *Rows*-Auflistung diese Funktionalität dar:

```
ActiveDocument.Tables(1).Rows.Alignment = wdAlignRowCenter
```



Seit Word 2000 können Tabellen frei auf der Seite, mit Textfluss, positioniert werden. In der Benutzeroberfläche erfolgt dies durch Ziehen am »Ziehpunkt«, der am oberen linken Tabellenrand erscheint, wenn sich die Einfügemarke in einer Tabelle befindet, oder über die Registerkarte *Tabelle* im Dialogfeld *Tabelleneigenschaften*. Die genaue Stelle lässt sich mit dem Dialogfeld *Tabellenposition* festlegen, die über die Schaltfläche *Optionen* zu erreichen ist. (Diese Einstellungen sind jenen für Grafiken sehr ähnlich. Mehr Informationen finden Sie im Abschnitt »Grafiken: Die *InlineShape*- und *Shape*-Objekte« in diesem Kapitel.)

Abbildg. 6.33 Eine Tabelle mit Textflussformatierung genau positionieren



Im Objektmodell stellt die Eigenschaft `WrapAroundText` diese Funktionalität dar. Wie `Alignment` ist auch sie eine Eigenschaft der `Rows`-Auflistung:

```
ActiveDocument.Tables(1).Rows.WrapAroundText = True
```

Die Position wird, ähnlich wie bei grafischen Objekten, mit den Eigenschaften `HorizontalPosition` und `VerticalPosition`, im Zusammenspiel mit `RelativeHorizontalPosition` und `RelativeVerticalPosition` festgelegt. Die beiden letzten bestimmen, ob die Position relativ zu Zeichen, Zeile, Absatz, Spalte, Rand oder Seite sein soll, und sind in der Tabelle 6.17 sowie in der Tabelle 6.18 näher beschrieben. `HorizontalPosition` und `VerticalPosition` legen den Abstand in Bezug auf diesen Punkt fest und können eine Zahl des Datentyps `Single` oder ein `WdTablePosition`-Konstantwert sein.

Tabelle 6.13 `WdTablePosition`-Konstantwerte

<code>WdTablePosition</code> -Enum	Wert	Beschreibung
<code>wdTableBottom</code>	-999997	Die Tabelle wird bündig zum unteren Text- oder Seitenrand positioniert.
<code>wdTableCenter</code>	-999995	Die Tabelle wird senkrecht oder waagrecht zwischen Text- oder Seitenrändern positioniert.
<code>wdTableInside</code>	-999994	Die Tabelle wird am inneren Text- oder Seitenrand positioniert, wenn <i>Gerade/ungerade anders in Datei/Seite einrichten/Layout</i> aktiviert ist.
<code>wdTableLeft</code>	-999998	Die Tabelle wird bündig zum linken Text- oder Seitenrand positioniert.
<code>wdTableOutside</code>	-999993	Die Tabelle wird am äußeren Text- oder Seitenrand positioniert, wenn <i>Gerade/ungerade anders in Datei/Seite einrichten/Layout</i> aktiviert ist.
<code>wdTableRight</code>	-999996	Die Tabelle wird bündig zum rechten Text- oder Seitenrand positioniert.

Tabelle 6.13 WdTablePosition-Konstantwerte (Fortsetzung)

WdTablePosition-Enum	Wert	Beschreibung
wdTableTop	-999999	Die Tabelle wird bündig zum oberen Text- oder Seitenrand positioniert.

Um beispielsweise eine Tabelle zentriert zwischen den Seitenrändern und 0,7 Zentimeter unter dem oberen Rand des verankernden Absatzes zu positionieren:

```
Set tbl = ActiveDocument.Tables(1)
tbl.Rows.WrapAroundText = True
tbl.Rows.RelativeHorizontalPosition = wdRelativeHorizontalPositionPage
tbl.Rows.HorizontalPosition = wdTableCenter
tbl.Rows.RelativeVerticalPosition = wdRelativeVerticalPositionParagraph
tbl.Rows.VerticalPosition = CentimetersToPoints(0.7)
```

In Word 2000 können solche Tabellen nicht auf die nächste Seite umbrechen, sondern sind auf eine einzige Seite beschränkt. In späteren Versionen ist dieses Verhalten nicht nur erlaubt, sondern auch standardmäßig aktiviert und kann in *Extras/Optionen/Kompatibilität* unterbunden werden. Im Objektmodell von Word 2002 und 2003 sorgt die folgende Anweisung dafür, dass eine Tabelle mit Textflussformatierung auf eine Seite beschränkt ist.

```
ActiveDocument.Compatibility(wdDontBreakWrappedTables) = True
```

**HINWEIS**

Mehr zu den Optionen in *Extras/Optionen/Kompatibilität* finden Sie bei <http://support.microsoft.com/default.aspx?scid=kb;de;288792>. Dieser Artikel wurde für Word 2002 geschrieben. Für Word 2003 gibt es nur eine englische Version bei <http://support.microsoft.com/default.aspx?scid=kb;en-us;288792>.

Spalten-  
breite

Wie schon erwähnt, sind die Spaltenbreiten einer Word-Tabelle seit Word 2000 nicht fix, sondern passen sich dem Inhalt an. Eine Aufforderung durch das Objektmodell gibt jedoch die momentane Spaltenbreite in Points zurück:

```
sngSpaltenBreite = ActiveDocument.Tables(2).Columns(1).Width
```

Um die Breite der ganzen Tabelle zu erhalten, summieren Sie die Breite aller Spalten:

```
Set tbl = ActiveDocument.Tables(2)
For Each col In tbl.Columns
    sngBreite = sngBreite + col.Width
Next
Debug.Print PointsToCentimeters(sngBreite)
```

(Es gibt zwar eine Eigenschaft `Table.PreferredWidth`. Diese gibt jedoch den Wert 9999999 zurück, wenn die Spalten sich dem Inhalt anpassen dürfen.)

Die Spaltenbreite wird auch mit der Eigenschaft `Width` festgelegt. Aber wenn die Spalten sich dem Inhalt anpassen dürfen, hat diese nur Empfehlungswert. Um eine feste Breite zuzuweisen, muss zuerst die `AllowAutoFit`-Eigenschaft für die Tabelle auf `False` gesetzt werden:

```
Set tbl = ActiveDocument.Tables(2)
tbl.AllowAutoFit = False
For Each col In tbl.Columns
    col.Width = 75
Next
```

Zeilen-  
höhe

Auch die Zeilenhöhen von Word-Tabellen sind standardmäßig variabel und passen sich dem Inhalt an. Dies gilt für alle Word-Versionen. Im Gegensatz zu den Spalten gibt eine Nachfrage der `Row.Height`-Eigenschaft nicht die aktuelle Zeilenhöhe zurück, sondern die Einstellung in *Tabelle/Tabelleneigenschaften/Zeile*. Ist *Höhe definieren* nicht aktiviert, wird der Wert 9999999 zurückgegeben. Ist *Zeilenhöhe* auf *Mindestens* gesetzt, entspricht der Rückgabewert der Mindesthöhe. Nur, wenn *Zeilenhöhe* auf *Genau* festgelegt ist, liefert die Eigenschaft die aktuelle Zeilenhöhe.

Im Objektmodell wird das Verhalten über die `HeightRule`-Eigenschaft geregelt. Es gibt drei Einstellungen, die jenen des Dialogfelds entsprechen: `wdRowHeightAtLeast` (Mindestzeilenhöhe), `wdRowHeightExactly` (genaue Zeilenhöhe) sowie `wdRowHeightAuto` (automatische Zeilenhöhe). Ihre Verwendung wird in Listing 6.62 veranschaulicht. Bitte beachten Sie, dass das Festlegen einer Zeilenhöhe für eine Zeile mit automatischer Zeilenhöhe die `HeightRule` auf `wdRowHeightAtLeast` festlegt.

**Listing 6.62** Die Auswirkungen der *Row*-Eigenschaften *Height* und *HeightRule*

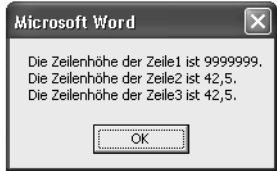
```
Sub TabellenZeilenHöhen()
    Dim strZeilenhöhen As String
    Dim tbl As Word.Table
    Dim row As Word.Row

    Set tbl = ActiveDocument.Tables(2)
    Set row = tbl.Rows(1)
    row.HeightRule = wdRowHeightAuto
    'row.Height = CentimetersToPoints(1.5) würde
    'row.HeightRule = wdRowHeightAtLeast bedeuten.
    Set row = tbl.Rows(2)
    'Die folgende Zeile ist eigentlich überflüssig, da die nächste
    'diese Eigenschaft automatisch festlegt
    row.HeightRule = wdRowHeightAtLeast
    row.Height = CentimetersToPoints(1.5)
    Set row = tbl.Rows(3)
    row.HeightRule = wdRowHeightExactly
    row.Height = CentimetersToPoints(1.5)
    For Each row In tbl.Rows
        strZeilenhöhen = strZeilenhöhen & "Die Zeilenhöhe der Zeile" & _
            CStr(row.Index) & " ist " & CStr(row.Height) & "." & vbCrLf
    Next
    MsgBox strZeilenhöhen
End Sub
```

Das Resultat sehen Sie in Abbildung 6.34. Obwohl die Zeilen 2 und 3 angeblich die gleiche Höhe haben, ist dies offensichtlich nicht der Fall. Für Zeile 2 gilt eine Mindesthöhe, für Zeile 3 eine genaue. Ferner zu beachten ist, dass »überlaufender« Inhalt einer Zeile mit genauer Höhe unten »verschwindet«. Er ist noch vorhanden, jedoch nicht sichtbar.

Abbildg. 6.34 Rückgabewerte der drei Zeilenhöhe-Typen

Zeile mit automatischer Höhenanpassung		
Zeile mit mindest Zeilenhöhe. Sie wird aber wachsen, wenn sich mehr Text darin befinden, wie hier der Fall ist		
Zeile mit fester Zeilenhöhe. Auch wenn mehr Text in der Zelle ist,		



Unter normalen Umständen lässt sich also die Höhe einer Tabelle nicht ermitteln, außer sie besteht ausschließlich aus Zeilen mit genauer Höhe. Die einzige Möglichkeit ist, sich der Information-Eigenschaft zu bedienen, in Verbindung mit dem Argument `wdVerticalPositionRelativeToPage`. Das Listing 6.63 zeigt, wie die Höhe einer Tabelle annähernd berechnet werden könnte.

Um die Höhe einer Tabelle annähernd zu berechnen, werden die Positionen relativ zur Seite der ersten Zeile der ersten Zelle sowie der letzten Zeile der letzten Zelle ermittelt. Hinzu kommt die Schriftgröße der letzten Zelle, die Abstände zwischen Zellenrändern und dem Text, sowie die Breite der Zellenrahmen. Dieser Vorschlag setzt voraus, dass die letzte Zeile nicht mit einer genauen Höhe formatiert ist.

**Listing 6.63** Die gesamte Höhe einer Tabelle kann nur annähernd ermittelt werden, indem viele Faktoren geprüft werden

```

Sub TabellenHöheBerechnen()
    Dim tbl As Word.Table
    Dim celStart As Word.Cell
    Dim celEnde As Word.Cell
    Dim rngStart As Word.Range
    Dim rngEnde As Word.Range
    Dim sngHöhe As Single

    Set tbl = ActiveDocument.Tables(1)
    Set celStart = tbl.Cell(1, 1)
    Set rngStart = celStart.Range
    Set celEnde = tbl.Cell(tbl.Rows.Count, 1)
    Set rngEnde = celEnde.Range
    'Bereich in der letzten Textzeile der Zelle setzen
    rngEnde.Collapse Direction:=wdCollapseEnd
    rngEnde.MoveEnd Unit:=wdWord, Count:=-1
    sngHöhe = rngEnde.Information(wdVerticalPositionRelativeToPage) _
        - rngStart.Information(wdVerticalPositionRelativeToPage) _
        + (rngEnde.Font.Size + celStart.TopPadding + celEnde.BottomPadding) _
        + celStart.Borders.OutsideLineWidth + celStart.Borders.OutsideLineWidth _
        + celEnde.Borders.OutsideLineWidth + celEnde.Borders.OutsideLineWidth
    Debug.Print PointsToCentimeters(sngHöhe)
End Sub

```

Zellen  
verbinden

In Word-Tabellen können nebeneinander liegende Zellen waagrecht sowie senkrecht verbunden werden. In der Benutzerschnittstelle werden die Zellen markiert und dann der Menübefehl *Tabelle/Zellen verbinden* gewählt. Im Objektmodell entspricht diesem Befehl die Methode `Merge`, die mit dem `Selection`- oder `Range`-Objekt benutzt wird:

```
Set tbl = ActiveDocument.Tables(1)
Set rng = tbl.Cell(2, 1).Range
rng.MoveEnd Unit:=wdCell, Count:=1
rng.Cells.Merge
```

Zellen  
teilen

Zellen können gleichwohl auch in mehrere aufgeteilt werden. Auch der Befehl *Zellen teilen* steht im Menü *Tabellen* zur Verfügung. Nach dessen Aufruf wird ein Dialogfeld eingeblendet, in dem die resultierende Zeilen- und Spaltenanzahl festgelegt wird. Das Objektmodell bietet die Methode `Cell.Split` an. Die folgende Codezeile teilt die erste Zelle der dritten Zeile in zwei untereinander stehende Zellen.

```
tbl.Cells(3, 1).Split 2, 1
```

Diese beiden Methoden sind relativ einfach und wären kaum erwähnenswert, wenn sie nicht zu einem weitaus komplexeren Thema führen würden: dem Umgang mit Tabellen, die verbundene und/oder geteilte Zellen enthalten. Mehr darüber erfahren Sie im folgenden Abschnitt.



Die Beispieldatei *Bsp06\_03\_Table.doc* finden Sie auf der CD-ROM zum Buch im Ordner `\Beispiele\Kap06`.

## Informationen aus Tabellen holen

Nicht nur die Erstellung und Formatierung von Tabellen wird automatisiert. Daten müssen auch aus Tabellen gelesen werden. Ist die Tabelle symmetrisch – sie besteht aus der gleichen Anzahl von Zellen in allen Zeilen und Spalten, und keine Zellen sind verbunden, so stellt diese Aufgabe kein großes Problem dar.

Zuerst muss die zu bearbeitende Tabelle identifiziert werden. Es könnte sich um die Tabelle handeln, in der sich die Einfügemarke befindet (`Selection.Tables(1)`). Falls Sie herausfinden müssen, welchen Indexwert diese Tabelle hat, entweder im Dokument oder in einem bestimmten Bereich, können Sie sich der in Kapitel 5 beschriebenen Technik bedienen.

Sie können auf ähnliche Art und Weise die erste, zweite oder  $n$ -te Tabelle im Dokument, in einem Abschnitt oder nach einer bestimmten Formatvorlage oder Text (in welchem Fall diese gesucht, der Bereich bis zum Dokumentende erweitert und der Tabellenindex benutzt wird) ansprechen. Eine Tabelle kann auch mit einer Textmarke versehen und ausfindig gemacht werden: `Doc.Bookmarks("Tabelle").Range.Tables(1)`.

### ACHTUNG

Was nicht zuverlässig funktioniert, ist der Gebrauch der `ID`-Eigenschaft. Wir machen immer wieder die Erfahrung, dass jemand diese Eigenschaft in der `IntelliSense`-Liste sieht und versucht, sie zur Identifizierung einer Tabelle zu verwenden, ohne den Hilfetext dazu genau durchzulesen. Wir weisen ausdrücklich darauf hin, dass diese Eigenschaft nur für die Spei-



cherung eines Dokuments als Webseite erhalten bleibt. Hier ein Auszug aus der Hilfe: »Gibt beim Speichern des aktuellen Dokuments als Webseite die Beschriftungskategorie für das angegebene Objekt zurück oder legt sie fest.«

Zelle für  
Zelle

Die offensichtlichste Methode, Daten aus einer Tabelle zu lesen, ist, durch alle Zellen zu schleifen und deren Inhalt zu lesen. Dieser Text kann in einem Array festgehalten und direkt in eine getrennte Textdatei oder in eine Datenbank geschrieben werden. Dabei ist darauf zu achten, dass sich am Ende jeder Tabellenzeile eine verborgene Absatzmarke (Zeichencode 13) sowie ein »Ende-der-Zelle-Zeichen« (Zeichencode 7) befinden. Meistens sind diese nicht erwünscht und müssen abgetrennt werden, wofür die Funktion *TrimZellenInhalt* in Listing 6.64 sorgt.

Listing 6.64

Daten aus einer Tabelle Zelle für Zelle auslesen und in ein Array aufnehmen

```
Sub DatenAusTabelleLesen_1()
    Dim tbl As Word.Table
    Dim lAnzZeilen As Long
    Dim lAnzSpalten As Long
    Dim lZeile As Long
    Dim lSpalte As Long
    Dim aDaten() As String
    Dim strZellenInhalt As String

    Set tbl = ActiveDocument.Tables(1)
    'Haben nicht alle Zeilen und Spalten die gleiche Anzahl
    'Zellen, ist die Tabelle nicht symmetrisch; abbrechen.
    If Not tbl.Uniform Then Exit Sub

    lAnzZeilen = tbl.Rows.count - 1
    lAnzSpalten = tbl.Columns.count - 1
    ReDim Preserve aDaten(lAnzZeilen, lAnzSpalten)

    For lZeile = LBound(aDaten, 1) To UBound(aDaten, 1)
        For lSpalte = LBound(aDaten, 2) To UBound(aDaten, 2)
            strZellenInhalt = tbl.Cell(lZeile + 1, lSpalte + 1).Range.Text
            strZellenInhalt = TrimZellenInhalt(strZellenInhalt)
            aDaten(lZeile, lSpalte) = strZellenInhalt
        Next lSpalte
    Next lZeile

    Debug.Print aDaten(lAnzZeilen, lAnzSpalten)
End Sub

'Am Ende jedes Zellenbereichs steht eine Absatzmarke (Chr(13))
'sowie "Ende-der-Zelle"-Zeichen (Chr(7))
Function TrimZellenInhalt(str As String)
    str = Left(str, Len(str) - 2)
    TrimZellenInhalt = str
End Function
```



Die Beispieldatei *Bsp06\_04\_Table.doc* finden Sie auf der CD-ROM im Ordner *Beispiele\Kap06*.

Convert-  
ToText-  
Methode

Da die obige Methode das Gegenstück zur Zelle-für-Zelle-Erstellung einer Tabelle darstellt, ist sie auch ähnlich langsam. Schneller geht's, wenn die Tabelle zuerst in zeichengetrennten Text umgewandelt und bearbeitet wird, wie Listing 6.65 veranschaulicht. Hierzu wird die ConvertToText-Methode (entspricht dem Menübefehl *Tabelle/Umwandeln/Tabelle in Text*) genutzt, die eine Tabelle in einen Textbereich umwandelt. Jede Tabellenzeile wird zu einem Absatz; für die Zellenbezeichner kann ein beliebiges Zeichen festgelegt werden. In Abbildung 6.35 – das Resultat von Listing 6.65 – trennt ein Tabulatorzeichen die Zellinhalte. Ein »|«-Zeichen wurde durch den Code anstelle einer Absatzmarke innerhalb einer Tabellenzeile eingefügt, während die drei Leerzeichen ein Tabulatorzeichen ersetzt haben. Diese werden später wieder zurückverwandelt.

**Abbildg. 6.35** Das Resultat der Umwandlung einer Tabelle in Text

Zelle-eins	Zelle-zwei	Zelle-drei
Zelle-vier	Zelle-fünf	Zelle-sechs
Zelle-sieben	Zelle-acht	Zelle-neun

Zelle-eins → Zelle-zwei → Zelle-drei  
 Zelle-vier → Zelle-fünf → Zelle-sechs  
 Zelle-sieben → Zelle-acht → Zelle-neun

Um zu gewährleisten, dass die Trennung in Datenfelder und Datensätze korrekt erfolgt, sollten sicherheitshalber alle in der Tabelle befindlichen Absatzmarken und Trennzeichen durch andere, nicht vorhandene Zeichen ersetzt werden. Dies wird im Listing mit der Hilfsprozedur *TrennZeichen-Ersetzen* ausgeführt. Beim Schreiben des Texts in das Array werden die Zeichen wieder ausgetauscht, so dass der Originaltext aus den Tabellenzellen gespeichert wird.

Am Schluss der Hauptprozedur werden die drei im Dokument ausgeführten Handlungen rückgängig gemacht, so dass die Tabelle wieder wie am Anfang vorliegt. Alternativ könnte sie für die Umwandlung in ein neues, temporäres Dokument übernommen werden, um sicherzugehen, dass das ursprüngliche nicht beschädigt wird.

**Listing 6.65** Eine Tabelle in eine zeichengetrennte Zeichenkette umwandeln und diese in ein Array schreiben

```
Sub DatenAusTabellenLesen_2()
    Dim tbl As Word.Table
    Dim rng As Word.Range
    Dim strTab As String
    Dim strPara As String
    Dim strTabErsatz As String
    Dim strParaErsatz As String
    Dim aTrennZeichen(1, 1) As String
    Dim aDatenTemp() As String
    Dim aDatenFelder() As String
    Dim aDaten() As String
    Dim strDaten As String
    Dim lZaehlerDS As Long
    Dim lZaehlerFeld As Long
    Dim strDatenFeld As String

    strTab = vbTab
```

**Listing 6.65** Eine Tabelle in eine zeichengetrennte Zeichenkette umwandeln und diese in ein Array schreiben (Fortsetzung)

```

strPara = vbCrLf
strTabErsatz = "   "
strParaErsatz = "|"

Set tbl = ActiveDocument.Tables(1)
'Die Trennzeichen für die umgewandelte Tabelle mit anderen
'Zeichen ersetzen.
aTrennZeichen(0, 0) = strTab
aTrennZeichen(0, 1) = strTabErsatz
aTrennZeichen(1, 0) = strPara
aTrennZeichen(1, 1) = strParaErsatz
TrennZeichenErsetzen tbl.Range, aTrennZeichen()
'Die Tabelle in zeichengetrennten Text umwandeln.
Set rng = tbl.ConvertToText(Separator:=vbTab, NestedTables:=False)
strDaten = rng.Text
'Den Text auseinander nehmen und in ein Array schreiben.
'Zuerst wird jeder Absatz (=Datensatz) in ein Array-Element geschrieben.
aDatenTemp() = Split(strDaten, strPara)
'Die Anzahl der Datensätze ist bekannt. Nun die Anzahl der Felder ermitteln.
aDatenFelder() = Split(aDatenTemp(0), strTab)
'Damit wird das Array für die Felder initialisiert.
ReDim Preserve aDaten(UBound(aDatenTemp), UBound(aDatenFelder()))
'Nun die Datenfelder aus jedem Datensatz trennen.
For lZaehlerDS = LBound(aDatenTemp()) To UBound(aDatenTemp())
    aDatenFelder() = Split(aDatenTemp(lZaehlerDS), vbTab)
    'Diese werden in das Daten-Array geschrieben.
    For lZaehlerFeld = LBound(aDatenFelder()) To UBound(aDatenFelder())
        'Zuerst die ersetzten Trennzeichen wieder herstellen.
        strDatenFeld = aDatenFelder(lZaehlerFeld)
        strDatenFeld = Replace(strDatenFeld, strTabErsatz, strTab)
        strDatenFeld = Replace(strDatenFeld, strParaErsatz, strPara)
        aDaten(lZaehlerDS, lZaehlerFeld) = strDatenFeld
    Next lZaehlerFeld
Next lZaehlerDS
'Die Tabelle wieder herstellen.
ActiveDocument.Undo Times:=3
Debug.Print aDaten(1, 0)
End Sub

Sub TrennZeichenErsetzen(ByRef rng As Word.Range, ByRef aTrennZeichen() As String)
    Dim lZaehler As Long

    With rng.Find
        .ClearFormatting
        .MatchAllWordForms = False
        .MatchCase = False
        .MatchSoundsLike = False
        .MatchWholeWord = False
        .MatchWildcards = False
        .Format = False
        .Forward = True
        .Wrap = wdFindStop
    End With
    For lZaehler = LBound(aTrennZeichen(), 1) To UBound(aTrennZeichen(), 2)

```

**Listing 6.65** Eine Tabelle in eine zeichengetrennte Zeichenkette umwandeln und diese in ein Array schreiben (*Fortsetzung*)

```
rng.Find.Execute FindText:=aTrennZeichen(1Zaehler, 0),
replacewith:=aTrennZeichen(1Zaehler, 1), Replace:=wdReplaceAll
Next 1Zaehler
rng.Find.Execute
End Sub
```



Die Beispieldatei *Bsp06\_04\_Table.doc* finden Sie auf der CD-ROM im Ordner *Beispiele\Kap06*.

Aus Platzgründen zeigen wir für C# hier nur die Handhabung der ConvertToText-Methode. Bitte beachten Sie, wie das Escape-Zeichen "\t" für ein Tabulatorzeichen zuerst einer Variablen des Typs String zugewiesen werden muss und diese dann in ein Object für das Methodenargument konvertiert wird.

```
string tab = "\t";
wd.Table tbl = wdApp.ActiveDocument.Tables[1];
//Die Tabelle in zeichengetrennten Text umwandeln.
object objSeparator = tab;
object objMissing = System.Reflection.Missing.Value;
wd.Range rng = tbl.ConvertToText(ref objSeparator, ref objMissing);
string daten = rng.Text;
```

#### TIPP

Unter Word 2003 bietet sich die Alternative, das XML-Format der Tabelle zu bearbeiten. Es ist auch möglich, das Dokument im HTML- oder RTF-Format zu speichern und dieses Ergebnis direkt zu bearbeiten.

Verschachtelte Tabellen

Seit Word 2000 werden verschachtelte Tabellen unterstützt. Diese werden grundsätzlich wie jede andere Tabelle erstellt und formatiert, wie das Listing 6.66 veranschaulicht. Dabei gibt es einen schwerwiegenden Aussetzer: Beim Einfügen der Tabelle wird das Argument NumRows ignoriert. Noch schlimmer: Word kann sich nicht merken, ob Zellen in der Tabelle auch tatsächlich vorhanden sind.

Das Listing 6.66 enthält die Funktion *TabelleEinfuegen*, die für die Erstellung der ersten verschachtelten Tabelle eingesetzt wird. Beachten Sie, wie diese die Anzahl der Zeilen kontrolliert und eine Schleife durchführt, um die neue Tabelle mit den fehlenden Zeilen zu ergänzen.

Innerhalb dieser Tabelle wird eine weitere eingefügt, dieses Mal »ganz normal«. Das Ergebnis ist in Abbildung 6.36 zu sehen. Statt drei hat die neue Tabelle nur eine Zeile. Zudem wurde der Text nicht in die zweite Zelle der zweiten Zeile eingefügt, sondern in die zweite Zelle der ersten Zeile, und dies ohne eine Fehlermeldung oder sonstigen Hinweis, dass die Zelle nicht vorhanden wäre. Äußerste Vorsicht ist also im Umgang mit verschachtelten Tabellen geboten!

**Listing 6.66** Verschachtelte Tabellen erstellen

```
Sub VerschachtelteTabellen()
Dim rng As Word.Range
Dim tblInnere As Word.Table
Dim tblInnere2 As Word.Table
Dim row As Word.row
```

Listing 6.66 Verschachtelte Tabellen erstellen (Fortsetzung)

```

'Verschachtelte Tabelle in der 2. Zelle der 2. Zeile einfügen.
Set rng = ActiveDocument.Tables(1).Cell(2, 2).Range

Set tblInnere = TabelleEinfuegen(rng, 3, 3)
Set row = tblInnere.Rows(1)
row.Range.Font.Bold = True
row.Range.Font.Size = 9
row.Cells(1).Range.Text = "Spalten Überschrift"
Set rng = tblInnere.Rows(2).Cells(2).Range
Set tblInnere2 = rng.Tables.Add(rng, 3, 3)
'Auch keine Fehlermeldung, wenn eine Zeile angesprochen wird,
'die nicht vorhanden ist. Der Text wird einfach in die nächst höhere Zelle geschrieben.
tblInnere2.Cell(2, 2).Range.Text = "Hallo"
Debug.Print "Anzahl Zeilen: " & tblInnere2.Rows.Count
End Sub

Function TabelleEinfuegen(rng As Word.Range, lAnzZeilen As Long, _
    lAnzSpalten As Long) As Word.Table
    Dim tbl As Word.Table
    Dim lZaehler As Long

    Set tbl = rng.Tables.Add(rng, lAnzZeilen, lAnzSpalten)
    'Bei verschachtelten Tabellen wird nur eine Zeile erstellt!
    For lZaehler = tbl.Rows.Count To lAnzZeilen - 1
        tbl.Rows.Add
    Next

    Set TabelleEinfuegen = tbl
End Function

```

Abbildg. 6.36 Das Ergebnis von Listing 6.66

	<b>Spalten Überschrift</b>			
		Hallo		



Die Beispieldatei *Bsp06\_05\_Table.doc* finden Sie auf der CD-ROM im Ordner *Beispiele\Kap06*.

Das Lesen von Daten aus verschachtelten Tabellen geht relativ problemlos und unterscheidet sich im Prinzip nicht von der Arbeit mit einer »gewöhnlichen« Tabelle. Beim Schleifen durch die Zellen wird die Anzahl der Tabellen in der Zelle nachgeprüft. Ist sie größer als eins, enthält die Zelle verschachtelte Tabellen, und auch diese können durchgeschleift werden, wie in Listing 6.67. Selbst eine verschachtelte Tabelle könnte weitere Tabellen enthalten, weshalb die Prozedur *ZellenNachVerschachtelteTabellenDurchsuchen* rekursiv gestaltet ist (sie ruft sich nach Bedarf selbst auf). Das Ergebnis für die Tabelle in Abbildung 6.36 ist:

Liste der verschachtelten Tabellen  
Haupttabelle hat eine verschachtelte Tabelle in Reihe 2, Spalte 2 mit der Verschachtelungsebene 1  
Untertabelle 1 hat eine verschachtelte Tabelle in Reihe 2, Spalte 2 mit der Verschachtelungsebene 2

Wenn eine Tabelle vorliegt, und nicht bekannt ist, ob sie verschachtelt ist, und wenn ja, in welcher Ebene, wird die NestingLevel gefragt. Gibt sie 0 (Null) zurück, ist die Tabelle nicht verschachtelt.

**Listing 6.67** Alle Zellen einer Tabelle sowie alle Zellen von eventuell darin verschachtelten Tabellen durchschleifen

```
Sub VerschachtelteTabellenFinden()
    Dim tbl As Word.Table
    Dim strListe As String
    Dim strBezeichner As String

    Set tbl = Selection.Tables(1)
    strBezeichner = "Haupttabelle"
    strListe = "Liste der verschachtelten Tabellen" & vbCrLf & vbCrLf
    ZellenNachVerschachtelteTabellenDurchsuchen tbl, strBezeichner, strListe, 1
    Debug.Print strListe
End Sub

Sub ZellenNachVerschachtelteTabellenDurchsuchen(tblStart As Word.Table, _
    strBezeichner As String, ByRef strListe As String, lZaehler As Long)
    Dim tbl As Word.Table
    Dim cel As Word.Cell
    Dim lAnzTabellen As Long

    For Each cel In tblStart.Range.Cells
        lAnzTabellen = cel.Tables.Count
        If lAnzTabellen > 0 Then
            For Each tbl In cel.Tables
                strListe = strListe & strBezeichner & _
                    " hat eine verschachtelte Tabelle in Reihe " & cel.RowIndex & _
                    ", Spalte " & cel.ColumnIndex & " mit der Verschachtelungsebene " & _
                    & tblStart.NestingLevel & vbCrLf

                ZellenNachVerschachtelteTabellenDurchsuchen tbl, "Untertabelle " & _
                    CStr(lZaehler), strListe, lZaehler + 1
            Next tbl
        End If
    Next cel
End Sub
```

Verbun-  
dene  
Zellen

Zurück nun zu den verbundenen Zellen. Word bietet kein Gegenstück zu den HTML-Eigenschaften »Rowspan« und »Colspan«, womit festgestellt wird, wie eine Tabelle strukturiert wurde. Zudem bleibt die Arbeit mit den Rows- und Columns-Auflistungen untersagt, sobald eine unterschiedliche Anzahl an Zellen in Zeilen oder Spalten vorhanden ist. Da wartet Word mit den folgenden Laufzeitfehlern auf: 5991 »Es können keine individuellen Reihen in dieser Sammlung adressiert werden, weil die Tabelle vertikal verbundene Zellen enthält.« bzw. 5992 »Es können keine individuellen Spalten in dieser Sammlung adressiert werden, weil die Zellenwerte in der Tabelle unterschiedliche Werte aufweisen.«.

Die beste Methode, um der Struktur einer Tabelle auf den Grund zu gehen ist, die Tabelle in ein neues Dokument zu kopieren, dieses als gefiltertes HTML zu speichern und die HTML Datei zu

bearbeiten. Dort stehen, wie in Abbildung 6.37 ersichtlich, nicht nur die »Rowspan«- und »Colspan«-Informationen, sondern auch Informationen über die Zellenausrichtung und Spaltenbreite liegen bereit.

**Abbildg. 6.37** Ausschnitt aus einer als gefiltertes HTML gespeicherten Word-Tabelle, im Texteditor geöffnet. Der grau hinterlegte Text hebt die nützlichen Informationen hervor.

```
<td width=224 colspan=2 valign=top style='width:167.7pt;border-top:none;
border-left:none;border-bottom:solid windowtext 1.0pt;border-right:solid
windowtext 1.0pt; padding:0cm 5.4pt 0cm 5.4pt'> <p class=MsoNormal>&nbsp;</p>
</td>
</tr>
<tr>
```

Die zweitbeste Methode, falls Word 2003 vorliegt, ist die XML-Eigenschaft den Tabellenbereich abzufragen und deren Resultat zu analysieren (mehr über Words XML-Format steht im Teil VI):

Muss es wirklich mit den Bordmitteln des Objektmodells gehen, bereiten Sie sich auf etwas Kopfzerbrechen vor. Das Listing 6.68 zeigt zwei Prozeduren, die senkrecht sowie waagrecht verbundene Zellen aufspüren. Zuerst wird mit der Uniform-Eigenschaft geprüft, ob die Tabelle symmetrisch aufgebaut ist (gleiche Anzahl von Zellen in jeder Zeile und Spalte). Wenn nicht, werden zwei weitere Prozeduren aufgerufen. Diese verschieben den Bereich von einer Zelle zur nächsten und ermitteln die Spalte bzw. Zeile mittels der Range-Eigenschaft. Diese wird mit dem Wert eines Zählers verglichen, um festzustellen, ob sie voneinander abweichen. Ist dies der Fall, wurde eine Spalte bzw. Zeile »übersprungen«, was heißt, es liegt eine verbundene Zelle vor. In Abbildung 6.38 sehen Sie eine Beispieltabelle sowie das Resultat des Listings.

**Abbildg. 6.38** Eine Tabelle mit verbundenen Zellen


Die Zellen Z2S2 bis Z4S2 sind vertikal verbunden  
 Die Zellen Z1S6 bis Z5S6 sind vertikal verbunden  
 Die Zellen Z3S4 bis Z3S5 sind horizontal verbunden  
 Die Zellen Z6S2 bis Z6S6 sind horizontal verbunden

Es ist zu beachten, dass diese zwei Prozeduren nur funktionieren, wenn die Zellen der ersten Zeile nicht horizontal und die Zellen der ersten Spalte nicht vertikal verbunden sind, da diese als die Ausgangspunkte für die Bearbeitung der Zeilen und Spalten dienen. Erwarten Sie andere Tabellenstrukturen, müssen die Prozeduren entsprechend angepasst werden.

**Listing 6.68** Prozeduren, um festzustellen, welche Zellen einer Tabelle verbunden sind

```
Sub VerbundeneTabellenZellen()
    Dim tbl As Word.Table

    Set tbl = Selection.Tables(1)
    If Not tbl.Uniform Then
```

**Listing 6.68** Prozeduren, um festzustellen, welche Zellen einer Tabelle verbunden sind (Fortsetzung)

```

    'Enthält verbundene oder geteilte Zellen
    VertikalVerbundeneZellen tbl
    HorizontalVerbundeneZellen tbl
End If
End Sub

Sub VertikalVerbundeneZellen(tbl as Word.Table)
    Dim rng As Word.Range
    Dim cel As Word.Cell
    Dim lAnzSpalten As Long
    Dim count As Long
    Dim lAktuelleZeile As Long
    Dim lAktuelleSpalte As Long

    lAnzSpalten = tbl.Columns.count
    For count = 1 To lAnzSpalten
        Set rng = tbl.Cell(1, count).Range
        lAktuelleZeile = rng.Information(wdStartOfRangeRowNumber)
        lAktuelleSpalte = rng.Information(wdStartOfRangeColumnNumber)
        Do
            lAktuelleZeile = lAktuelleZeile + 1
            rng.Collapse wdCollapseEnd
            'Am Ende der Zelle
            rng.MoveEnd wdCharacter, -1
            rng.Select
            'Eine Zeile nach unten
            Selection.MoveDown
            'Falls es noch innerhalb der Tabelle ist
            If Selection.Information(wdWithInTable) Then
                Set cel = Selection.Cells(1)
                Set rng = cel.Range
                If rng.Information(wdStartOfRangeRowNumber) <> lAktuelleZeile Then
                    Debug.Print "Die Zelle Z" & lAktuelleZeile - 1 & _
                        "S" & lAktuelleSpalte " bis Z" & cel.RowIndex - 1 & _
                        "; "s" & lAktuelleSpalte & " sind vertikal verbunden"

                    lAktuelleZeile = cel.RowIndex
                End If
            End If
        Loop While Selection.Information(wdWithInTable)
    Next
End Sub

Sub HorizontalVerbundeneZellen(tbl as Word.Table)
    Dim rng As Word.Range
    Dim cel As Word.Cell
    Dim lAnzZeilen As Long
    Dim lAnzSpalten As Long
    Dim count As Long
    Dim lAktuelleZeile As Long
    Dim lAktuelleSpalte As Long
    lAnzZeilen = tbl.Rows.count
    lAnzSpalten = tbl.Columns.count
    For count = 1 To lAnzZeilen
        Set rng = tbl.Cell(count, 1).Range
        lAktuelleZeile = rng.Information(wdStartOfRangeRowNumber)
    Next
End Sub

```



Listing 6.68 Prozeduren, um festzustellen, welche Zellen einer Tabelle verbunden sind (Fortsetzung)

```

lAktuelleSpalte = rng.Information(wdStartOfRangeColumnNumber)
Do
    lAktuelleSpalte = lAktuelleSpalte + 1
    '
    'Am Ende der Zelle
    rng.Select
    WordBasic.NextCell
    'Falls die letzte Zelle der Tabelle verbunden ist
    If tbl.Rows.count > lAnzZeilen Then
        ActiveDocument.Undo (1)
        Debug.Print "Die Zellen Z" & lAktuelleZeile &
        "S" & lAktuelleSpalte - 1 & " bis Z" & lAktuelleZeile _
        & "S" & lAnzSpalten & " sind horizontal verbunden"
        Exit Sub
    End If

    'Falls sich die Markierung noch innerhalb der Tabelle befindet
    If Selection.Information(wdWithinTable) Then
        Set cel = Selection.Cells(1)
        Set rng = cel.Range
        If rng.Information(wdStartOfRangeColumnNumber) <> lAktuelleSpalte Then
            Debug.Print "Die Zellen Z" & lAktuelleZeile &
            "S" & lAktuelleSpalte - 1 & " bis Z" & lAktuelleZeile _
            & "S" & cel.ColumnIndex - 1 & " sind horizontal verbunden"

            lAktuelleSpalte = cel.ColumnIndex
        End If
        'Debug.Print rng.Information(wdStartOfRangeRowNumber) & ", " _
        & rng.Information(wdStartOfRangeColumnNumber)
    End If
Loop While lAktuelleSpalte < lAnzSpalten

Next
End Sub

```



Die Beispieldatei *Bsp06\_04\_Table.doc* finden Sie auf der CD-ROM im Ordner *Beispiele\Kap06*.

**HINWEIS**

Mit Tabellen aus anderen Anwendungen wie Excel oder Access befassen sich der Abschnitt »Feldfunktionen« in diesem Kapitel sowie die Kapitel 10 und 11.

## Automatische Nummerierung mit Listen

In Word 97 wurde eine neue automatische Nummerierung eingeführt, die seither für viel Ärger und Unsicherheit gesorgt hat. Früher war die Nummerierung einfach zu verstehen, zuverlässig ... und wenig flexibel. Die »neue« Nummerierung ist äußerst flexibel, scheinbar intuitiv und einfach zu benutzen, aber in Wirklichkeit ist deren Handhabung anfällig und schwer auf zuverlässige Art und Weise zu implementieren.

Das grundlegende Problem liegt in der Listenverwaltung, die so tief im Hintergrund läuft, dass es für den Anwender fast unmöglich ist, zu erkennen, welche Listenelemente welcher Liste gehören.

Wird in einem Dokument viel ausprobiert und experimentiert, können die internen Strukturen an den Rand des Kollapses gelangen.

Eine eingehende Diskussion aller Aspekte würde für sich ein ganzes Buch in Anspruch nehmen. In diesem Abschnitt werden die Grundlagen der Automatisierung ausgelegt, ergänzt mit einigen Tipps, wie die schwerwiegendsten Fallen zu vermeiden sind.

## Listeigenschaften ermitteln

Im Word-Jargon stellen Aufzählungen (ob nummeriert oder mit Symbolen) *Lists* dar. Eine Gruppe nummerierter Elemente (Absätze) ist ein List-Objekt. Die Anzahl der Listen eines Dokuments wird mit der Count-Eigenschaft abgefragt:

```
ActiveDocument.Lists.Count
```

Ein List-Objekt hat einige nützliche Methoden, wie `ConvertNumbersToText` (automatisch in statische Nummerierung umwandeln) und `RemoveNumbers` (Nummerierung entfernen).

### HINWEIS

In einem Hinweis des Hilfetexts befindet sich ein Fehler. Es steht geschrieben: »Wenn diese Methode einem **List**-Objekt zugewiesen wird, werden Zahlen nur von Absätzen in der angegebenen Liste entfernt. Eingeschobene Zahlen aus anderen Listen werden entfernt.«

Es soll jedoch heißen »Eingeschobene Zahlen aus anderen Listen werden *nicht* entfernt.«

Mit der Eigenschaft `ListParagraphs` wird der Zugang zu den nummerierten Absätzen (auch wenn sie im Text nicht zusammenhängend sind) geboten. Jedes Element der `ListParagraphs`-Auflistung stellt ein gewöhnliches Paragraph-Objekt dar, wie dieses Codeschnipsel veranschaulicht:

```
Dim lst As Word.List
Dim para As Word.Paragraph
Set lst = ActiveDocument.Lists(1)
For each para in lst.ListParagraphs
    MsgBox para.Range.Text
Next
```

Bitte beachten Sie, dass `para.Range.Text` nur den Text des Absatzes, ohne Nummerierung, zurückgibt. Die Nummerierungseigenschaften werden über die `ListFormat`-Eigenschaft des Paragraph-Objekts angesprochen. Damit wird beispielsweise ermittelt, wie die Nummerierung aussieht (`ListString`), welchen Wert (`ListValue`) sie hat (unabhängig des Aussehens), sowie, bei Gliederungen, die Listebene (`ListLevel`). Diese werden in Listing 6.69 und in Abbildung 6.39 veranschaulicht.

Listing 6.69 Eigenschaften von Listen in einem Dokument ermitteln

```
Sub DasListObjekt()
    Dim doc As Word.Document, lst As Word.List, para As Word.Paragraph
    Dim lListZaehler As Long, lParaZaehler As Long, lAnzListen As Long, lAnzParas As Long
    Dim sFormatvorlage As String, sListvorlage As String, sVisuelleZiffer As String
    Dim sOrdinalZahl As String, sListEbene As String, sListTyp As String

    Set doc = ActiveDocument
```

Listing 6.69 Eigenschaften von Listen in einem Dokument ermitteln (Fortsetzung)

```

lAnzListen = doc.Lists.Count
For lListZaehler = 1 To lAnzListen
    Set lst = doc.Lists.Item(lListZaehler)
    lAnzParas = lst.ListParagraphs.Count
    MsgBox "Das aktuelle Dokument enthält " & CStr(lAnzListen) & " Liste(n)." & _
        vbCr & "Liste #" & CStr(lListZaehler) & " umfasst " & _
        CStr(lAnzParas) & " Absätze."
    For lParaZaehler = 1 To lAnzParas
        Set para = lst.ListParagraphs(lParaZaehler)
        sFormatvorlage = para.Style
        With para.Range.ListFormat
            sListvorlage = .ListTemplate.Name
            sVisuelleZiffer = .ListString
            sOrdinalZahl = CStr(.ListValue)
            sListEbene = CStr(.ListLevelNumber)
            sListTyp = CStr(.ListType)
        End With
        MsgBox "Absatz #" & CStr(lParaZaehler) & " hat folgende Eigenschaften:" & _
            vbCr & "Formatvorlage:" & vbTab & sFormatvorlage & vbCr & _
            "Listvorlage: " & vbTab & sListvorlage & vbCr & "Listebene:" & vbTab & _
            sListEbene & vbCr & "Der Wert:" & vbTab & sOrdinalZahl & _
            ", formatiert als die Zeichen: " & sVisuelleZiffer & vbCr & _
            "Die Liste ist vom Typ " & sListTyp
    Next
Next
End Sub

```

Abbildg. 6.39 Eigenschaften von Listen in einem Dokument ermitteln

Dieses Dokument enthält eine unterbrochene Liste:

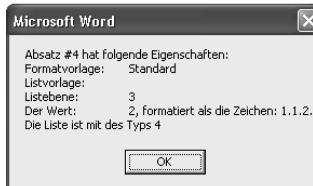
- a)→ Absatz eins
- b)→ Absatz zwei

Dazwischen liegender Text.

- c)→ Absatz drei
- d)→ Absatz vier

Eine neue Liste, mit mehreren Ebenen, folgt:

- 1.→ Ebene eins
  - 1.1.→ Ebene zwei
    - 1.1.1.→ Ebene drei
    - 1.1.2.→ Ebene drei, noch einmal
- 2.→ Ebene eins, ein zweites Mal



Die Beispieldatei *Bsp06\_01\_Num.doc* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap06*.

Mit diesen Eigenschaften können Informationen über die Nummerierung eines Dokuments ermittelt werden. Sie sollen jedoch nicht dazu gebraucht werden, um Nummerierungen vorzunehmen oder zu ändern. Statt direkt auf den Text einzuwirken, soll Automatisierungscode direkter mit den Verwaltungsstrukturen arbeiten.

## Listvorlagen (*ListTemplates*)

Im Hintergrund kennzeichnet Word jeden nummerierten Absatz mit einem Vermerk, zu welcher Liste er gehört. Die Listen werden im Dokument an einer zentralen Stelle geführt. Der Absatz schlägt an dieser Stelle nach, wie das Listenelement (also sich selbst) anzuzeigen ist.

### HINWEIS

Verhält sich ein Listenelement »komisch« (falsche Formatierung oder Nummerierung) wurde es wahrscheinlich intern mit einem anderen Listenobjekt verbunden.

Listen ihrerseits beziehen ihre Formatierungseinstellungen aus so genannten *ListTemplates* (Listvorlagen). Diese dienen als Schablonen für die Listformatierung und definieren alle Eigenschaften einer Liste. Ein *ListTemplate* umfasst entweder eine oder neun Ebenen, je nachdem, ob sie »einfach« oder »gegliedert« ist. *ListTemplates* stellen somit die Grundlage für die Nummerierung von Word dar. Um die Nummerierung eines Dokuments zu beherrschen, muss eine Kontrolle über die *ListTemplates* ausgeübt werden.

Der Anwender erfährt nie etwas von diesen Objekten, erstellt sie jedoch quasi »am laufenden Band«. Jedes Mal, wenn er eine Vorschau in *Format/Nummerierung und Aufzählungszeichen* anklickt, wird dem Dokument ein neues *ListTemplate*-Objekt zugefügt. Dokumente werden damit schnell »verseucht«, und es gibt keine Schnittstelle, diese direkt aus dem Dokument zu löschen. Auch nicht über das Objektmodell.

Ziel des Herstellers einer Dokumentvorlage muss es sein, dem Anwender Werkzeuge an die Hand zu geben, um die Nummerierung korrekt zu benutzen. Zudem sind Richtlinien für den Gebrauch unerlässlich.

### Ein Dokument von überzähligen *ListTemplates* bereinigen

Das Ansammeln von Hunderten *ListTemplates* in einem Dokument führte in früheren Word-Versionen zu Dokumentbeschädigungen. In Word 2002 und Word 2003 hat Microsoft eine Funktionalität eingebaut, die unbenutzte *ListTemplates* aus einem Dokument entfernt, sobald eine bestimmte Anzahl vorhanden ist.

Wir Menschen haben nicht die gleiche Möglichkeit wie das Word-Programm...haben folgende Möglichkeiten, *ListTemplates* aus einer Word-Datei zu entfernen:

- Die Datei in Word öffnen, den gesamten Text – ohne die letzte Absatzmarke – kopieren und in ein neues Dokument einfügen.
- Die Datei als eine Webseite oder, in Word 2003, als XML speichern. Das Resultat wird in einem Texteditor (oder als Text in Word) geöffnet, die Listenvorlagen gesucht und manuell gelöscht. Wobei sicherzustellen ist, dass keine davon mit Text im Dokument (einer Liste) verbunden ist.



Von dieser letzten Methode ist abzuraten, da die Gefahr der Dokumentbeschädigung sehr groß ist. Sie sollte nur eingesetzt werden, wenn keine andere Wahl besteht. (Allerdings eignet sie sich hervorragend dazu, um sich mit den internen Verhältnissen zwischen Listen und Absätzen vertraut zu machen.)

Listvorlagen enthalten nur Anweisungen über einen Satz von Formatierungsbefehlen. Mehrere Listen können durchaus mit einer einzigen Listvorlage verbunden sein. Folglich ist es möglich, herauszufinden, mit welcher Listvorlage eine Liste verbunden ist. Das Gegenteil jedoch nicht; eine Listvorlage gibt darüber keine Auskunft zurück, mit welchen Listen sie verbunden ist (wenn überhaupt).

## Listvorlagen erstellen

Der Entwickler hat die Möglichkeit, die Eigenschaften vorhandener ListTemplates abzufragen und anzupassen, sowie eigene zu erstellen. Dabei ist es wichtig, dass keine »wilden« Listvorlagen erstellt werden oder in der Dokumentvorlage vorhanden sind. Dies bedingt, dass die Dokumentvorlage von einer »sauberen« *Normal.dot* erstellt wird, die garantiert keine Listvorlagen enthält. Wir raten also, die aktuelle *Normal.dot* umzubenennen, so dass Word beim Starten ein sauberes Exemplar erstellt, worauf dann die neue Dokumentvorlage basiert.

### HINWEIS

In Kapitel 5 wurde die Vorbereitung der Dokumentvorlage *Normal.dot* besprochen, die allen Word-Dokumenten zu Grunde liegt.

Die  
Name-  
Eigen-  
schaft

Das ListTemplate-Objekt hat nur wenige Eigenschaften. Eine davon, und eine sehr wichtige, ist die Name-Eigenschaft. Nur benannte Listvorlagen können eindeutig identifiziert werden. Es ist dabei jedoch wichtig, dass Sie den Namen *nicht* direkt als Indexwert benutzen, da dies mehrere Listvorlagen mit der gleichen Bezeichnung zur Folge haben, und »verwaiste« Listvorlagen verursachen könnte.

Stattdessen soll der Code durch die im Dokument vorhandenen schleifen, wie die Funktion in Listing 6.70 veranschaulicht. Falls eine Listvorlage mit der angegebenen Bezeichnung bereits vorhanden ist, wird diese zurückgegeben. Sonst wird eine neue erstellt und zurückgegeben.

Listing 6.70

Funktion, um eine bestehende oder neue Listvorlage des Gliederungstyps zurückzugeben

```
Public Function ListTemplateIndex(ListTemplateName As String, _
    Source as Word.Document) As ListTemplate
    Dim LT As ListTemplate

    For Each LT In Source.ListTemplates
        If LT.Name = ListTemplateName Then
            Set ListTemplateIndex = LT
            Exit For
        End If
    Next

    If ListTemplateIndex Is Nothing Then
        '»True« bedeutet, die ListTemplate hat neun Ebenen
        Set ListTemplateIndex = Source.ListTemplates.Add(True)
        ListTemplateIndex.Name = ListTemplateName
    End If
    Set LT = Nothing
End Function
```

## Eine einfache Liste

List-Level-Objekt	Nachdem nun ein ListTemplate-Objekt vorliegt, kann das Aussehen der damit verbundenen Listen definiert werden. Dazu dienen die Listebenen – das ListLevels-Element. Wie schon erwähnt, kann eine Liste einfach (besteht aus einer einzigen Ebene) oder gegliedert sein (hat neun Ebenen). Jede dieser Ebenen ist ein ListLevel.
Outline-Numbered	Ob eine Listvorlage einfach oder gegliedert ist wird mit der Eigenschaft OutlineNumbered festgelegt (False = einfach).

Wie eine einfache Listvorlage mit einem Aufzählungszeichen erstellt wird, veranschaulicht das Listing 6.71 bzw. das Listing 6.72. Nachdem das ListTemplate-Objekt vorliegt, wird das ListLevel-Objekt an die Prozedur *ListLevelFormatierungFestlegen* übergeben, zusammen mit den Formatierungen dafür. Die Listvorlage wird mit einer Formatvorlage verbunden. Das Resultat ist in Abbildung 6.40 ersichtlich. Bitte beachten Sie, dass die Absatz-Formatvorlage bereits im Dokument vorhanden sein muss, um eine Listvorlage damit zu verbinden. Für Sinnvoll halte ich noch den zusätzlichen Hinweis, dass beim Anlegen der Formatvorlage (in diesem Fall "Meine Aufzählung") dieser **nicht** der Formatvorlagentyp "Liste", sondern der Formatvorlagentyp "Absatz" zugewiesen ist. Nebenbei gefragt: Warum muss die Formatvorlage eigentlich bereits vorhanden sein? Könnte man nicht gleichzeitig mit der Prozedur "AufzählungslisteErstellen" dem Anwender diesen Vorgang abnehmen?

CM: Könnte man tun, aber die Beispiele sollen doch eher diesen Teil des Objektmodells hervorheben, und möglichst wenig "abschweifen"..

### HINWEIS

Würde es sich um eine Listvorlage mit neun Ebenen handeln, würde jedes ListLevel-Objekt an die Prozedur übergeben, bis alle definiert sind.

Die weiteren verwendeten ListLevel-Eigenschaften sind in Tabelle 6.14 beschrieben.

Listing 6.71 Eine Listvorlage definieren

```
Sub AufzählungslisteErstellen()
    Dim doc As Word.Document
    Dim LTAufzählung As Word.ListTemplate
    Dim sLTName As String
    Dim lAufzählungZeichencode As Long
    Dim LL As Word.ListLevel

    Set doc = ActiveDocument
    sLTName = "Pfeil-Aufzählung"
    lAufzählungZeichencode = 220 'Wingdings
    Set LTAufzählung = ListTemplateIndex(sLTName, doc)
    LTAufzählung.OutlineNumbered = False
    Set LL = LTAufzählung.ListLevels(1)
    ListLevelFormatierungFestlegen LL, wdListLevelAlignLeft, wdListNumberStyleNone, _
        ChrW$(lAufzählungZeichencode), 0, CentimetersToPoints(0.7),
        CentimetersToPoints(0.7), wdTrailingTab, False, 1, "Wingdings", False, _
        "Meine Aufzählung"
End Sub

Private Sub ListLevelFormatierungFestlegen(LL As Word.ListLevel, _
    LL_Ausrichtung As WdListLevelAlignment, LL_NummerArt As WdListNumberStyle, _
    LL_NummerFormat As Long, LL_NummerPos As Single, LL_TabPos As Single, _
```

Listing 6.71 Eine Listvorlage definieren (Fortsetzung)

```

LL TextPos As Single, LL FolgeZeichen As WdTrailingCharacter,
LL Restart As Boolean, LL BeginnenBei As Long, LL Schrift As String, _
LL IstFett As Boolean, LL Formatvorlage As String)

With LL
    .Alignment = LL_Ausrichtung
    .Font.Name = LL_Schrift
    .Font.Bold = LL_IstFett
    .NumberStyle = LL_NummerArt
    .NumberFormat = ChrW$(LL_NummerFormat)
    .NumberPosition = LL_NummerPos
    .ResetOnHigher = LL_Restart
    .StartAt = LL_BeginnenBei
    .TabPosition = LL_TabPos
    .TextPosition = LL_TextPos
    .TrailingCharacter = LL_FolgeZeichen
    .LinkedStyle = LL_Formatvorlage
End With
End Sub

```

Listing 6.72 (.NET): Die C#-Version

```

private void AufzählungslisteErstellen_CS()
{
    wd.Document doc = wdApp.ActiveDocument;
    string LTName = "Pfeil-Aufzählung";
    wd.ListTemplate LTAufzählung = ListTemplateIndex(LTName, doc);
    LTAufzählung.OutlineNumbered = false;
    wd.ListLevel LL = LTAufzählung.ListLevels[1];
    ListLevelFormatierungFestlegen(LL, wd.WdListLevelAlignment.wdListLevelAlignLeft,
        wd.WdListNumberStyle.wdListNumberStyleNone, "ü",
        Of, wdApp.CentimetersToPoints(0.7f), wdApp.CentimetersToPoints(0.7f),
        wd.WdTrailingCharacter.wdTrailingTab, 0, 1, "Wingdings",
        -1, "Meine Aufzählung");
}

private wd.ListTemplate ListTemplateIndex(string ListTemplateName,
    wd.Document Source)
{
    wd.ListTemplate templT = null;
    //Gibt die Listvorlage mit dem angegebenen Namen zurück.
    //Ist keine mit dieser Bezeichnung vorhanden, wird eine neue erstellt.
    foreach (wd.ListTemplate LT in Source.ListTemplates)
    {
        if (LT.Name == ListTemplateName)
        {
            templT = LT;
            break;
        }
    }
    if (templT == null)
    { // "True" bedeutet, die ListTemplate hat neun Ebenen
        object objTrue = true;
        object objName = ListTemplateName;
        templT = Source.ListTemplates.Add(ref objTrue, ref objName);
    }
}

```

**Listing 6.72** (.NET): Die C#-Version (*Fortsetzung*)

```

    }
    return tempLT;
}

private void ListLevelFormatierungFestlegen(wd.ListLevel LL,
    wd.WdListLevelAlignment LL_Ausrichtung,
    wd.WdListNumberStyle LL_NummerArt, string LL_NummerFormat,
    float LL_NummerPos, float LL_TabPos, float LL_TextPos,
    wd.WdTrailingCharacter LL_FolgeZeichen, int LL_Restart,
    int LL_BeginnenBei, string LL_Schrift, int LL_IstFett,
    string LL_Formatvorlage)
{
    try
    {
        LL.Alignment = LL_Ausrichtung;
        LL.Font.Name = LL_Schrift;
        LL.Font.Bold = LL_IstFett;
        LL.NumberStyle = LL_NummerArt;
        LL.NumberFormat = LL_NummerFormat;
        LL.NumberPosition = LL_NummerPos;
        LL.ResetOnHigher = LL_Restart;
        LL.StartAt = LL_BeginnenBei;
        LL.TabPosition = LL_TabPos;
        LL.TextPosition = LL_TextPos;
        LL.TrailingCharacter = LL_FolgeZeichen;
        LL.LinkStyle = LL_Formatvorlage;
    }
    catch (System.Exception ex)
    { System.Windows.Forms.MessageBox.Show(ex.Message); }
}

```

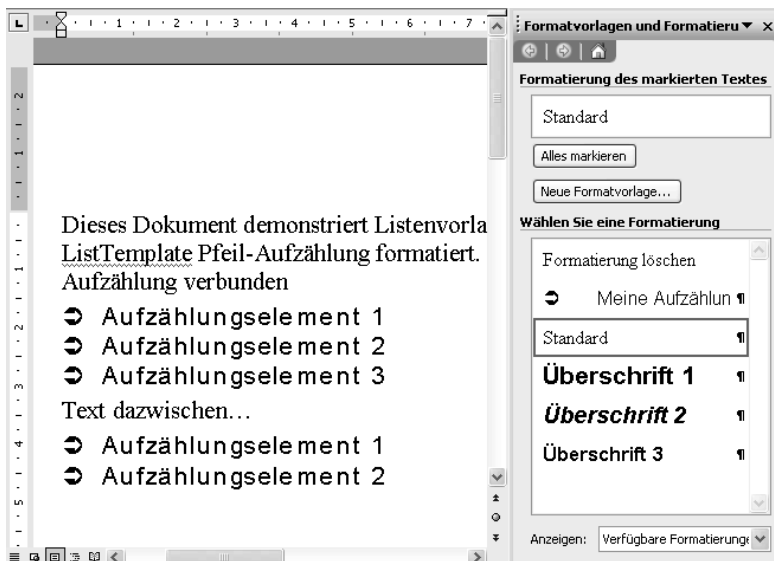
**Abbildg. 6.40** Das Aufzählungsformat wurde programmtechnisch erstellt und mit der Formatvorlage verbunden




Tabelle 6.14 Die Eigenschaften, die die Formatierung einer jeden Ebene einer Aufzählung bestimmen

ListLevel-Eigenschaft	Mögliche Werte/Bemerkungen
Alignment	WdListLevelAlignment-Konstantwerte: wdListLevelAlignCenter wdListLevelAlignLeft wdListLevelAlignRight Ausrichtung des Aufzählungszeichens im Raum zwischen dem linken Rand und des Einzuges.
Font	Die Schriftart des Aufzählungszeichens oder Nummer. Hat keine Wirkung auf den Text des Absatzes. Wird nichts angegeben, übernehmen die Zeichen die Scharfteigenschaften des Absatzes.
NumberFormat	Akzeptiert eine Zeichenkette. Stellt das Aussehen der Aufzählung oder Nummerierungsebene dar. Es darf auch Zeichen beinhalten. Ebenen werden mit »%n« angegeben. Um beispielsweise in der dritten Gliederungsebene »Abschnitt 3.1.1« zu sehen, wäre der Wert für diese Eigenschaft: Abschnitt %1.%2.%3
NumberPosition	Akzeptiert einen Wert des Typs <b>Single</b> , angegeben in Points. Stellt die Position der Aufzählung relativ zum linken Rand dar. Ein negativer Wert positioniert die Aufzählung am linken Textrand.
NumberStyle	WdListNumberStyle-Konstantwert. Die üblichsten in westlichen Länder sind: wdListNumberStyleNone wdListNumberStyleArabic wdListNumberStyleBullet wdListNumberStyleLegal wdListNumberStyleLowercaseLetter wdListNumberStyleLowercaseRoman wdListNumberStyleUppercaseLetter wdListNumberStyleUppercaseLetter Legt die Art Aufzählungszeichen fest (arabische, römische, Buchstaben usw.)
RestartOnHigher	Legt fest, ob die Nummerierung neu beginnt, wenn sie auf einer Aufzählung einer höheren Ebene folgt.
StartAt	Die Nummer, womit eine Ebene anfängt, wenn sie neu beginnt.
TabPosition	Akzeptiert einen Wert des Typs <b>Single</b> , angegeben in Points. Stellt die Position des Tabstopps relativ zum linken Rand dar. Der Text der ersten Zeile fängt hier an.
TextPosition	Akzeptiert einen Wert des Typs <b>Single</b> , angegeben in Points. Legt den Einzug für die zweiten und folgenden Textzeilen fest. Hat Vorrang vor der Einzugseinstellung der Absatzformatvorlage und ersetzt diese.
TrailingCharacter	WdTrailingCharacter-Konstantwert. wdTrailingNone wdTrailingSpace wdTrailingTab Legt das Zeichen fest, das dem Aufzählungszeichen folgt. Meistens wird ein Tab-Zeichen gewählt, um die <b>TabPosition</b> anzuspringen. Diese und <b>TextPosition</b> haben keine Wirkung, wenn <b>TrailingCharacter</b> nicht <b>wdTrailingTab</b> entspricht.

**Tabelle 6.14** Die Eigenschaften, die die Formatierung einer jeden Ebene einer Aufzählung bestimmen (Fortsetzung)

ListLevel-Eigenschaft	Mögliche Werte/Bemerkungen
LinkedStyle	Optional. Verbindet die Listvorlage mit einem Absatzformat. Besteht eine Verbindung, wird die Aufzählung automatisch zugewiesen, als ob sie Teil der Formatvorlage wäre. Jede Listebene kann mit einer anderen Formatvorlage verbunden werden.



Die Beispieldatei *Bsp06\_02\_Num.doc* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap06*.

## Eine Gliederungsliste der anderen Art

Oft wird gefragt, ob statischer Text als Teil einer Formatvorlage definiert werden könnte. Leider lautet die Antwort darauf: »Nein«. Einen Umweg gibt es dennoch, wenn eine Listvorlage mit der Formatvorlage verbunden wird. Aus der Erläuterung zur NumberFormat-Eigenschaft ist hervorgegangen, dass statischer Text einen Teil des Nummernbildes bilden kann. Wird eine Listvorlage mit der Formatvorlage verknüpft, erscheint dieser Text am Absatzanfang, wie in Abbildung 6.41 ersichtlich.

Die Listvorlage wurde mit dem Code in Listing 6.73 erstellt. Im Gegensatz zu Listing 6.71 wird eine Listvorlage des Typs Gliederung festgelegt, die neun Listebenen zur Verfügung stellt. Deren fünf werden in dieser Prozedur definiert.

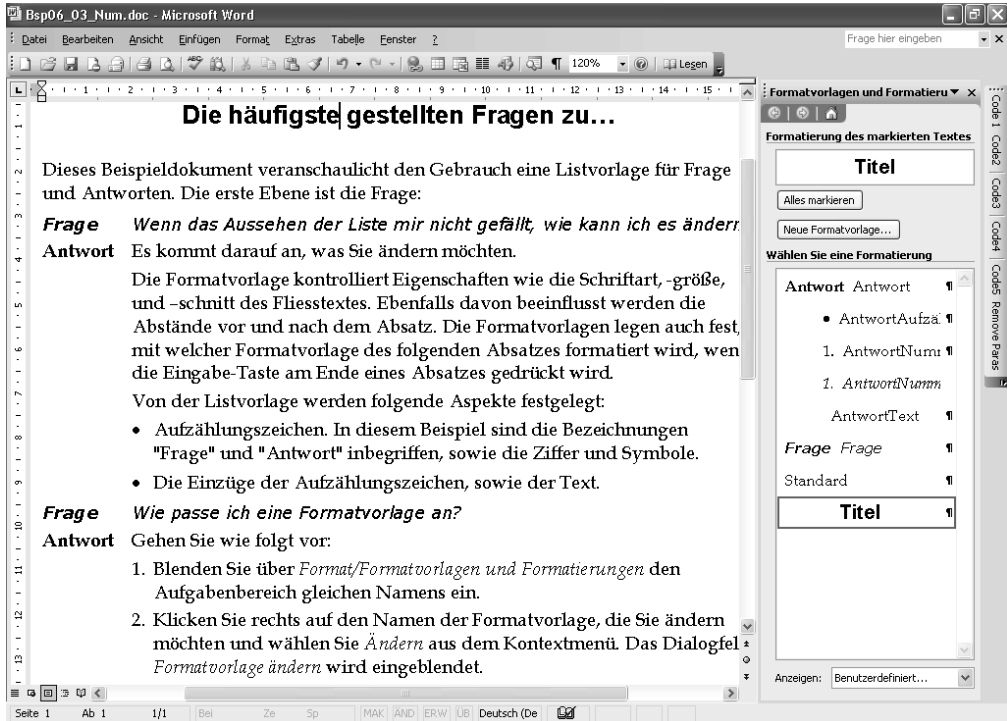
### HINWEIS

Word weist bei der Erstellung einer Listvorlage allen neun Ebenen eine Standardformatierung zu. Wenn unbenutzte Ebenen ein Problem darstellen, sollen die nicht verwendeten Ebenen »neutral« formatiert werden. Dies könnte entweder ohne Aufzählungszeichen und ohne Einzüge sein, oder mit ähnlichen Einstellungen wie die letzte »echte« Ebene. Solche Ebenen werden meist nicht mit einer Formatvorlage verbunden.

Die Formatvorlagen sind so definiert, dass ein Antwort-Absatz direkt auf einen Frage-Absatz folgt. Ein als *AntwortText* formatierter Absatz folgt auf den *Antwort*-Absatz. Mit der Einfügemarke in einem *AntwortText*-Absatz kann durch Drücken der Tastenkombination + + die *AntwortNummeriert*-Formatierung angewendet werden. Ein nochmaliges Drücken formatiert den Absatz mit *AntwortAufzählung*. Die Tastenkombination + + wechselt zu einer höheren Listebene.

Oder der Anwender wählt – ohne lange zu überlegen – die korrekte Formatierung im Aufgabenbereich aus. Die Schnittstellen zu den Formatierungsbefehlen für die Nummerierung könnten entfernt werden, so dass der Benutzer nur die vordefinierten Listvorlagen verwendet.

Abbildg. 6.41 Die ersten fünf Ebenen der Listvorlage definieren die Elemente eines Fragen-und-Antwort-Dokuments



Listing 6.73 Fünf Ebenen einer Listvorlage vom Typ *Gliederung* definieren

```
Sub FAQListvorlageErstellen()
    Dim doc As Word.Document
    Dim LTAufzählung As Word.ListTemplate
    Dim sLTName As String
    Dim lAufzählungZeichencode As Long
    Dim LL As Word.ListLevel

    Set doc = ActiveDocument
    sLTName = "FAQ"
    Set LTAufzählung = ListTemplateIndex(sLTName, doc)
    LTAufzählung.OutlineNumbered = True
    Set LL = LTAufzählung.ListLevels(1)
    ListLevelFormatierungFestlegen LL, wdListLevelAlignLeft, wdListNumberStyleNone, _
        "Frage", 0, CentimetersToPoints(2), CentimetersToPoints(2), _
        wdTrailingTab, False, 1, "", True, "Frage"
    Set LL = LTAufzählung.ListLevels(2)
    ListLevelFormatierungFestlegen LL, wdListLevelAlignLeft, wdListNumberStyleNone, _
        "Antwort", 0, CentimetersToPoints(2), CentimetersToPoints(2), _
        wdTrailingTab, False, 1, "", True, "Antwort"
    Set LL = LTAufzählung.ListLevels(3)
    ListLevelFormatierungFestlegen LL, wdListLevelAlignLeft, wdListNumberStyleNone, _
        "", CentimetersToPoints(2), CentimetersToPoints(2), CentimetersToPoints(2), _
        wdTrailingNone, False, 1, "", False, "AntwortText"
    Set LL = LTAufzählung.ListLevels(4)
```

**Listing 6.73** Fünf Ebenen einer Listvorlage vom Typ *Gliederung* definieren (Fortsetzung)

```
ListLevelFormatierungFestlegen LL, wdListLevelAlignLeft, wdListNumberStyleArabic,
"%4.", CentimetersToPoints(2), CentimetersToPoints(2.5), CentimetersToPoints(2.5), _
wdTrailingTab, True, 1, "", False, "AntwortNummeriert"
Set LL = LTAufzählung.ListLevels(5)
ListLevelFormatierungFestlegen LL, wdListLevelAlignLeft, wdListNumberStyleNone,
".", CentimetersToPoints(2), CentimetersToPoints(2.5), CentimetersToPoints(2.5), _
wdTrailingTab, True, 1, "Symbol", False, "AntwortAufzählung"
End Sub
```

Die Prozedur *FundAExtrahieren* in Listing 6.74 zeigt, wie einfach es ist, den Inhalt aller Absätze der ersten zwei Gliederungsebenen aus dem Dokument herauszulesen, um eine Zusammenfassung des Dokuments zu erstellen.

**Listing 6.74** Die Fragen und erster Absatz jeder Antwort als Zusammenfassung in ein neues Dokument übernehmen

```
Public Sub FundAExtrahieren()
    Dim docQuelle As Word.Document
    Dim docNeu As Word.Document
    Dim rng As Word.Range
    Dim para As Word.Paragraph

    Set docQuelle = ActiveDocument
    Set docNeu = Application.Documents.Add
    Set rng = docNeu.Content
    For Each para In docQuelle.Lists(1).ListParagraphs
        If para.Range.ListFormat.ListLevelNumber <= 2 Then
            rng.FormattedText = para.Range.FormattedText
            'Da diese Auflistung vom Dokumentende bis zum Dokumentanfang
            'durchschleift wird, wird jeder zusätzlicher Absatz am
            'Anfang des Bereichs eingefügt.
            rng.Collapse wdCollapseStart
        End If
    Next
End Sub
```



Die Beispieldatei *Bsp06\_03\_Num.doc* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap06*.

## Feldfunktionen

Nicht nur Automatisierungscode sorgt in Word für einen dynamischen Dokumentinhalt. Die Word-Anwendung bietet über 70 Feldfunktionen, womit Informationen wie Seitennummer, Datum, Dateiname und Dokumenteigenschaften im Text angezeigt werden können. Zusätzlich sind sie auch für Verweise und Verknüpfungen verantwortlich. Es macht Sinn, in einem Dokumentprojekt möglichst die Word-internen Fähigkeiten einzusetzen, um Zeit zu sparen und unnötigen Aufwand zu vermeiden.

**HINWEIS** In der Benutzerschnittstelle öffnen Sie über den Menübefehl *Einfügen/Feld* das Dialogfeld *Feld*. Seit Word 2000 werden englische Bezeichnungen in allen lokalen Sprachversionen gebraucht, weshalb eine vollständige Liste mit den früheren deutschen Bezeichnungen zum Nachschlagen im Anhang B für Sie bereitsteht. Außerdem finden Sie dort weitere allgemeine Informationen zum Thema Feldfunktionen. Detaillierte Angaben über Zweck und Verwendung einzelner Feldfunktionen finden Sie in den Hilfe-Dateien von Word. Suchen Sie dort nach dem Begriff »*Feldname* Feld« (ohne die Anführungszeichen).

In diesem Abschnitt werden Feldfunktionen vom Entwicklerstandpunkt aus betrachtet, mit einigen kurzen Beispielen für deren Einsatz.

Feldfunktionen werden im Word-Objektmodell mit dem `Field`-Objekt und der `Fields`-Ausflistung dargestellt. Diese sind ihrerseits Eigenschaften eines `Document`-, `Range`- oder `Selection`-Objekts.

Update-  
Methode

Word aktualisiert Feldfunktionen nicht laufend; das würde zu viele Ressourcen benötigen und Word würde extrem langsam laufen. Irgendeine Handlung muss also die Aktualisierung auslösen. Zudem werden nicht alle Feldfunktionen durch die gleichen Handlungen aktualisiert. Um sicherzustellen, dass Feldresultate auf dem aktuellsten Stand sind, muss die Aktualisierung ausdrücklich erfolgen, was im Code mit der `Update`-Methode erreicht wird. Um alle im Haupttextteil vorhandenen Feldfunktionen zu aktualisieren, genügt die folgende Anweisung:

```
doc.Fields.Update
```

**HINWEIS** Welche Auslöser welche Feldfunktionen aktualisieren, ist nicht gut dokumentiert. Die einzige uns bekannte Quelle ist der englische Knowledge Base-Artikel für Word 97, »WD97: Fields Updated When Document Repaginated«, zu finden unter <http://support.microsoft.com/?kbid=89953/en-us>. Seit dieser Version wurde zwar die eine oder andere Feldfunktion anders eingestuft, der Artikel bietet jedoch einen Einblick in die interne Logik.

**Kind** Sie können über das Objektmodell mit der `Kind`-Eigenschaft herausfinden, zu welcher allgemeinen Aktualisierungskategorie ein bestimmtes Feld gehört. Diese Eigenschaft gibt einen von vier `WdFieldKind`-Konstantwerten zurück, die in der Tabelle 6.15 aufgelistet sind. Ein Beispiel für ihren Einsatz befindet sich in Listing 6.76.

**Tabelle 6.15** Die `WdFieldKind`-Konstantwerte

<code>WdFieldKind</code> -Enum	Wert	Beschreibung
<code>wdFieldKindCold</code>	3	Ein Feld, das kein Ergebnis hat (z. B. XE-Felder (Indexeintrag), TC-Felder (Verzeichniseintrag) oder private Felder).
<code>wdFieldKindHot</code>	1	Ein Feld, das automatisch jedes Mal, wenn es angezeigt wird oder wenn die Seite neu formatiert wird, aktualisiert wird, das aber auch manuell aktualisiert werden kann (z. B. INCLUDE-PICTURE oder FORMDROPDOWN).
<code>wdFieldKindNone</code>	0	Ein ungültiges Feld (z. B. zwei Feldzeichen ohne Inhalt).

Tabelle 6.15 Die *WdFieldKind*-Konstantenwerte (Fortsetzung)

WdFieldKind-Enum	Wert	Beschreibung
wdFieldKindWarm	2	Ein Feld, das aktualisiert werden kann und ein Ergebnis hat. Diese Art umfasst sowohl Felder, die automatisch bei Änderungen der Quelle aktualisiert werden, als auch Felder, die manuell aktualisiert werden können (z. B. DATE oder INCLUDETEXT).

**TIPP**

Obwohl das Inhaltsverzeichnis durch eine Feldfunktion (TOC) generiert wird, löst die Update-Methode keine Aktualisierung dieser Feldfunktion aus. Jedes Inhaltsverzeichnis muss ausdrücklich über das entsprechende TablesOfContents-Objekt aktualisiert werden. Das erste Inhaltsverzeichnis eines Dokuments wird demzufolge so aktualisiert:

```
ActiveDocument.TablesOfContents(1).Update
```

Lock-  
Eigen-  
schaft  
Unlink-  
Methode

Unter Umständen soll der dynamische Inhalt eines Dokuments sich nicht mehr aktualisieren, beispielsweise wenn das Dokument außer Haus geschickt oder archiviert wird. Word bietet hierzu zwei Möglichkeiten: die Feldfunktionen sperren oder sie in gewöhnlichen Text bzw. eingebettete Objekte umwandeln.

Mit der Eigenschaft `Locked` wird ermittelt bzw. festgelegt, ob eine Feldfunktion gesperrt ist oder sich aktualisieren lässt. Der Wert `True` bedeutet, sie ist gesperrt; `False`, dass sie aktualisiert werden kann.

Im Gegensatz zur Sperrung ist das Umwandeln des Feldergebnisses in Text oder ein eingebettetes Objekt nicht widerrufbar. Dafür ist die `Unlink`-Methode zuständig. Das folgende Codeschnipsel veranschaulicht, wie die Feldfunktionen im Dokumenttext gesperrt, während diejenigen der Kopfzeile des ersten Abschnitts in statischen Text umgewandelt werden.

```
doc.Fields.Locked = True
doc.Sections(1).Headers(wdHeaderFooterPrimary).Range.Fields.Unlink
```

Mehr über Abschnitte sowie Kopf- und Fußzeilen erfahren Sie im Abschnitt »Abschnitte im Dokument: Das *Section*-Objekt« in diesem Kapitel.

**HINWEIS**

Wie in der Diskussion »Das ganze Dokument mit VBA durchsuchen« des Abschnitts »Die Nadel im Heuhaufen: *Find/Replace* einsetzen« in diesem Kapitel erklärt, besteht ein Dokument aus mehreren Teilen. Um sämtliche Feldfunktionen in allen Teilen zu aktualisieren, zu sperren oder aufzulösen, müssen alle Dokumentteile, wie in der erwähnten Diskussion vorgestellt, angesprochen werden.



Hinter jeder Feldfunktion steckt ein Feldcode. Dieser besteht aus einem Paar Feldklammern, einem Feldnamen, sowie keinem oder mehreren Schaltern, die das Verhalten oder das Resultat beeinflussen. Beim ersten Blick sehen die Klammern wie normale geschweifte Klammern aus, sind es aber nicht. Feldklammern können nur über einen Word-Befehl mit der Tastenkombination `[Strg] + [F9]` oder über das Objektmodell eingefügt werden. Der Feldname ist immer ein Wort in englischer Sprache, das die Funktion mehr oder weniger beschreibt.

Schalter werden immer durch ein Backslash, gefolgt von einem Buchstaben signalisiert. Jeder Buchstabe steht für eine bestimmte, für die jeweilige Feldfunktion spezifische Option, die in der Word-Hilfe näher erläutert wird. Unter Umständen folgt eine definierende Angabe, die meist (aber nicht immer) in Anführungszeichen steht.

**TIPP**

Die Feldcodes lassen sich mit der Tastenkombination **Alt + F9** ein- und ausblenden.

Ein Beispiel ist in Abbildung 6.42 abgebildet. Wie der Feldname »INDEX« andeutet, ist das Resultat dieser Feldfunktion ein Index: eine Liste aller Indexeinträge – XE-Feldfunktionen – in einem Dokument. Es wird von Word bei der Bestätigung des Dialogfelds *Index*, unter dem Menüpunkt *Einfügen/Referenz/Index und Verzeichnisse*, eingefügt. Die Schalter \e, \c, und \z legen die Trennzeichen zwischen einem Indexeintrag und der zugehörigen Seitenzahl, die Anzahl der Spalten sowie die Sprache für die Sortierreihenfolge fest. Jede Feldfunktion hat einen eigenen Satz von Schaltern; der gleiche Buchstabe kann für mehrere Feldfunktionen eine ganz andere Bedeutung haben.

**Abbildg. 6.42** Eine Feldfunktion, um einen Index zu generieren. Die graue Schattierung wird über *Extras/Optionen/Ansicht* geregelt.

```
{INDEX \e " " \c "2" \z "1031" }
```

Add-  
Methode

Im Automatisierungscode wird eine Feldfunktion mit der Methode `Fields.Add(Range, [Type], [Text], [PreserveFormatting])` eingefügt. Das Argument `Range` ist erforderlich und gibt den Bereich an, wo die Feldfunktion einzufügen ist. Wird der Typ nicht angegeben, werden leere Feldklammern, ohne Feldnamen, eingefügt. Im Argument `Text` wird der gesamte, restliche Inhalt, bis zur abschließenden Klammer festgelegt. Schließlich gibt `PreserveFormatting` an, ob der Schalter `\* MergeFormat` hinzugefügt werden soll.

Die Eigenschaft `Type` erwartet einen `WdFieldType`-Konstantwert. Eine Liste davon, mit den englischen und früheren deutschen Feldnamen gegenübergestellt, finden Sie im Anhang B. Die Konstantwerte entsprechen zum großen Teil den englischen Feldnamen. Sie müssen aber nicht unbedingt den Type angeben. Es ist durchaus erlaubt, den Feldnamen als Teil des Text-Arguments anzugeben.

Wir empfehlen, `PreserveFormatting` auf `False` zu setzen, außer Sie wollen ausdrücklich den `\* MergeFormat`-Schalter im Feldcode haben. Dieser Schalter speichert im Feldergebnis vorgenommene Zeichenformatierungen, so dass sie bei der Aktualisierung beibehalten werden. Da sich aber die Position der Zeichen bei der Aktualisierung ändern könnte, steht die Formatierung allzu oft am falschen Ort, oder kann nicht aus dem Feld entfernt werden. `\* MergeFormat` ist vor allem nützlich in *IncludePicture*-Feldfunktionen (siehe den Grafiken-Abschnitt »Verknüpfungen erstellen und verwalten« in diesem Kapitel), um die Grafikgröße festzuhalten, und in Feldern, dessen Ergebnis eine Tabelle ist (*Link* und *Database*).

Das *Index*-Feld in Abbildung 6.42 kann auf eine der zwei beschriebenen Arten, mit oder ohne Angabe des Type-Arguments, eingefügt werden:

```

strQuote = Chr$(34) 'Anführungszeichen
Set rng = Selection.Range
'Mit Angabe des Typs
Set fld = rng.Fields.Add(Range:=rng, Type:=wdFieldIndex, Text:="\e " & strQuote & _
    vbTab & strQuote & " \c " & strQuote & "2" & strQuote & " \z " & strQuote & "1031" _
    & strQuote, PreserveFormatting:=False)
'Ohne Angabe des Typs
Set fld = rng.Fields.Add(Range:=rng, Text:="Index \e " & strQuote & _
    vbTab & strQuote & " \c " & strQuote & "2" & strQuote & " \z " & strQuote & "1031" _
    & strQuote, PreserveFormatting:=False)

```

In C#:

```

string quote = "\"; //Anführungszeichen;
wd.Range rng = wdApp.Selection.Range;
//Mit Angabe des Typs
object objFieldType = (object) wd.WdFieldType.wdFieldIndex;
object objFieldText = (object) ("\\e " + quote + "\\t" + quote
    + " \\c " + quote + "2" + quote + " \\z " + quote + "1031" + quote);
object objFalse = false;
wd.Field fld = rng.Fields.Add(rng, ref objFieldType, ref objFieldText, ref objFalse);
fld = null;
//Ohne Angabe des Typs
objFieldText = (object) ("Index \\e " + quote + "\\t" + quote
    + " \\c " + quote + "2" + quote + " \\z " + quote + "1031" + quote);
fld = rng.Fields.Add(rng, ref objMissing, ref objFieldText, ref objFalse);

```

#### HINWEIS

Nicht alle Feldfunktionen können überall in ein Dokument eingefügt werden, wo Text stehen kann. Insbesondere mahnen wir bei AutoFormen (Textfeldern) zur Vorsicht. Auch wenn Word es zulässt, eine Feldfunktion in ein Textfeld einzufügen, ist es nicht gewährleistet, dass sie korrekt aktualisiert oder von Word »gesehen« wird. Das Paradebeispiel hierfür stellen Querverweise und Einträge für Inhaltsverzeichnisse dar, welche von Word vollkommen ignoriert werden. Wenn sich eine Feldfunktion in einem Bereich befinden soll, der mit Textflussformatierung umgeben wird, benutzen Sie am besten eine Tabelle oder einen Positionsrahmen.

Eine Feldfunktion hat zwei Aspekte: den Feldcode und das Feldergebnis. Entsprechend stellt das Field-Objekt zwei Eigenschaften zur Verfügung, um die Arbeit mit beiden Moden zu ermöglichen: Code sowie Result. Jede dieser Eigenschaften gibt einen Range zurück. Somit kann ein Feldcode direkt bearbeitet werden, ohne die Bildschirmanzeige ändern zu müssen.

In der Diskussion »Verknüpfungen erstellen und verwalten« des Abschnitts »Grafiken: Die *InlineShape*- und *Shape*-Objekte« in diesem Kapitel werden Pfadangaben verknüpfter Grafiken diskutiert. Gelegentlich muss der Pfad zur Quelldatei angepasst werden; was auf verschiedene Wege unternommen werden kann. Eine Möglichkeit ist, den Feldcode der *IncludePicture*-Feldfunktion direkt zu bearbeiten. Im folgenden Beispiel werden alle Feldfunktionen im Dokumenthaupttext durchschleift und geprüft, ob sie des Typs *wdFieldInlineShape* sind. Wenn ja, wird die Pfadangabe im Feldcode geändert und anschließend die Feldfunktion aktualisiert. Diese Methode lässt sich für andere Feldfunktionen ebenfalls anwenden, die Verknüpfungen verwalten.



**HINWEIS**

In der Benutzerschnittstelle müssten die Feldcodes eingeblendet und mit *Suchen und Ersetzen* bearbeitet werden. Bei einer Bearbeitung des Feldcodes über das Range-Objekt bleibt der Bildschirm ruhig

Listing 6.75

Die Pfadangabe einer jeden Grafik, die in der Zeile mit dem Text steht, ändern

```
Sub GrafikPfadAnpassen()
    Dim doc As Word.Document
    Dim rng As Word.Range
    Dim fld As Word.Field
    Dim bMergeFormatSchalter As Boolean
    Dim bSaveDataSchalter As Boolean
    Dim strFeldCode As String
    Dim strNeuerPfad As String
    Dim strMergeFormat As String
    Dim strSaveData As String
    Dim strGrafikName As String
    Dim lPosFeldName As Long
    Dim strNeuerFeldCode As String

    strNeuerPfad = "C:\\Windows\\"
    strMergeFormat = UCase("\\* MergeFormat")
    strSaveData = LCase("\\d")
    strFeldCode = ""
    strGrafikName = ""
    strNeuerFeldCode = ""
    Set doc = ActiveDocument
    For Each fld In doc.Fields
        If fld.Type = wdFieldIncludePicture Then
            Set rng = fld.Code
            strFeldCode = rng.Text
            'Festhalten, ob der Schalter vorhanden ist, und entferne ihn aus dem Code
            If InStr(UCase(strFeldCode), strMergeFormat) <> 0 Then
                bMergeFormatSchalter = True
                strFeldCode = Replace(strFeldCode, strMergeFormat, "")
            End If
            'Festhalten, ob der Schalter vorhanden ist, und entferne ihn aus dem Code
            If InStr(LCase(strFeldCode), strSaveData) <> 0 Then
                bSaveDataSchalter = True
                strFeldCode = Replace(strFeldCode, strSaveData, "")
            End If
            'Feldname und Leerzeichen entfernen
            strFeldCode = Trim(Replace(UCase(strFeldCode), "INCLUDEPICTURE", ""))
            lPosFeldName = InStrRev(strFeldCode, "\\")
            If lPosFeldName = 0 Then lPosFeldName = InStrRev(strFeldCode, "/")
            If lPosFeldName <> 0 Then
                strNeuerFeldCode = "IncludePicture " & strNeuerPfad & _
                    Mid(strFeldCode, lPosFeldName + 1)
                If bMergeFormatSchalter Then strNeuerFeldCode = strNeuerFeldCode & _
                    & " " & strMergeFormat
                If Not bSaveDataSchalter Then strNeuerFeldCode = strNeuerFeldCode & _
                    " " & strSaveData
                fld.Code.Text = strNeuerFeldCode
                fld.Update
            End If
        End If
    Next fld
End Sub
```



Die Beispieldatei *Bsp06\_01\_Field.doc* finden Sie auf der CD-ROM im Ordner *\Beispiele\Kap06*.

Wenn eine Feldfunktion mit der rechten Maustaste angeklickt wird, erscheinen feldspezifische Optionen im Kontextmenü; Word erkennt, wenn sich die Markierung in einer Feldfunktion befindet. Das Objektmodell enthält hierfür kein Gegenstück, und anders als für viele Objekte gibt `Selection.Fields(1)` kein `Field`-Objekt zurück, wenn sich die Markierung in einer Feldfunktion befindet. Mit Hilfe der `InRange`-Eigenschaft ist es möglich, herauszufinden, ob sich die Markierung in einer Feldfunktion befindet, wie das Listing 6.76 veranschaulicht.

Der Markierungsbereich wird mit dem Bereich der letzten bis zu diesem Punkt sich im Dokument befindenden Feldfunktion verglichen. Überschneiden sich die Bereiche, steht die Markierung in einer Feldfunktion. Bitte beachten Sie, wie bei »kalten« Feldfunktionen, die kein Resultat anzeigen – wie etwa *XE*, *RD* und *TC* Felder –, der Feldcode als Bereich genommen wird, während für »normale« Feldfunktionen das Feldresultat herangezogen wird; »verborgene« Feldfunktionen geben kein Resultat zurück.

**Listing 6.76** Ermitteln, ob sich die Markierung in einer Feldfunktion befindet

```
Function IstMarkierungImFeld() as Boolean
    Dim lAnzFelder As Long
    Dim rngDok As Word.Range
    Dim rngMarkierung As Word.Range
    Dim fld As Word.Field
    Dim bInFeld As Boolean

    bInFeld = False
    Set rngMarkierung = Selection.Range
    Set rngDok = rngMarkierung.Duplicate
    'Der Bereich erstreckt sich vom Dokumentanfang ...
    rngDok.Start = ActiveDocument.Range.Start
    '... bis zu einem Zeichen nach der Markierung, falls die Markierung
    'am Anfang eines Feldes steht (das Feld ist grau hinterlegt).
    rngDok.End = rngMarkierung.End + 1
    'Auch der markierte Bereich wird um ein Zeichen erweitert
    rngMarkierung.End = rngMarkierung.End + 1
    'Die Anzahl der Felder bis zur Markierung ermitteln
    lAnzFelder = rngDok.Fields.Count
    Set fld = ActiveDocument.Fields(lAnzFelder)

    'Falls ein verborgenes Feld wie XE, RD oder TC vorliegt ...
    If fld.Kind = wdFieldKindCold Then
        '... wird der Feldcodebereich mit dem Markierungsbereich verglichen.
        If rngMarkierung.InRange(fld.Code) Then
            Debug.Print "Ja, im FeldCode"
            bInFeld = True
        End If
    Else
        'Sonst wird der Feldresultatbereich mit dem Markierungsbereich verglichen.
        If rngMarkierung.InRange(fld.Result) Then
            Debug.Print "Ja, im Feldresultat"
            bInFeld = True
        End If
    End If
    'Liegt die Markierung am Feldanfang, wird der Startpunkt des Feldcodebereichs
```

Listing 6.76 Ermitteln, ob sich die Markierung in einer Feldfunktion befindet (Fortsetzung)

```

' mit dem Endpunkt des Markierungsbereichs verglichen.
If Not bInFeld And (fld.Code.Start = rngMarkierung.End) Then
    Debug.Print "Ja, am Feldanfang"
    bInFeld = True
End If
If bInFeld = False Then Debug.Print "Nicht in einem Feld "
IstMarkierungImFeld = bInFeld
End Sub

```



Die Beispieldatei *Bsp06\_01\_Field.doc* finden Sie auf der CD-ROM im Ordner `\Beispiele\Kap06`.

Es ist nicht möglich, eine Feldfunktion mit einer VBA-Funktion zu verbinden, um dynamisch das Ergebnis der VBA-Funktion als Feldergebnis anzuzeigen. Word-Entwickler haben sich diese Fähigkeit seit mehr als einem Jahrzehnt gewünscht; ob es sie angesichts der aktuellen Sicherheitsfragen jemals geben wird, ist fraglich.

Die einzige Feldfunktion, die eine Verbindung zu einem VBA-Projekt herstellen kann, ist die *Macrobutton*-Feldfunktion. Sie hat die Syntax `{ MACROBUTTON Makroname Anzeigetext }`. Makroname darf ein beliebiges Wort sein, muss aber auf eine Prozedur in einem zugänglichen VB-Projekt weisen, wenn Code ausgeführt werden soll. Anzeigetext stellt eine Eingabeaufforderung, ein Symbol und/oder eine Grafik dar. Die Länge ist auf eine Textzeile begrenzt (Anzeigetext als Feldresultat kann nicht über Zeilen umbrechen).

Die Prozedur *Makroname* wird durch einen Doppelklick auf die Feldfunktion ausgelöst. Programmtechnisch kann die Methode `DoClick` diese Benutzerhandlung nachahmen, wie das Listing 6.77 veranschaulicht.

Das doppelte Anklicken ist für den Benutzer immer schwieriger, als ein einfaches Anklicken, und der heutige Benutzer ist es eher gewohnt, mit einem einzigen Klick eine Handlung auszulösen. Word bietet die Anwendungsoption `Application.Options.ButtonFieldClicks` an. Wird sie auf »1« festgelegt, führt ein einfacher Klick auf ein *Macrobutton*-Feld das Makro aus. Um zu einem Doppelklick zurückzukehren, muss die Option auf »1« (Zero) festgelegt werden. Bitte beachten Sie, dass sich diese Option auf alle Dokumentfenster der laufenden Word-Sitzung auswirkt.

Listing 6.77 Mit der Methode *DoClick* wird das in einer *Macrobutton*-Feldfunktion angegebene Makro ausgeführt

```

Sub ButtonTestAusführen()
    Dim fld As Word.Field
    Dim doc As Word.Document
    Dim rng As Word.Range

    Set doc = ActiveDocument
    Set rng = Selection.Range
    For Each fld In doc.Fields
        If InStr(fld.Code, "ButtonTest") <> 0 Then
            fld.Result.Select
            fld.DoClick
        End If
    Next
    rng.Select
End Sub

```

**Listing 6.77** Mit der Methode *DoClick* wird das in einer *Macrobutton*-Feldfunktion angegebene Makro ausgeführt (*Fortsetzung*)

```
End Sub

Sub ButtonTest()
    Dim fld As Word.Field
    Dim strFeldCode As String

    Set fld = Selection.Fields(1)
    strFeldCode = fld.Code.Text
    If InStr(strFeldCode, "Hier anklicken") <> 0 Then
        strFeldCode = "MacroButton ButtonTest Hier wurde angeklickt"
    Else
        strFeldCode = "MacroButton ButtonTest Hier anklicken"
    End If
    fld.Code.Text = strFeldCode
End Sub
```



Die Beispieldatei *Bsp06\_01\_Field.doc* finden Sie auf der CD-ROM im Ordner *\Beispiele\Kap06*.

## Verknüpfungen, Tabellen und Berechnungen

Wie mehrmals schon erwähnt, verwalten Feldfunktionen Verknüpfungen zu Text innerhalb des Dokuments sowie in anderen Dateien, seien es andere Word-Dokumente, Textdateien, Grafiken oder sogar Excel-Tabellen und Excel-Diagramme oder Datenbanktabellen. An dieser Stelle stellen wir einige Aspekte der Herstellung von Verknüpfungen und ihrer Verwaltung vor.

**WICHTIG** Viele Verknüpfungsarten unterstützen von sich aus eine dynamische Aktualisierung. Neuere Sicherheitsmaßnahmen können diese Funktionalität unterbinden. Mehr hierzu steht im Knowledge Base-Artikel »Installation des Word-Updates ändert das Verhalten von Word-Feldern« unter <http://support.microsoft.com?kbid=330079>.

In der Benutzerschnittstelle werden Verknüpfungen über verschiedene Menübefehle des Menüs *Einfügen*, über Symbolleisten-Schaltflächen oder über die Zwischenablage hergestellt. Diese haben durchaus ihre Gegenstücke im Word-Objektmodell, der Umgang damit ist jedoch teilweise umständlich oder gar unzuverlässig. In vielen Fällen ist es schneller, die Feldfunktion in den Text direkt einzufügen, anstatt sich mit einer »Insert« oder gar der Paste-Methode abzufragen.

**HINWEIS** Eine Diskussion zur Feldfunktion *IncludePicture* finden Sie im Abschnitt »Grafiken: Die *InlineShape*- und *Shape*-Objekte« in diesem Kapitel.

Um die benötigte Syntax zu ermitteln, fügen Sie die Datei, Grafik oder Tabelle über die Benutzerschnittstelle mit Verknüpfung ein. Drücken Sie **[Alt] + [F9]**, um den Feldcode zu sehen – dies ergibt die Grundinformationen für das Text-Argument, die den Bedürfnissen der Anwendung angepasst werden.

Nehmen wir als Beispiel eine umfangreiche Excel-Tabelle, die sich über mehrere Seiten erstreckt. In Word ist, im Gegensatz zu Excel, die nützliche Größe eines grafischen Objekts auf maximal die Größe einer Seite begrenzt. Ist es größer, wird es visuell einfach abgeschnitten. Alle in Word zur Verfügung stehenden Menübefehle für die Einfügung einer Excel-Tabelle fügen ein grafisches Objekt ein; nur über die Zwischenablage kann eine Excel-Tabelle in eine Word-Tabelle umgewandelt werden, so dass diese über mehrere Seiten umbrochen wird (siehe Abbildung 6.43).

**HINWEIS**

Eingebettete Excel-Tabellenobjekte werden in Kapitel 11 vorgestellt.

Bei der Automatisierung soll jedoch wegen der Gefahr eines Konflikts mit dem Benutzer möglichst *nicht* mit der Zwischenablage gearbeitet werden. Es bleibt der Weg über die *Link*-Feldfunktion, die unter der Tabelle in Abbildung 6.43 sichtbar ist. Diese besteht aus dem Feldnamen, der OLE-Klasse der Anwendung, woraus das Objekt stammt (*Excel.Sheet.8*, was für alle Versionen von 97 bis einschließlich 2003 gilt), der Dateipfadangabe, dem Datenbereich sowie mehreren Schaltern.

Damit steht die Grundsyntax für die Add-Methode bereit und kann nach Bedarf angepasst werden. Beispielsweise darf die Zeilen/Spalten-Bereichangabe durch einen Bereichsnamen ersetzt werden. Somit könnte das Text-Argument lauten:

```
"LINK Excel.Sheet.8 " & Chr$(34) & " " & Chr$(34) &
"C:\\WordBuch\\Beispiele\\Kap06\\Bsp06_02_Field.xls" & Chr$(34) & _
"Tabelle1!wdFieldType" & Chr$(34) & " \a \f 5 \h \* MERGEFORMAT" _
```

**HINWEIS**

*Link*-Feldfunktionen unterstützen keine relativen Pfadangaben.

Abbildg. 6.43

Eine verknüpfte Excel-Tabelle, die als eine Word-Tabelle statt OLE-Objekt ins Dokument eingebunden wurde. Somit kann der Umbruch über mehrere Seiten erfolgen.

WdFieldType	Deutscher-Ausdruck	Englischer-Ausdr.	Englischer-Ausdr.	Deutscher-Ausdruck
wdFieldAddressBlock	(neu in Word 2002)	AddressBlock	=Formula	=Ausdruck
wdFieldGreetingLine	(neu in Word 2002)	GreetingLine	AddressBlock	(neu in Word 2002)
wdFieldEditTime	(neu in Word 2002)	EditTime	Advanced	Versetzten
wdFieldExpression	=Ausdruck	=Formula	Ask	Frage
wdFieldSection	Abchnitt	Section	Author	Autoren
wdFieldDate	AktualDate	Date	AutoNum	AutoNr
wdFieldQuote	Angeben	Quote	AutoNumLg	AutoNrDez
wdFieldNumPages	AnzSeiten	NumPages	AutoNumOut	AutoNrGlo
wdFieldNumWords	AnzWörter	NumWords	AutoText	AutoText
wdFieldNumChars	AnzZeichen	NumChars	AutoTextList	AutoTextListe
wdFieldAutoNum	AutoNr	AutoNum	BarCodes	(nur gültig für US-Version)

WdFieldType	Deutscher-Ausdruck	Englischer-Ausdr.	Englischer-Ausdr.	Deutscher-Ausdruck
wdFieldAutoNumLg	AutoNrDez	AutoNumLg	Comments	Kommentar
wdFieldAutoNumOutline	AutoNrGlo	AutoNumOut	Compare	Vergleich
wdFieldAuthor	Autoren	Author	CreateDate	Erstelldate
wdFieldAutoText	AutoText	AutoText	Databases	Datenbanken
wdFieldAutoTextList	AutoTextListe	AutoTextList	Date	AktualDate
wdFieldUserAddress	BenutzerAdress	UserAddress	DocProperty	DokEigenschaft

```
"LINK Excel.Sheet.8 " & Chr$(34) & " " & Chr$(34) &
"C:\\WordBuch\\Beispiele\\Kap06\\Bsp06_02_Field.xls" & Chr$(34) & _
"Tabelle1!wdFieldType" & Chr$(34) & " \a \f 5 \h \* MERGEFORMAT" _
```

**TIPP**

Wollen Sie etwas ohne dynamische Verknüpfung ins Dokument einfügen, gehen Sie auf die gleiche Weise vor, lösen allerdings die Verknüpfung anschließend mit der `Unlink`-Methode auf:

```
Set fld = doc.Fields.Add(Range:=Selection.Range,
    Text:="IncludePicture C:\\Windows\\Feder.bmp", PreserveFormatting:=True)
fld.Unlink
```

Wenn wir schon beim Thema »Excel« sind, wenden sich die Gedanken den Berechnungen und Kalkulationen zu. Ohne Frage ist dafür Excel das geeignetere Werkzeug. Excel steht jedoch nicht immer zur Verfügung und eignet sich wegen der maximalen Größe von einer Seite nicht für alle Fälle.

Berechnungen können, obwohl etwas umständlicher, auch in Word vorgenommen werden. Die Funktionalität wird über Feldfunktionen bereitgestellt, genauer über die *Ausdruck*-Feldfunktion: { = }. Nähere Angaben zu den Möglichkeiten dieser Feldfunktion finden Sie in der Hilfe, wenn Sie nach dem Begriff »= (Formula)-Feld« (ohne Anführungszeichen) suchen. Wir befassen uns hier mit zwei Aspekten.

Auf  
Tabellen-  
inhalt  
verweisen

Das Hilfethema erklärt, wie Tabellenbezüge zu formulieren sind. Darunter befindet sich ein Abschnitt »Bezüge zu Zellen in einer anderen Tabelle«, worin steht: »Um Bezüge zu Zellen in einer anderen Tabelle herzustellen oder um von außerhalb der Tabelle einen Bezug zu einer Zelle herzustellen, kennzeichnen Sie die Tabelle mit einer Textmarke. Über das Feld { =average(Tabelle2 b:b) } wird für die Spalte B in der mit der Textmarke *Tabelle2* gekennzeichneten Tabelle der Mittelwert errechnet.«

Das stimmt soweit. Was nicht präzisiert wird, ist das notwendige Heranziehen einer Funktion, wie *average*, um die Tabelle innerhalb der Textmarke anzusprechen. Diese Formel würde eine Syntax-Fehlermeldung verursachen: { = (Tabelle2 C1) }. Um den Wert einer einzelnen Tabellenzelle abzufragen, kann die Funktion *Sum* verwendet werden: { = Sum(Tabelle2 C1) }.

Ver-  
schach-  
telte  
Feldfunk-  
tionen

Nicht alle Aufgaben, die mit Feldfunktionen zu lösen sind, lassen sich mit einer einzigen Feldfunktion lösen. Oft müssen Feldfunktionen ineinander verschachtelt werden. In der Benutzerschnittstelle wird dies mit eingeblendeten Feldcodes getan. Programmtechnisch gestaltet sich die Aufgabe als äußerst komplex, weil das Verschachteln von Feldfunktionen im Objektmodell nicht ausdrücklich vorgesehen ist.

Der Vorgang kann mit dem Makrorekorder aufgezeichnet und ohne Anpassung verwendet werden, es gibt dabei aber zwei Probleme:

- Die Eingabe der Feldfunktionen muss genau sitzen, Fehler sind nicht erlaubt.
- Das Ein- und Ausblenden von Feldfunktionen in großen Dokumenten verzögert sich, weil das Seitenlayout angepasst werden muss. Zudem könnte die markierte Stelle im Text versetzt werden.

Wie fast immer ist die Arbeit mit dem Range-Objekt vorzuziehen.

Die Abbildung 6.44 stellt das Beispiel für diese Diskussion vor. Word stellt einen Formatierungsschalter zur Verfügung, der Zahlen in Text umwandelt, aber leider nur bis zum Wert 999.999,99. Zahlen, die größer sind als eine Million, verursachen eine Fehlermeldung als Feldfunktionsergebnis. In der heutigen Zeit sind Millionenbeträge alltäglich geworden; eine Anpassung der Funktionalität drängt sich auf.

Oben in der Abbildung steht die Zahl, die ganz unten in Text umwandelt wurde. Die Feldfunktion, die dahinter steckt, befindet sich dazwischen, als einfacher Text wiedergegeben. Die Feldfunktionen sind bis auf fünf Ebenen verschachtelt.

**Abbildg. 6.44** Mit einer komplexen, verschachtelten Feldfunktion können Zahlen in Millionenhöhe in Text umwandelt werden

10356237,95

```
{ QUOTE { SET n { REF ZahlInText } } { Set m { = int({ n }/1000000) } { Set r { = MOD({ n };1000000) } } { IF { n } < 1000000 "{ n}* cardtext }" " { Quote "{ If { m } = 1 "einemillion" "{ m}* cardtext }millionen" } { If { r } = 0 "" "{ r}* cardtext } } "\* lower \* CharFormat }" }
```

zehnmillionendriehundertsechsfünfzigtausendzweihundertachtunddreißig

Die Prozedur *CardTextFunktionAufbauen* ruft die Funktion *FeldCodeEinfuegen* in Listing 6.78 auf und übergibt ihr die in eine Feldfunktion umzuwandelnde Zeichenkette sowie den Zielbereich. Diese arbeitet sich Zeichen für Zeichen durch die angegebene Zeichenkette und speichert die Zeichen in einem »Buffer«. Wenn eine öffnende geschweifte Klammer vorliegt, wird der Text aus dem Buffer eingefügt, gefolgt von einem Paar Feldklammern. Die darin befindlichen Leerzeichen werden gelöscht, der Zielbereich dazwischen gesetzt und die Zeichenkette weiter in der nun leeren Buffer gelesen. Liegt eine schließende Klammer vor, wird der Text im Buffer innerhalb den Feldklammern eingefügt, und in den Zielbereich nach der schließenden Klammer gesetzt. Es geht auf ähnliche Weise weiter, bis die Zeichenkette abgearbeitet wurde. Am Schluss gibt die Funktion den Feldfunktion-enthaltenden Bereich an die rufende Prozedur *CardTextFunktionAufbauen* zurück, wo die Feldfunktion im Bereich aktualisiert wird.

**ACHTUNG** Bitte beachten Sie, dass Funktion nicht kontrolliert, ob die Feldfunktion gültig ist oder ob der Zielbereich eine Feldfunktion akzeptieren kann.

**Listing 6.78** Eine Zeichenkette in eine Feldfunktion umwandeln

```
Function FeldCodeEinfuegen(ByVal strFeldCode As String, _
    ByRef rngZielBereich As Word.Range) As Word.Range

    ' Zweck:
    ' Wandelt Text in Feldcodes um, wobei:
    ' "{" bedeutet eine öffnende Feldklammer
    ' "}" bedeutet eine schließende Feldklammer
    ' "{-" bedeutet eine normale, öffnende, geschweifte Klammer
    ' "-}" bedeutet eine normale, schließende, geschweifte Klammer
    ' "-{" bedeutet ein - Zeichen
    ' "-}" gefolgt von anderen Zeichen wird verworfen

    ' Parameter:
    ' strFeldCode: die Zeichenkette, die den Feldcode definiert
    ' ZielBereich: wo die Feldfunktion einzufügen ist

    Const EscapeChar = "-"
```

**Listing 6.78** Eine Zeichenkette in eine Feldfunktion umwandeln (Fortsetzung)

```

Dim lIndex As Long
Dim bEscapeCharFound As Boolean
Dim sBuffer As String
Dim sChar As String
Dim rngStart As Word.Range
bEscapeCharFound = False
sBuffer = ""
Set rngStart = rngZielBereich.Duplicate

rngZielBereich.TextRetrievalMode.IncludeFieldCodes = True
For lIndex = 1 To Len(strFeldCode)
    sChar = Mid(strFeldCode, lIndex, 1)
    Select Case sChar
        Case EscapeChar:
            If bEscapeCharFound Then
                sBuffer = sBuffer + sChar
                bEscapeCharFound = False
            Else
                bEscapeCharFound = True
            End If
        Case "{"
            If bEscapeCharFound Then
                sBuffer = sBuffer + sChar
                bEscapeCharFound = False
            Else
                ' Wir sind am Anfang einer Feldfunktion angelangt.
                rngZielBereich.Text = sBuffer
                sBuffer = ""
                rngZielBereich.Collapse direction:=wdCollapseEnd
                rngZielBereich.Fields.Add Range:=rngZielBereich, Type:=wdFieldEmpty, _
                    Text:="", PreserveFormatting:=False
                rngZielBereich.SetRange Start:=rngZielBereich.Start + 1, _
                    End:=rngZielBereich.Start + 1
                rngZielBereich.Delete unit:=wdCharacter, Count:=2
            End If
        Case "}"
            If bEscapeCharFound Then
                sBuffer = sBuffer + sChar
                bEscapeCharFound = False
            Else
                ' Wir sind am Ende einer Feldfunktion angelangt.
                rngZielBereich.InsertAfter Text:=sBuffer
                rngZielBereich.Select
                sBuffer = ""
                rngZielBereich.Collapse direction:=wdCollapseEnd
                rngZielBereich.SetRange Start:=rngZielBereich.End + 1, _
                    End:=rngZielBereich.End + 1
            End If
        Case Else
            sBuffer = sBuffer + sChar
    End Select
Next
rngZielBereich.InsertAfter Text:=sBuffer
rngZielBereich.Start = rngStart.Start
Set FeldCodeEinfuegen = rngZielBereich
End Function

```



Listing 6.78 Eine Zeichenkette in eine Feldfunktion umwandeln (Fortsetzung)

```

Sub CardTextFunktionAufbauen()
    Dim strTeststrFeldCode As String
    Dim rngTarget As Word.Range

    Set rngTarget = Selection.Range
    strTeststrFeldCode = rngTarget.Text
    rngTarget.InsertAfter vbCr
    rngTarget.Collapse wdCollapseEnd
    Set rngTarget = FeldCodeEinfuegen(strTeststrFeldCode, rngTarget)
    rngTarget.Fields.Update
End Sub

```

Feldfunktion in Text umwandeln

Das Gegenstück zur Funktion, die Text in eine Feldfunktion umwandelt, ist eine Prozedur, die eine Feldfunktion in Text konvertiert, wie in Listing 6.79. Somit bleibt Ihnen erspart, eine komplexe Feldfunktion wie diejenige in Abbildung 6.44 manuell eintippen zu müssen, um sie im Code weiterverwenden zu können.

**HINWEIS**

Ein Verweis auf die UserForm-Bibliothek *Microsoft Forms 2.0 Object Library* ist notwendig, um über das *DataObject* das Resultat in die Zwischenablage zu kopieren. Falls auf Ihrem System die Bibliothek nicht in der Liste unter *Extras/Verweise* vorhanden ist, klicken Sie auf *Durchsuchen*, und im Ordner *\Windows\system32* wählen Sie die Datei *FM20.DLL* an.

Listing 6.79 Diese Prozedur wandelt die markierte Feldfunktion in einfachen Text um

```

Sub FeldCodeInText()
    Dim rng As Word.Range
    Dim strFeldCode As String
    Dim strNeu As String
    Dim i As Long
    Dim CurrChar As String
    Dim CurrSetting As Boolean
    Dim oData As MSForms.DataObject

    'Die Vorbereitungen treffen.
    Set rng = Selection.Range
    rng.TextRetrievalMode.IncludeFieldCodes = True
    strNeu = ""
    Application.ScreenUpdating = False
    strFeldCode = rng.Text

    'Zeichen für Zeichen durch den Text arbeiten und das Resultat aufbauen.
    'Dabei öffnende oder schließende Klammer durch geschweifte ersetzen.
    For i = 1 To Len(strFeldCode)
        CurrChar = Mid(strFeldCode, i, 1)
        Select Case CurrChar
            Case Chr(19)
                CurrChar = "{"
            Case Chr(21)
                CurrChar = "}"
            Case Else
            End Select
        strNeu = strNeu + CurrChar
    Next i

```

**Listing 6.79** Diese Prozedur wandelt die markierte Feldfunktion in einfachen Text um (*Fortsetzung*)

```
'Das Resultat in die Zwischenablage übernehmen, so dass der Benutzer
'es einfügen kann, wo er will.
Set oData = New DataObject
oData.SetText strNeu
oData.PutInClipboard
End Sub
```



Die Beispieldatei *Bsp06\_02\_Field.doc* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap06*.

## Grafiken: Die *InlineShape*- und *Shape*-Objekte

Da Word ein Textverarbeitungsprogramm ist, bekundet es manchmal etwas Mühe mit Grafiken. Vor allem ist die VBA-Schnittstelle dafür weder vollständig noch besonders intuitiv. Ziel dieses Abschnitts ist, den allgemeinen Umgang mit Grafiken zu erklären, sowie einige Besonderheiten hervorzuheben.

Inline-  
Shape vs.  
Shape

Es gibt zwei verschiedene Methoden, eine Grafik in ein Dokument einzubetten: in der Zeile mit dem Text oder mit einer Textflussformatierung. Steht eine Grafik in der Zeile mit dem Text, ist sie Teil der *InlineShapes*-Auflistung, Word behandelt sie grundsätzlich wie ein Textzeichen, und die VBA-Befehle gehören dem Word-Objektmodell an. Wurde sie jedoch mit Textfluss formatiert, steht sie in der Zeichnungsebene des Dokuments, ist Teil der *Shapes*-Auflistung, und die passenden Befehle befinden sich im Office-Objektmodell, da Zeichnungsobjekte Office-übergreifend sind.



Jede Word-Version fügt Grafiken standardmäßig anders ein. In Word 2002 und 2003 darf der Benutzer sogar die Methode selber bestimmen (Menübefehl *Extras/Optionen*, Registerkarte *Bearbeiten*, Option *Bild einfügen als*). Folglich muss der Code, wollen Sie alle Grafiken eines Dokuments bearbeiten, beide Auflistungen berücksichtigen.

## Grafiken einfügen

Einsichten in das Objektmodell für beide Auflistungen sind durch den Makrorekorder möglich. Es gibt aber einige Tricks, um eine Grafik nach deren Einfügung zu markieren, um sie weiter zu manipulieren. In Listing 6.80 befinden sich zwei in Word 2003 aufgezeichnete Prozeduren für das Einfügen und die Größenänderung von Grafiken. Bei der ersten wurde die Grafik in die Zeile mit dem Text eingefügt, bei der zweiten mit Textformatierung.

Aus Datei  
mit  
AddPic-  
ture

Beim Vergleich der zwei Routinen fällt auf, dass sowohl die *InlineShapes*- wie die *Shapes*-Auflistungen eine *AddPicture*-Methode besitzen, um eine Grafikdatei einzufügen. Die beiden haben aber zum Teil unterschiedliche Argumente (zweite Codezeile).

In der nächsten Codezeile sehen Sie, wie ein *InlineShape* bzw. ein *Shape* markiert wird. Im ersten Fall steht nach der Einfügung die Einfügemarke links neben der Grafik. Halten Sie also die -Taste fest und drücken Sie dann . Wurde ein *Shape* eingefügt, müssen Sie es mit der Maus anklicken.

Letztlich machen wir darauf aufmerksam, dass der Makrorekorder nicht nur die geänderten Optionen im Dialogfeld *Grafik formatieren* aufzeichnet, sondern mehrere Optionen aus allen Registerkarten. Sie müssen also den Code anpassen, so dass er nur die gewünschte Änderung vornimmt, wenn er später ausgeführt wird. Auffallend ist, dass gerade die Größenänderung für das Shape nicht aufgezeichnet wurde. Die Codezeilen

```
Selection.InlineShapes(1).LockAspectRatio = msoTrue
Selection.InlineShapes(1).Height = 113.4
Selection.InlineShapes(1).Width = 113.4
```

sind für die Größenänderung des InlineShapes zuständig.

Listing 6.80

Dieser aufgezeichnete Code gibt Aufschluss über die *InlineShape*- bzw. *Shape*-Objekte

```
Sub InlineShapeImportierenUndBearbeiten()

    Selection.InlineShapes.AddPicture FileName:="C:\WINDOWS\Seifenblase.bmp", _
        LinkToFile:=False, SaveWithDocument:=True
    Selection.MoveLeft Unit:=wdCharacter, Count:=1, Extend:=wdExtend
    Selection.InlineShapes(1).Fill.Visible = msoFalse
    Selection.InlineShapes(1).Fill.Solid
    Selection.InlineShapes(1).Fill.Transparency = 0#
    Selection.InlineShapes(1).Line.Weight = 0.75
    Selection.InlineShapes(1).Line.Transparency = 0#
    Selection.InlineShapes(1).Line.Visible = msoFalse
    Selection.InlineShapes(1).LockAspectRatio = msoTrue
    Selection.InlineShapes(1).Height = 113.4
    Selection.InlineShapes(1).Width = 113.4
    Selection.InlineShapes(1).PictureFormat.Brightness = 0.5
    Selection.InlineShapes(1).PictureFormat.Contrast = 0.5
    Selection.InlineShapes(1).PictureFormat.ColorType = msoPictureAutomatic
    Selection.InlineShapes(1).PictureFormat.CropLeft = 0#
    Selection.InlineShapes(1).PictureFormat.CropRight = 0#
    Selection.InlineShapes(1).PictureFormat.CropTop = 0#
    Selection.InlineShapes(1).PictureFormat.CropBottom = 0#
End Sub

Sub ShapeImportierenUndBearbeiten()
    ActiveDocument.Shapes.AddPicture(Anchor:=Selection.Range, FileName:=
        "C:\WINDOWS\Zapotek.bmp", LinkToFile:=False, SaveWithDocument:=True). _
        WrapFormat.Type = wdWrapSquare
    ActiveDocument.Shapes("Picture 2").Select
    Selection.ShapeRange.Fill.Visible = msoFalse
    Selection.ShapeRange.Fill.Solid
    Selection.ShapeRange.Fill.Transparency = 0#
    Selection.ShapeRange.Line.Weight = 0.75
    Selection.ShapeRange.Line.DashStyle = msoLineSolid
    Selection.ShapeRange.Line.Style = msoLineSingle
    Selection.ShapeRange.Line.Transparency = 0#
    Selection.ShapeRange.Line.Visible = msoFalse
    Selection.ShapeRange.LockAspectRatio = msoTrue
    Selection.ShapeRange.Rotation = 0#
    Selection.ShapeRange.PictureFormat.Brightness = 0.5
    Selection.ShapeRange.PictureFormat.Contrast = 0.5
    Selection.ShapeRange.PictureFormat.ColorType = msoPictureAutomatic
```

**Listing 6.80** Dieser aufgezeichnete Code gibt Aufschluss über die *InlineShape*- bzw. *Shape*-Objekte (Fortsetzung)

```
Selection.ShapeRange.PictureFormat.CropLeft = 0#
Selection.ShapeRange.PictureFormat.CropRight = 0#
Selection.ShapeRange.PictureFormat.CropTop = 0#
Selection.ShapeRange.PictureFormat.CropBottom = 0#
Selection.ShapeRange.Left = 70.85
Selection.ShapeRange.Top = 198.1
Selection.ShapeRange.RelativeHorizontalPosition = _
wdRelativeHorizontalPositionColumn
Selection.ShapeRange.RelativeVerticalPosition = _
wdRelativeVerticalPositionParagraph
Selection.ShapeRange.Left = CentimetersToPoints(0)
Selection.ShapeRange.Top = CentimetersToPoints(0)
Selection.ShapeRange.LockAnchor = False
Selection.ShapeRange.LayoutInCell = True
Selection.ShapeRange.WrapFormat.AllowOverlap = True
Selection.ShapeRange.WrapFormat.Side = wdWrapBoth
Selection.ShapeRange.WrapFormat.DistanceTop = CentimetersToPoints(0)
Selection.ShapeRange.WrapFormat.DistanceBottom = CentimetersToPoints(0)
Selection.ShapeRange.WrapFormat.DistanceLeft = CentimetersToPoints(0.32)
Selection.ShapeRange.WrapFormat.DistanceRight = CentimetersToPoints(0.32)
Selection.ShapeRange.WrapFormat.Type = wdWrapSquare
End Sub
```

Wie im Abschnitt »Die gegenwärtige Markierung: *Selection* und ähnliche Objekte« in diesem Kapitel erwähnt, soll Code mit dem *Selection*-Objekt möglichst vermieden werden. Listing 6.81 bzw. Listing 6.82 zeigt den bereinigten Code. Da die *AddPicture*-Methode immer ein Objekt zurückgibt, ist es einfach, eine entsprechende Objekt-Variable zu deklarieren und ihr das Grafikobjekt gleich beim Importieren zuzuweisen.

Beachten Sie, wie die Positionierung des Shapes nochmals nach der Anpassung der Größe durchgeführt wird. In diesem Fall wird die Grafik relativ zum verankernden Absatz positioniert. (Entspricht der Einstellung *Objekt mit Text verschieben* der Registerkarte *Bildposition*, die über die Schaltfläche *Weitere* der Registerkarte *Layout* im Dialogfeld *Grafik formatieren* erreicht wird (Abbildung 6.46).)

**Listing 6.81** Der bearbeitete Code führt nur die gewünschten Handlungen aus: Grafik einfügen und die Größe anpassen

```
Sub InlineShapeImportierenUndBearbeiten2()
    Dim ils As Word.InlineShape

    Set ils = Selection.InlineShapes.AddPicture(FileName:="C:\WINDOWS\Seifenblase.bmp", _
        LinkToFile:=False, SaveWithDocument:=True)
    ils.LockAspectRatio = msoTrue
    ils.Height = 113.4
    ils.Width = 113.4
End Sub

Sub ShapeImportierenUndBearbeiten2()
    Dim shp As Word.Shape

    Set shp = ActiveDocument.Shapes.AddPicture(Anchor:=Selection.Range, FileName:= _
        "C:\WINDOWS\Zapotek.bmp", LinkToFile:=False, SaveWithDocument:=True)
    shp.WrapFormat.Type = wdWrapSquare
    shp.LockAspectRatio = msoTrue
```

**Listing 6.81** Der bearbeitete Code führt nur die gewünschten Handlungen aus: Grafik einfügen und die Größe anpassen (Fortsetzung)

```
shp.Height = 113.4
shp.RelativeHorizontalPosition = _
    wdRelativeHorizontalPositionColumn
shp.RelativeVerticalPosition = _
    wdRelativeVerticalPositionParagraph
shp.Left = CentimetersToPoints(0)
shp.Top = CentimetersToPoints(0)
End Sub
```

**Listing 6.82** (.NET): Die C#-Version

```
private void InlineShapeImportierenUndBearbeiten_CS()
{
    wd.Selection sel = wdApp.Selection;
    object objRange = (object) sel.Range;
    wd.InlineShape ils = sel.InlineShapes.AddPicture(@"C:\WINDOWS\Seifenblase.bmp",
        ref objFalse, ref objTrue, ref objRange);
    ils.LockAspectRatio = Microsoft.Office.Core.MsoTriState.msoTrue;
    ils.Height = 113.4f;
    ils.Width = 113.4f;
}

private void ShapeImportierenUndBearbeiten_CS(wd.Document doc)
{
    wd.Selection sel = wdApp.Selection;
    object objRange = (object) sel.Range;
    wd.Shape shp = doc.Shapes.AddPicture(@"C:\WINDOWS\Zapotek.bmp", ref objFalse,
        ref objTrue, ref objMissing, ref objMissing, ref objMissing, ref objMissing,
        ref objRange);
    shp.WrapFormat.Type = wd.WdWrapType.wdWrapSquare;
    shp.LockAspectRatio = Microsoft.Office.Core.MsoTriState.msoTrue;
    shp.Height = 113.4f;
    shp.RelativeHorizontalPosition =
        wd.WdRelativeHorizontalPosition.wdRelativeHorizontalPositionColumn;
    shp.RelativeVerticalPosition =
        wd.WdRelativeVerticalPosition.wdRelativeVerticalPositionParagraph;
    shp.Left = wdApp.CentimetersToPoints(0);
    shp.Top = wdApp.CentimetersToPoints(0);
}
```



Die Beispieldatei *Bsp06\_01\_Graf.doc* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap06*.

Über die  
Zwischen-  
ablage

Grafiken werden nicht nur aus Dateien importiert, sie werden auch aus der Zwischenablage eingefügt. Wie beim aufgezeichneten Code besteht auch hier das Problem: Wie wird das eingefügte Objekt angesprochen, so dass mit der Aufzeichnung weitergefahren werden kann? Die Methoden Paste und PasteSpecial geben, im Gegensatz zu AddPicture, kein Objekt zurück.

Prozeduren für diese Aufgabe finden Sie in Listing 6.83. Im Fall eines InlineShape-Objekts steht nach dem Einfügen aus der Zwischenablage die Einfügemarke rechts neben der Grafik. Dies funktioniert also ähnlich wie bei der Aufzeichnung beim Einfügen aus einer Datei: Die Markierung wird mit der Tastenfolge + um ein Zeichen – diesmal nach links – erweitert.

Nach Ausführung der Methode Paste ist ein Shape markiert, also kein Problem. Wollen Sie aber PasteSpecial (Menübefehl *Bearbeiten/Inhalte einfügen*) einsetzen, um eine Verknüpfung herzustellen oder den Datentyp festzulegen, braucht es ein »Workaround«, da die Markierung im Text bleibt. Der Code muss das eingefügte Objekt irgendwie erkennen können. Die Frage ist nur, wie.

Im Abschnitt »Die Positionierung von Shape-Objekten« in diesem Kapitel wird beschrieben, wie jedes grafische Objekt mit einem bestimmten Absatz verbunden (verankert) sein muss. Folglich müsste das Paragraph-Objekt der aktuellen Markierung oder des Zielbereichs (Range) das eingefügte Objekt liefern. Problematisch wird es jedoch, wenn mehrere Objekte im gleichen Absatz verankert sind.

Im Beispielcode der Prozedur *InhalteEingefuegtesShapeErfassen* wurde das Problem mit Hilfe der AlternativeText-Eigenschaft eines Shape-Objekts gelöst. Standardmäßig wird die Eigenschaft neu eingefügter Grafiken nicht belegt. Bevor die Grafik eingefügt wird, wird diese Eigenschaft für alle im Absatz verankerten Objekte mit einem eindeutigen Wert belegt, wenn sie noch keine Angabe enthält. Nach dem Einfügen wird nochmals durch den ShapeRange geschleift, um das einzige Objekt ohne einen AlternativeText-Eintrag zu finden – dies ist die eingefügte Grafik – und es einer Objekt-Variablen zugewiesen. Danach werden die vom Code generierten AlternativeText-Angaben entfernt.

**HINWEIS**

Ein Fehler wird ausgelöst, falls sich bei der Ausführung der Prozedur *InhalteEingefuegtesShapeErfassen* keine Grafik, sondern ein anderer Dateityp in der Zwischenablage befindet. Die Prozedur fängt den Fehler auf und blendet eine entsprechende Meldung ein. Das Office-Objektmodell bietet keine Schnittstelle an, um den Datentyp der Zwischenablage zu ermitteln.

**Listing 6.83** Prozeduren, um das über die Zwischenablage eingefügte Objekt im Code zu erfassen

```
Sub EingefuegtesInlineShapeErfassen()
    Dim rng As Word.Range
    Dim ils As Word.InlineShape

    Set rng = Selection.Range
    'rng.Paste
    rng.PasteSpecial Placement:=wdInLine, DataType:=wdPasteMetafilePicture
    rng.MoveStart wdCharacter, -1
    Set ils = rng.InlineShapes(1)
End Sub

Sub EingefuegtesShapeErfassen ()
    Dim rng As Word.Range
    shp As Word.Shape

    Set rng = Selection.Range
    rng.PasteSpecial Placement:=wdFloatOverText, DataType:=wdPasteMetaPicture
    Set shp = rng.ShapeRange(1)
    Debug.Print shp.Name
End Sub

Sub InhalteEingefuegtesShapeErfassen()
    Dim rng As Word.Range
    Dim shpNeu As Word.Shape
    Dim shp As Word.Shape
    Dim counter As Long

    On Error GoTo Fehlerbehandlung
```

Listing 6.83 Prozeduren, um das über die Zwischenablage eingefügte Objekt im Code zu erfassen (Fortsetzung)

```

Set rng = Selection.Range
counter = 0
'Es ist nur möglich, einen ShapeRange durchzuschleifen,
'wenn mindestens ein Shape vorhanden ist, also testen
If rng.Paragraphs(1).Range.ShapeRange.Count > 0 Then
    'Sicherstellen, dass die Eigenschaft AlternativeText aller
    'im Absatz verankerten Shapes belegt ist
    'mit eindeutigen Inhalt (Datum + Zeit des Kontrollgangs)
    For Each shp In rng.Paragraphs(1).Range.ShapeRange
        If Len(shp.AlternativeText) = 0 Then
            shp.AlternativeText = "Autogenerated" & Format(Now, "yyymmdd_hhnnss_") & counter
            counter = counter + 1
        End If
    Next
End If
rng.PasteSpecial Placement:=wdFloatOverText, DataType:=wdPasteMetafilePicture
'Die neue Grafik finden
For Each shp In rng.Paragraphs(1).Range.ShapeRange
    If Len(shp.AlternativeText) = 0 Then
        Set shpNeu = shp
    End If
Next
'Da AlternativeText bei eingeblendeten nicht druckbaren Zeichen
'über die Grafik angezeigt wird, alle "Autogenerated" löschen
For Each shp In rng.Paragraphs(1).Range.ShapeRange
    If Left(shp.AlternativeText, 13) = "Autogenerated" Then
        shp.AlternativeText = ""
    End If
Next
Debug.Print shpNeu.Name

Fertigstellen:
Exit Sub

Fehlerbehandlung:
Select Case Err.Number
    Case 5342 'Unbekannter Datentyp beim Einfügen
        MsgBox "Keine Grafik in der Zwischenablage vorhanden", _
            vbOKOnly + vbCritical, "Kapitel 6 - Grafiken"
    Case Else
        MsgBox "Es gab den folgenden unerwarteten Fehler in der Prozedur " _
            & "InhalteEingefuegtesShapeErfassen:" & vbCr & vbCr & _
            Err.Number & vbCr & vbCr & Err.Description, vbCritical + vbOKOnly, _
            "Kapitel 6 - Grafiken"
End Select
End Sub

```

Dialog-  
feld  
Grafik  
einfügen

Oft müssen Anwendungen interaktiv mit dem Benutzer arbeiten. Wir wollen es ihm beispielsweise ermöglichen, eine Grafik auszuwählen, aber der Code soll diese einfügen und weiter bearbeiten. Am einfachsten geht es, wenn wir das Word-eigene Dialogfeld *Grafik einfügen* im Code verwenden können. Ein Beispiel hierfür finden Sie in Kapitel 24.

Es ist auch möglich, in Word 2002 und Word 2003, das `FileDialog`-Objekt einzusetzen, falls Sie über die Benutzerschnittstelle mehr Kontrolle brauchen (beispielsweise nur bestimmte Dateiendungen anbieten wollen). Das `FileDialog`-Objekt wird im Kapitel 14 vorgestellt.



Die Beispieldatei *Bsp06\_01\_Graf.doc* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap06*.

## Verknüpfungen erstellen und verwalten

Jede der diversen Methoden, um Grafiken einzufügen, erlaubt eine Verknüpfung mit der Ursprungsdatei. Wird eine Grafik mit der *PasteSpecial*-Methode eingefügt, bedient sich der Entwickler der *Link*-Eigenschaft, um die Verknüpfung herzustellen.

Wie aus Listing 6.81 hervorgeht, hat die *AddPicture*-Methode eine optionale *LinkToFile*-Eigenschaft, die für eine Dateiverknüpfung sorgt. Zudem hat die Methode die optionale Eigenschaft *SaveInDocument*. Diese wird nur berücksichtigt, wenn *LinkToFile* auf *True* gesetzt wird. Ist *SaveInDocument* ebenfalls *True*, wird die Grafik in der Dokumentstruktur gespeichert. Sonst enthält das Dokument nur die Verknüpfungsinformationen, was in einer kleineren Dateigröße resultiert. Standardmäßig werden beide auf *False* gesetzt.

Verknüpfungen eingebetteter Grafiken werden über die *LinkFormat*-Eigenschaft gesteuert. Die wichtigsten Eigenschaften und Methoden sind in Tabelle 6.16 aufgelistet.

**Tabelle 6.16** Eigenschaften für die Verwaltung von Verknüpfungen

LinkFormat-Eigenschaft oder -Methode	Datentyp	Beschreibung
<i>BreakLink</i>		Löst die Verknüpfung auf. (*)
<i>SavePictureWithDocument</i>	Boolean	Wenn <b>False</b> , wird nur die Verknüpfung zur Grafik im Dokument gespeichert. (*)
<i>SourceFullName</i> <i>SourceName</i> <i>SourcePath</i>	Zeichenkette	Geben die vollen Pfadangaben, den Dateinamen sowie den Dateipfad zurück. (*)
<i>Update</i>		Aktualisiert die Grafik. (*)
<i>Gilt nur für InlineShapes</i>		
<i>Locked</i>	Boolean	Wenn <b>True</b> , kann die Grafik nicht aktualisiert werden. (1)
(*) Fügt in Word 2002 und 2003 automatisch den Schalter <i>* Mergeformat</i> hinzu, auch wenn er bereits vorhanden ist, was zu Formatierungsverlusten führen kann.		

### SourceFullName-Eigenschaft in Word 97 und Word 2000

Zwar stellt das Word-Objektmodell die *LinkFormat.SourceFullName*-Eigenschaft für *InlineShape*- sowie *Shape*-Objekte zur Verfügung. Wird sie jedoch benutzt, um die Pfadangabe einer verknüpften Grafik zu ändern, stürzen frühere Word-Versionen (z.B. 97 und 2000) ab:

```
ActiveDocument.Shapes("MeinBild").LinkFormat.SourceFullName = "C:\Grafiken\MeinBild.bmp"
```



Das Problem wurde in Word 2002 behoben, was aber nicht hilft, wenn Sie frühere Versionen automatisieren müssen. Ein gutes »Workaround« für Shape-Objekte in diesen Versionen gibt es nicht. In diesem Fall muss ein Shape in ein InlineShape konvertiert werden, um auf den Feldcode zuzugreifen:

```
Set ils = ActiveDocument.Shapes("MeinBild").ConvertToInlineShape
```

Wir raten davon ab, eine in ein InlineShape konvertierte Grafik mit der ConvertToShape-Methode zurück in ein Shape zu wandeln. Es geht, aber eine nochmalige Konvertierung in ein InlineShape beschädigt den Feldcode, was einen Dokumentabsturz verursachen kann. Stattdessen sollen Sie den Textfluss um die Grafik anders hinbekommen, indem Sie die Grafik als ein InlineShape in einen Positionsrahmen oder in ein Textfeld einfügen (siehe den Abschnitt »Textfluss-Formatierung« in diesem Kapitel).

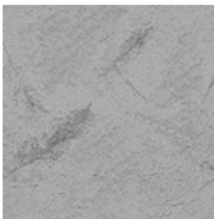
Im Hintergrund verwaltet Word verknüpfte Grafiken über die Feldfunktion *IncludePicture* (in Abbildung 6.45 abgebildet). Neben zum Feldnamen besteht der Feldcode aus der Pfadangabe zur Ursprungsdatei; der Pfad darf relativ zum Speicherort des Dokuments sein oder absolut. Zusätzlich stehen der Feldfunktion zwei wichtige Schalter zur Verfügung:

- \d: Weist Word an, dass die Grafikdaten nicht im Dokument gespeichert werden sollen (gleich `SavePictureInDocument = False.`)
- \\* *Mergeformat*: Behält bei der Aktualisierung in Word vorgenommene Formatierungen bei.

Durch Drücken der Tastenkombination `[Alt] + [F9]` werden alle Feldcodes im *Dokumenttext* ein- und wieder ausgeblendet. Sie werden also nur diejenige für InlineShape-Objekte sehen, da Shapes außerhalb des Textes in der Zeichnungsebene stehen.

Abbildg. 6.45 In Wirklichkeit werden in Word verknüpfte Grafiken durch *IncludePicture*-Feldcodes verwaltet

```
{ INCLUDEPICTURE "C:\\Windows\\Feder.bmp" \* MERGEFORMAT }
```



**WICHTIG** Die Backslashes in Pfadangaben einer Feldfunktion müssen immer verdoppelt werden. Dies weil ein einzelnes Backslash-Zeichen in einem Feldcode einen Schalter identifiziert.

Relative Pfadangaben werden genauso wie in DOS geschrieben. Befindet sich die verknüpfte Datei im gleichen Ordner mit dem Dokument, braucht es nur den Dateinamen. Um auf die nächst höhere Orderebene zu weisen, wird `..\\` vor den Pfadnamen gesetzt; mehrere `..\\` dürfen aufeinander folgen. Bitte beachten Sie, dass sich in diesem Fall Word auf das aktuelle Verzeichnis für *die Anwendung* bezieht (meistens dort, wo das letzte Dokument geöffnet wurde), und nicht

auf den Ordner, worin sich das Dokument mit der Verknüpfung befindet. Diese könnten die gleichen sein, es ist aber nicht zwingend. Relative Pfadangaben sind also mit einem gewissen Risiko behaftet.

Bestimmt fragen Sie sich, warum wir hier die Feldfunktion behandeln. Die Antworten darauf sind folgende:

- Nur so ist es möglich, den \\* *Mergeformat*-Schalter im Dokument vorhandener Grafiken gezielt hinzuzufügen oder zu entfernen.
- Wegen des Problems in Word 2002 und 2003 mit dem automatischen Hinzufügen des Schalters \\* *Mergeformat* bei Verwendung der Eigenschaften und Methoden des Objektmodells ist es vorteilhaft, auch die Änderungen der Pfadangabe, das Hinzufügen oder Entfernen des Schalters \d, die Sperrung sowie die Aktualisierung der Verknüpfung über die Feldcodes vorzunehmen.
- Der Verknüpfungspfad kann direkt im Feldcode bearbeitet werden, um in Word 2000 das Problem mit SourceFullName zu umgehen.

Das Beispiel in Listing 6.84 zeigt, wie Sie einen Feldcode benutzen, um eine verknüpfte Grafik einzufügen und deren Größe zu ändern. Dann wird die Prozedur *GrafikFeldcodeBearbeiten* aufgerufen, um den Feldcode zu ändern. Es ist insbesondere zu beachten, dass als Folge der Aktualisierung des Feldcodes das damit verbundene VBA-Objekt gelöscht wird. Da es im Code weiterhin gebraucht wird, muss es wieder hergestellt werden. Am Ende der Prozedur *GrafikAlsFeldEinfuegen* wird die Verknüpfung aufgelöst, so dass nur eine gewöhnliche eingebettete Grafik im Dokument zurückbleibt.

**Listing 6.84** Eine Grafik als Feldfunktion einfügen und den Feldcode anschließend bearbeiten

```
Private Const strGRAFIKPFAD1 As String = "C:\\Windows\\Seifenblase.bmp"
Private Const strGRAFIKPFAD2 As String = "C:\\Windows\\Feder.bmp"

Sub GrafikAlsFeldEinfuegen()
    Dim ils As Word.InlineShape
    Dim rng As Word.Range

    Set rng = Selection.Range
    'Grafik in die Zeile mit dem Text einfügen
    Set ils = rng.Fields.Add(Range:=rng, Type:=wdFieldEmpty, _
        Text:="IncludePicture " & strGRAFIKPFAD1 & " \* Mergeformat \d ", _
        PreserveFormatting:=False).Result.InlineShapes(1)
    'Grafikgröße um 50% reduzieren
    ils.ScaleHeight = 50
    ils.ScaleWidth = 50
    'Feldcode ändern
    GrafikFeldcodeBearbeiten ils, strGRAFIKPFAD2, True, True
    'Die Verknüpfung auflösen
    ils.Fields.Unlink
End Sub

Sub GrafikFeldcodeBearbeiten(ByRef ils As Word.InlineShape, _
    ByRef strPfad As String, ByVal bMergeFormat As Boolean, _
    ByVal bSaveInDocument As Boolean)

    Dim strMergeFormat As String
    Dim strSaveInDocument As String
```

Listing 6.84 Eine Grafik als Feldfunktion einfügen und den Feldcode anschließend bearbeiten (Fortsetzung)

```

Dim rng As Word.Range

Set rng = ils.Range
strMergeFormat = ""
strSaveInDocument = ""
If bMergeFormat Then
    strMergeFormat = " \* MergeFormat"
End If
If Not bSaveInDocument Then
    strSaveInDocument = " \d"
End If
With rng.Fields(1)
    .Code.Text = "IncludePicture " & Chr$(34) & strPfad & Chr$(34) _
        & strMergeFormat & strSaveInDocument & " "
    .Update
End With
'Das InlineShape-Objekt wird durch die Aktualisierung zerstört
'Wir stellen es wieder her
Set ils = rng.InlineShapes(1)
End Sub

```



Die Beispieldatei *Bsp06\_02\_Graf.doc* finden Sie auf der CD-ROM zum Buch im Ordner `\Beispiele\Kap06`.

## Layout-Optionen

Allgemein sind InlineShape-Objekte Shape-Objekten vorzuziehen. Einige der Gründe wurden im Abschnitt »Verknüpfungen erstellen und verwalten« in diesem Kapitel aufgelistet. Es gesellen sich zu den Problemen mit der Link-Verwaltung:

- Nur InlineShape-Objekte sind in der normalen Ansicht sichtbar.
- InlineShape-Objekte können problemlos als verborgener Text formatiert werden. (Eine oft gestellte Frage ist, wie man den Ausdruck von gewissen Grafiken unterbindet.)
- InlineShape-Objekte sind berechenbarer. Bei leichter Dokumentbeschädigung wird die Positionierung von Shape-Objekten instabil.
- Beschriftungen in der Zeichnungsebene werden bei der Bildung von Inhaltsverzeichnissen und Querverweisen ignoriert.

Manchmal geht es auch tatsächlich ohne Textflussformatierung, aber gelegentlich ist sie ein »notwendiges Übel«. Glücklicherweise stehen Alternativen bereit. Sie können beispielsweise eine Grafik als InlineShape in eine Tabellenzelle einfügen. Da Word 2000 später den Textfluss um Tabellen unterstützt, aber die Tabelle sowie ihr Inhalt weiterhin als Teil des Dokumenttextes behandelt, entfallen die aufgelisteten Probleme.

Beschriftungen

Eine Tabelle als Behälter für Grafiken ist auch nützlich für Beschriftungen in wissenschaftlichen Dokumenten, die rechts ausgerichtet neben der Grafik stehen sollen.

Eine weitere, oft benutzte Möglichkeit ist, die Grafik als InlineShape in einen Positionsrahmen einzufügen. Beim Verschieben bleiben Grafik und Beschriftung zusammen.

Die beiden oben vorgestellten Lösungen finden Sie mit Code-Beispielen im Kapitel 24.

Ein weiterer Vorteil des Einfügens einer Grafik in einen Positionsrahmen ist, wenn dieser eine feste Breite oder Höhe hat, passt sich die Grafik dieser Dimension proportional an und die Größe muss nicht weiter bearbeitet werden. Beispielcode hierfür finden Sie in Listing 6.85.

**Listing 6.85** Grafik in einen Positionsrahmen mit fester Breite einfügen

```
Private Const strMSGTITEL As String = "Grafik einfügen"

Sub GrafikInPositionsRahmenEinfuegen()
    Dim rng As Word.Range
    Dim fram As Word.Frame
    Dim ils As Word.InlineShape

    On Error GoTo Fehlerbehandlung
    Set rng = Selection.Range.Paragraphs(1).Range
    'Der Positionsrahmen wird die gesamte Markierung oder den Absatz, worin sich die
    'Einfügemarke befindet, beinhalten. Es soll aber nur eine Absatzmarke beinhalten.
    If Len(rng.Text) > 1 Then
        'Der Positionsrahmen soll neben dem Absatz stehen, worin sich die Einfügemarke
        'befindet. Ein leerer Absatz muss also VOR dem aktuellen Absatz eingefügt werden,
        ' falls dieser nicht leer ist
        rng.Collapse wdCollapseStart
        rng.Text = vbCr
    End If
    'Positionsrahmen einfügen und formatieren
    Set fram = rng.Frames.Add(rng)
    fram.Borders.Enable = False
    fram.Width = CentimetersToPoints(3.6)
    'Den Bereich in den Positionsrahmen setzen
    Set rng = fram.Range
    rng.Collapse Direction:=wdCollapseEnd
    Set ils = rng.InlineShapes.AddPicture(FileName:="C:\Windows\Feder.bmp", _
        LinkToFile:=False, Range:=rng)
    Set rng = ils.Range
    rng.Collapse Direction:=wdCollapseEnd
    'Ein Leerzeichen nach dem Positionsrahmen
    rng.InsertAfter " "
    rng.Font.Size = 1

Exit Sub
Fehlerbehandlung:
Select Case Err.Number
    Case 4605
        MsgBox "Die Einfügemarke muss im Dokumenttext sein.", _
            vbOKOnly + vbCritical, strMSGTITEL
    Case Else
        MsgBox "Unerwarteter Fehler: " & CStr(Err.Number) & vbCr & vbCr & _
            Err.Description, vbOKOnly + vbCritical, strMSGTITEL
End Select
End Sub
```



Die Beispieldatei *Bsp06\_03\_Graf.doc* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap06*.

## Die Positionierung von Shape-Objekten

Da Word InlineShape-Objekte wie andere Textzeichen im Dokument behandelt, bedarf das Thema Positionierung keiner eingehenden Diskussion mehr. Shape-Objekte (und Positionsrahmen) hingegen unterliegen einer komplexen Regelung von Optionen. In der Benutzerschnittstelle befinden sich diese in der Menüfolge *Format/Grafik/Layout/Weitere*, in der Registerkarte *Bildposition* (Abbildung 6.46) bzw. unter *Format/Positionsrahmen*. (Nicht alle der hier vorgestellten Optionen gelten für Positionsrahmen, aber die Verhaltensregeln sind die gleichen.) Es ist empfehlenswert, sich mit der Wirkung der verschiedenen Optionen vertraut zu machen, bevor Sie versuchen, grafische Objekte programmtechnisch zu positionieren.

Abbildg. 6.46 Die Optionen, die die Positionierung von Shape-Objekten regeln



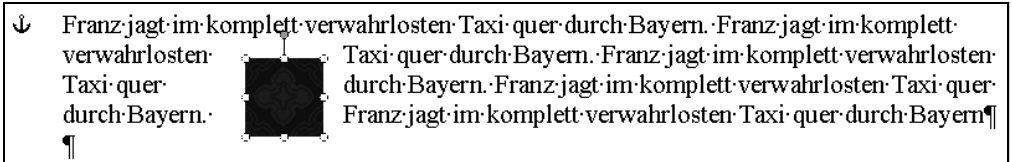
Die erste und wichtigste Regel für die Positionierung ist, dass ein grafisches Objekt (Shape oder Positionsrahmen (Frame)) *immer* mit einem Absatz verankert ist, und sich *immer* auf der gleichen Seite befindet wie dieser Absatz. In Abbildung 6.47 sehen Sie einen Objektanker. Falls das entsprechende Kontrollkästchen über *Extras/Optionen* auf der Registerkarte *Ansicht* aktiviert ist, soll er sichtbar sein, wenn sein grafisches Objekt markiert ist.

Der Anker kann mit der Maus verschoben werden, eine programmtechnische Schnittstelle dafür gibt es jedoch nicht. Der zu verankernde Bereich kann lediglich beim Einfügen eines Shape-Objekts über die *AddPicture*-Methode bestimmt werden. Zwar hat das Shape-Objekt eine Anker-Eigenschaft, diese ist jedoch nur lesbar und gibt den ankernden Bereich (ein Range-Objekt) zurück.

**PROFITIPP**

Wenn Sie ein bereits im Dokument vorhandenes grafisches Objekt explizit mit einem bestimmten Absatz verankern wollen, müssen Sie das Objekt ausschneiden (Cut-Methode), und diesen Absatz als Zielbereich für das Einfügen (Paste-Methode) auswählen.

**Abbildg. 6.47** Der Anker eines grafischen Objekts befindet sich immer links des Absatzes, mit dem das Objekt verankert ist



Veran-  
kern =  
Lock-  
Anchor

Wird das grafische Objekt mit der Maus verschoben, springt der Anker meistens zum nächst liegenden Absatz. Um dieses Verhalten zu unterbinden, muss das Kontrollkästchen *Verankern* auf der Registerkarte *Bildposition* aktiviert werden (Menübefehl *Format/Grafik*, Registerkarte *Layout*, Schaltfläche *Weitere*). Diese Option entspricht der Eigenschaft `LockAnchor` des `Shape`- oder `Frame`-Objekts.

Viele meinen, diese Option würde das grafische Objekt sperren, so dass es nicht verschoben werden kann, oder gar fest mit einer bestimmten Seite verbunden wird. Das ist nicht der Fall; dafür gibt es in Word gar keine Möglichkeit (außer der des allgemeinen Dokumentschutzes).

Objekt  
mit Text  
verschie-  
ben

Grundsätzlich gibt es zwei Arten der Positionierung für grafische Objekte: relativ zur Seite oder relativ zum verankernden Text. In der Benutzerschnittstelle entsprechen diese der Einstellung des Kontrollkästchens *Objekt mit Text verschieben* auf der Registerkarte *Bildposition*. Ist die Option nicht aktiviert, bleibt die Grafik an der gleichen Position auf der Seite, in der sich der verankernde Absatz befindet, egal wo der Absatz auf der Seite steht. Wird er zu einer anderen Seite verschoben, geht das grafische Objekt mit, und steht wieder in der gleichen Position relativ zur neuen Seite. War die Grafik beispielsweise 3 cm vom linken und 5 cm vom oberen Seitenrand auf Seite 4 zu finden, steht sie 3 cm vom linken und 5 cm vom oberen Seitenrand auch auf Seite 5, 6, oder gar 75.

Die genaue Position wird mit den weiteren Optionen in den beiden oberen Abschnitten der Registerkarte festgelegt. Die Möglichkeiten sind vielfältig und etwas verwirrend. Im Objektmodell sorgen eine Kombination von `RelativeVerticalPosition` (senkrecht) und von `RelativeHorizontalPosition` (waagrecht) in Zusammenspiel mit den `Top`-(oben) bzw. `Left`-(links) Eigenschaften für die Festlegung des Layouts. Die möglichen Werte der ersten beiden Eigenschaften sind in Tabelle 6.17 sowie in Tabelle 6.18 zusammengefasst.

**Tabelle 6.17** Werte für die Eigenschaft *RelativeVerticalPosition*

Enumeration	Wert	Beschreibung
<code>wdRelativeVerticalPositionLine</code>	3	Der Abstand wird relativ zur verankernden Zeile gemessen (mit Text verschieben)
<code>wdRelativeVerticalPositionMargin</code>	0	Der Abstand wird vom oberen Textrand gemessen
<code>wdRelativeVerticalPositionPage</code>	1	Der Abstand wird vom oberen Seitenrand gemessen
<code>wdRelativeVerticalPositionParagraph</code>	2	Der Abstand wird relativ zum verankernden Absatz gemessen (mit Text verschieben)

Tabelle 6.18 Werte für die Eigenschaft *RelativeHorizontalPosition*

Enumeration	Wert	Beschreibung
wdRelativeHorizontalPositionCharacter	3	Der Abstand wird relativ zum verankernden Textzeichen gemessen (mit Text verschieben)
wdRelativeHorizontalPositionColumn	2	Der Abstand wird vom linken Spaltenrand gemessen
wdRelativeHorizontalPositionMargin	0	Der Abstand wird vom linken Textrand gemessen
wdRelativeHorizontalPositionPage	1	Der Abstand wird vom linken Seitenrand gemessen

Zusätzlich zu einem numerischen Wert des Datentyps Single akzeptieren Left und Top noch die Enumerationen in Tabelle 6.19.

Tabelle 6.19 Werte für der Eigenschaften *Left* und *Top*

Enumeration	Wert	Position, relativ zum verankernden Text
wdShapeBottom	–999997	Unten
wdShapeCenter	–999995	Zentriert
wdShapeInside	–999994	Die Grafik wird am Bundrand eines Buches ausgerichtet (Gerade/ungerade anders in der Menüfolge Datei/Seiteneinrichten/Layout)
wdShapeLeft	–999998	links
wdShapeOutside	–999993	Die Grafik wird am Außenrand eines Buches ausgerichtet (Gerade/ungerade anders in der Menüfolge Datei/Seiteneinrichten/Layout)
wdShapeRight	–999996	Rechts
wdShapeTop	–999999	Oben

Um eine Grafik 3 cm vom linken Seitenrand und bündig mit dem oberen Textrand zu positionieren, wird wie in Listing 6.86 vorgegangen.

Listing 6.86 Ein grafisches Objekt relativ zur Seite positionieren

```
Sub GrafikGenauAufDerSeitePositionieren()
    Dim shp As Word.Shape

    'Zweites InlineShape im Dokument in ein Shape konvertieren
    Set shp = ActiveDocument.InlineShapes(2).ConvertToShape
    'Relativ zur Seite positionieren
    shp.RelativeHorizontalPosition = wdRelativeHorizontalPositionPage
    shp.Left = CentimetersToPoints(3)
    shp.RelativeVerticalPosition = wdRelativeVerticalPositionMargin
    shp.Top = wdShapeTop
End Sub
```



Die Beispieldatei *Bsp06\_03\_Graf.doc* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap06*.

### TIPP

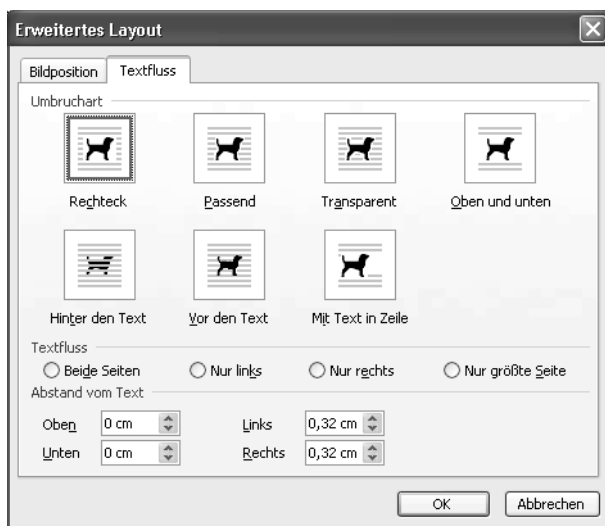
### Die Funktion *CentimetersToPoints*

In vielen Word-Dialogfeldern können Dimensionen in verschiedenen Maßen angegeben werden, wie etwa Zoll (Inches), Zentimeter und Points. Intern erwartet Word aber nur eine dieser Maßangaben – meistens Points –, was bedeutet, die anderen müssen konvertiert werden. Word stellt im Objektmodell eine Sammlung solcher Konvertierfunktionen zur Verfügung, wie *PointsToCentimeters*, *InchesToPoints*, *MillimetersToPoints*, *LinesToPoints*, *PicasToPoints* und umgekehrt. Diese Funktionen gehören ausschließlich dem Word- und keinem anderen Office-Objektmodell an.

## Textfluss-Formatierung

Neben der Registerkarte *Bildposition* enthält das Dialogfeld *Erweitertes Layout* die Registerkarte *Textfluss* (Abbildung 6.48), die einige zusätzliche Optionen zur Registerkarte *Layout* im Dialogfeld *Grafik formatieren* anbietet.

Abbildg. 6.48 Der genaue Textfluss wird im Dialogfeld *Erweitertes Layout* festgelegt



Textfluss

Im Objektmodell umfasst die Eigenschaft *WrapFormat* die Optionen im Abschnitt *Textfluss* und *Abstand vom Text*. Auch die meisten Umbrucharten sind dieser Eigenschaft zugeteilt, außer *Hinter den Text* und *Vor den Text*, die unter die Methode *ZOrder* fallen. Die *WrapFormat*-Eigenschaften mit ihren Werten und Wirkung sind in Tabelle 6.20 aufgelistet.



Tabelle 6.20 Die *WrapFormat*-Eigenschaften, die den Textfluss festlegen

Eigenschaft	Enumeration	Wert	Beschreibung oder Dialogfeldoption
Type	wdWrapInline	7	Gilt nur für Office-AutoFormen, die nicht in <b>Inl ineShape</b> -Objekte umgewandelt werden können. Die AutoForm verhält sich wie ein <b>Inl ineShape</b> , ist aber keines, was an den weißen Anfassern zu erkennen ist.
	wdWrapNone	3	Stellt das grafische Objekt vor den Text. Wird zurückgegeben, wenn die Grafik vor oder hinter dem Text steht.
	wdWrapSquare	0	Entspricht der Option <i>Rechteck</i>
	wdWrapThrough	2	Entspricht der Option <i>Transparent</i> . (Die Hintergrundfarbe scheint durch die Stellen, die mit »transparente Farbe« formatiert sind. Steht nicht bei allen Grafiktypen zur Verfügung.)
	wdWrapTight	1	Entspricht der Option <i>Passend</i> . Der Textfluss folgt seitlich dem Umriss des Hauptteils des Bildes und ignoriert den Hintergrund.
	wdWrapTopBottom	4	Entspricht der Option <i>Oben und unten</i> . Allgemein ist ein <b>Inl ineShape</b> , allein stehend in einem Absatz, dieser Einstellung vorzuziehen.
Side	wdWrapBoth	0	<i>Textfluss/Beide Seiten</i>
	wdWrapLargest	3	<i>Textfluss/Nur größte Seite</i>
	wdWrapLeft	1	<i>Textfluss/Nur links</i>
	wdWrapRight	2	<i>Textfluss/Nur rechts</i>
DistanceBottom			<i>Abstand vom Text/Unten</i>
DistanceLeft			<i>Abstand vom Text/Links</i>
DistanceRight			<i>Abstand vom Text/Rechts</i>
DistanceTop			<i>Abstand vom Text/Oben</i>

Wenn wir das grafische Layout eines Word-Dokuments betrachten, sehen wir, dass es aus drei dimensional »Ebenen« besteht: aus der Textebene, der Ebene dahinter und der Ebene davor. Diese sind in der Abbildung 6.49 klar ersichtlich. Von links nach rechts: hinter dem Text, die Textebene und vor dem Text.

Abbildg. 6.49 Die drei grafischen Ebenen eines Word-Dokuments


**HINWEIS**

In der Lösung »Tischkarte mit WordArt« (Kapitel 27) finden Sie Beispielcode für die Festlegung und Bestimmung der meisten Eigenschaften des Shape-Objekts, die die Positionierung und Formatierung beeinflussen.

Zeichen/Reihenfolge

Zudem können in allen drei Ebenen Grafiken übereinander liegen. Diese Reihenfolge wird in der Benutzerschnittstelle über den Menübefehl *Reihenfolge* der Schaltfläche *Zeichnen* in der gleichnamigen Symbolleiste definiert. Im Objektmodell entspricht diesem Befehl die Methode *ZOrder*, deren Werte in Tabelle 6.21 aufgelistet sind.

Tabelle 6.21 Die Werte der Methode *ZOrder*

Enumeration	Menübefehl
<code>msoBringForward</code>	<i>Eine Ebene nach vorne</i>
<code>msoBringToFront</code>	<i>Vor den Text bringen</i>
<code>msoBringToBack</code>	<i>In den Vordergrund</i>
<code>msoSendBackward</code>	<i>Eine Ebene nach hinten</i>
<code>msoSendBehindText</code>	<i>Hinter den Text bringen</i>
<code>msoSendToBack</code>	<i>In den Hintergrund</i>

Das Zusammenspiel der *WrapFormat*-Eigenschaften und der *ZOrder*-Methode kann recht spannend sein. Leider gibt es im Word-Objektmodell keine Methode, um festzustellen, in welcher dreidimensionalen Reihenfolge Grafiken in einer Ebene zueinander stehen. Es bietet zwar die Eigenschaft *ZOrderPosition*, aber diese verrät uns nur, in welcher Reihenfolge die Objekte ins Dokument eingefügt wurden (die zuletzt eingefügte steht an oberster Stelle), nicht welche Grafik vor oder hinter einer anderen steht.

Zusammenfassend:

- Sie können jederzeit herausfinden, mit welchem Textfluss eine Grafik formatiert ist, was meist verrät, ob sie sich in der Textebene befindet (nur `wdWrapNone` ist nicht in der Textebene).
- Es ist immer möglich, ein grafisches Objekt in eine beliebige Ebene, mit einem beliebigen Textfluss, zu positionieren.
- Auch die Festlegung einer dreidimensionalen Reihenfolge stellt kein Problem dar.
- Nicht herausfinden können Sie allerdings mit VBA, in welcher Reihenfolge grafische Objekte in der gleichen Ebene übereinander liegen.

Das folgende Beispiel soll diese Prinzipien grafisch veranschaulichen.

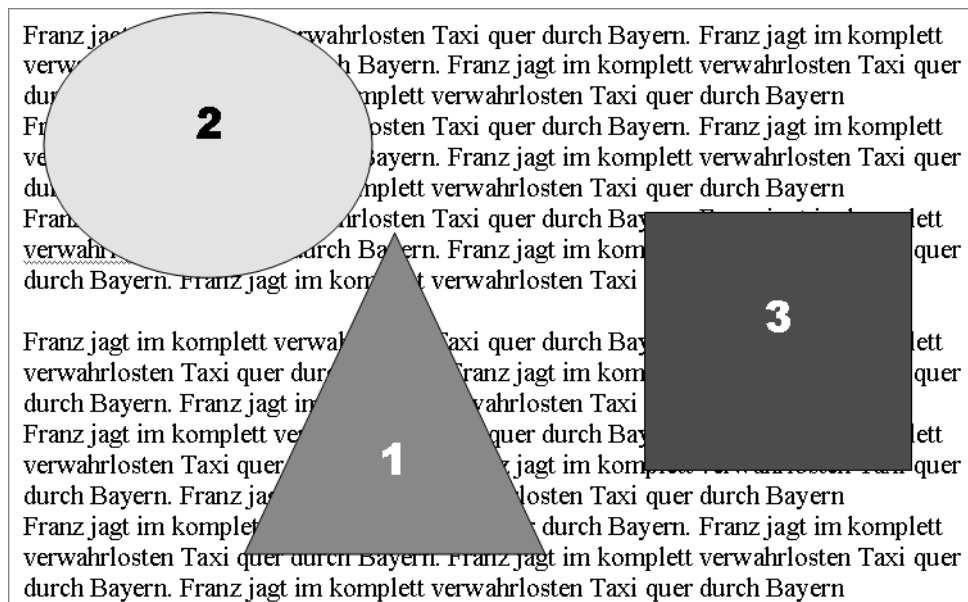
In Abbildung 6.50 sehen Sie drei AutoFormen (alle in der gleichen Ebene vor dem Text stehend), die in der Reihenfolge nummeriert sind, wie sie in das Dokument eingefügt wurden. Die Prozedur *GrafikenAusrichtenUndEinordnen* in Listing 6.87 richtet sie zuerst waagrecht sowie senkrecht zentriert auf der Seite (also übereinander) aus. Wie in Abbildung 6.51 ersichtlich, steht die zuletzt eingefügte Grafik (Nummer 3) vorn. Danach durchläuft das Makro nochmals alle Grafiken und schickt zuerst die Grafik mit `ZOrderPosition 1` nach hinten, dann die Nummer 2 und zuletzt die Nummer 3, so dass am Schluss die Nummer 1 wie in Abbildung 6.52 vorn erscheint.

Wenn anschließend das orangefarbene Dreieck hinter den Text gestellt wird, behält es seine `ZOrderPosition`, erscheint aber hinter den anderen Grafiken, da es hinter dem Text steht. Dieser Zustand ist jedoch, was der `ZOrderPosition` angeht, rein optisch.

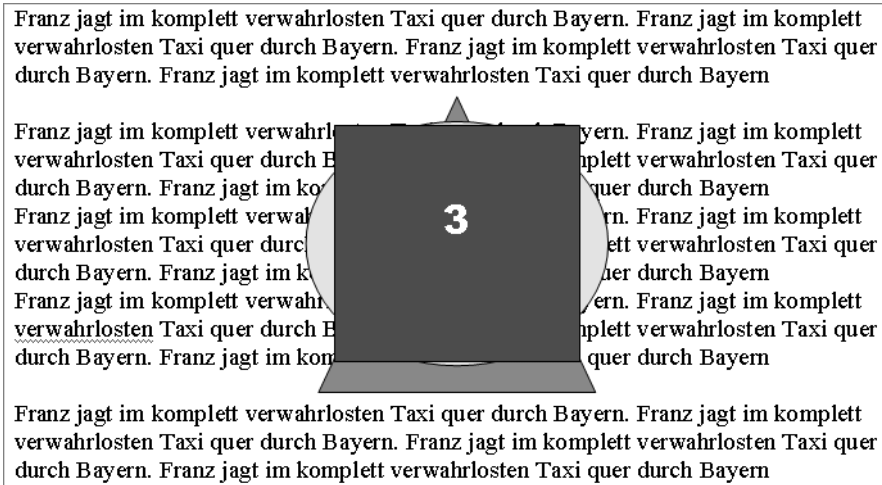
#### HINWEIS

Bitte bemerken Sie, wenn Sie mit einer `For Each...Next`-Schleife alle Shape Objekte in einem Dokument oder Bereich durchlaufen, dass diese in der Reihenfolge ihrer Verankerung geschieht. Mit dieser Reihenfolge hat die `ZOrderPosition` nichts zu tun.

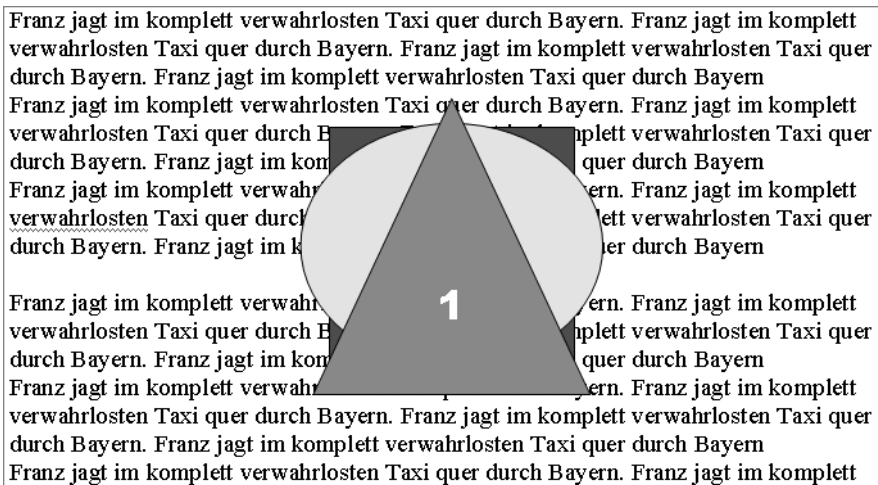
Abbildg. 6.50 Die Grafiken sind in der Reihenfolge nummeriert, in welcher sie eingefügt wurden



Abbildg. 6.51 Zieht man sie übereinander, liegt die zuletzt eingefügte zuoberst



Abbildg. 6.52 Die Prozedur in Listing 6.87 hat die ZOrder umgekehrt



Listing 6.87 AutoFormen zentriert ausrichten und hintereinander anordnen

```
Sub GrafikenAusrichtenUndEinordnen()
    Dim rng As Word.Range
    Dim shpRng As Word.ShapeRange
    Dim shp As Word.Shape
    Dim pgs As Word.PageSetup

    On Error GoTo FehlerBehandlung
    ' Markierung in den Text verschieben.
    Selection.GoTo What:=wdGoToPage,
        Count:=Selection.Information(wdActiveEndPageNumber)
    ' Nur die Grafiken dieser Seite bearbeiten.
```

Listing 6.87 AutoFormen zentriert ausrichten und hintereinander anordnen (Fortsetzung)

```

Set rng = Selection.Bookmarks("\Page").Range
Set shpRng = rng.ShapeRange
' Objektvariable, um Seitenrandinformationen zu ermitteln.
Set pgs = rng.Sections(1).PageSetup

'Die Grafiken werden in der Reihenfolge ihrer Verankerungen bearbeitet!
For Each shp In shpRng
    'Grafik zentriert relativ zu den Texträndern positionieren.
    shp.RelativeHorizontalPosition = wdRelativeHorizontalPositionMargin
    shp.Left = wdShapeCenter
    shp.RelativeVerticalPosition = wdRelativeVerticalPositionMargin
    shp.Top = wdShapeCenter
Next shp

'Die Reihenfolge der Grafiken umkehren.
For Each shp In shpRng
    Debug.Print shp.Name, shp.ZOrderPosition
    If InStr(shp.TextFrame.TextRange.Text, "1") <> 0 Then shp.ZOrder msoSendToBack
Next shp
For Each shp In shpRng
    Debug.Print shp.Name, shp.ZOrderPosition
    If InStr(shp.TextFrame.TextRange.Text, "2") <> 0 Then shp.ZOrder msoSendToBack
Next shp
For Each shp In shpRng
    Debug.Print shp.Name, shp.ZOrderPosition
    If InStr(shp.TextFrame.TextRange.Text, "3") <> 0 Then shp.ZOrder msoSendToBack
    shp.Select
Next shp
' Die oberste Grafik hinter den Text schicken; sie erscheint jetzt
' hinter allen anderen.
' Aber ihre ZOrderPosition bleibt die gleiche.
For Each shp In shpRng
    ' If shp.ZOrderPosition = (3) Then shp.ZOrder msoSendBehindText
    ' Debug.Print shp.Name, shp.ZOrderPosition
' Next shp
Exit Sub

FehlerBehandlung:
Select Case Err.Number
    Case 5852
        MsgBox "Fehler: " & Err.Number & ". (Objekt nicht vorhanden)" & vbCrLf & _
            "Das Makro findet keine grafischen Objekte auf dieser Seite, " & _
            "die über dem Text liegen.", vbCritical + vbOKOnly
    Case Else
        MsgBox "Fehler: " & Err.Number & vbCrLf & _
            "Beschreibung: " & Err.Description, vbCritical + vbOKOnly
End Select
End Sub

```



Die Beispieldatei *Bsp06\_04\_Graf.doc* finden Sie auf der CD-ROM zum Buch im Ordner \Beispiele\Kap06.

## Zeichnungsobjekte (AutoFormen)

Wie schon in diesem Abschnitt erwähnt, ist die Zeichnungsfunktionalität in Word ein gemeinsames Office-Anwendungspaket, das von mehreren Anwendungen (wie PowerPoint und Excel) benutzt wird. Wir werden daher die in VBA zur Verfügung gestellte Zeichnungsfunktionalität nicht eingehend diskutieren, da das Thema selbst ein ganzes Buch füllen würde und in anderen Büchern behandelt wird. Wir greifen lediglich einige immer wiederkehrende Fragen auf, die eine allgemeine Idee geben, wie mit VBA AutoFormen in einem Word-Dokument erstellt werden.

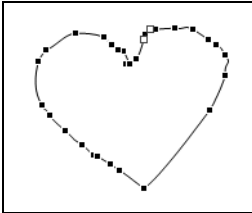
### HINWEIS

Angaben zu WordArt, das ein Objektmodell besitzt, finden Sie in Kapitel 11.

### Freihandformen bearbeiten

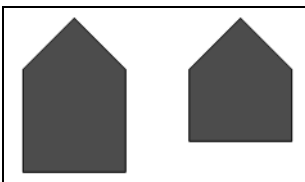
Etwas, das immer wieder für Unklarheit sorgt, ist die `Points`-Eigenschaft der `ShapeNodes`-Auflistung. Ein `ShapeNode` ist entweder ein Eckpunkt oder bei Kurven ein Punkt, der die Rundung bestimmt. Durch Änderung der Koordinaten (`Points`) wird das Aussehen einer *von Hand gezeichneten* AutoForm geändert. In der Benutzerschnittstelle werden sie über den Menüpunkt *Punkte bearbeiten* der *Zeichnen*-Symbolleiste in der gleichnamigen Symbolleiste aktiviert und sehen wie in Abbildung 6.53 aus.

**Abbildg. 6.53** Durch Bearbeitung der Punkte kann die Form einer gezeichneten Grafik angepasst werden. Die Punkte können verschoben, gelöscht oder neu hinzugefügt werden.



Die Unklarheit stammt aus den Hilfe-Angaben zur Eigenschaft. Obwohl die Objekt-Hierarchie `Shape.Nodes.Item.Points` lautet, erscheint eine Fehlermeldung (»die Typen sind unverträglich«), wenn eine als `Shape` deklarierte Objekt-Variable eingesetzt wird. Um das Problem zu umgehen, muss die Objekt-Variable als allgemeiner Typ `Object` deklariert werden. Listing 6.88 veranschaulicht das Problem und die Lösung. Das Makro erstellt ein Fünfeck als Vieleck (`Polygon`), dann werden die zwei senkrechten Seiten verkürzt, wie in Abbildung 6.54 ersichtlich.

**Abbildg. 6.54** Ein Fünfeck mit VBA erstellen und bearbeiten



Zuerst wird eine Objekt-Variable – `shp` – für die AutoForm als `Shape` deklariert. In der Datenfeld-Variablen `aEckpunkte` werden die Koordinaten des Fünfecks festgehalten. Je ein Koordinatenpaar

braucht es für jeden Richtungswechsel, zusätzlich eines für den Anfangs- sowie den Endpunkt. Diese werden in der ersten Dimension des Datenfelds festgehalten. Die zweite Dimension des Datenfelds hält die X- und Y-Koordinaten jedes Punkts relativ zur Seite fest.

**HINWEIS** Haben Anfangs- und Endpunkt einer Freihandform (Polyline) wie hier die gleichen Koordinaten, ist die Form geschlossen und kann gefüllt werden.

Nachdem die Koordinaten dem Datenfeld zugewiesen wurden, fügt die Prozedur das Fünfeck ein und setzt es der shp-Objektvariablen gleich. Damit kann die AutoForm weiter bearbeitet und formatiert werden – wie hier mit der Füllfarbe *Rot*.

Eigentlich sollten wir die gleiche Objekt-Variable einsetzen können, um die Eckpunktkoordinaten zu ändern. Nur tritt leider das beschriebene Problem auf. Es wird also die Objektvariable o\_shp als Object deklariert und dem Shape gleich gesetzt.

Noch ein Datenfeld wird gebraucht, um die Koordinaten des Punktes festzuhalten, den wir verschieben möchten. Auch hier muss eine neue Objekt-Variable her, da Points nur einer Objekt-Variablen des Datentyps Variant (aber keinem Datenfeld) zugewiesen werden kann. Zwei Ungereimtheiten also, worauf zu achten sind.

aPunkte hält also die X- und Y- Koordinaten des gewählten Eckpunktes (Node) fest, die den Variablen x und y zugewiesen werden. Mit der Methode SetPosition werden diese geändert. In diesem Fall bleibt die waagerechte Einstellung gleich, die senkrechte wird um 15 Punkte (typographisches Maß) verkürzt (höher gestellt).

**Listing 6.88** Ein Polygon erstellen und anschließend die Eckpunkte ändern

```
Sub EinFünfeckZeichnen()
    Dim shp As Word.Shape
    Dim aEckPunkte(1 To 6, 1 To 2) As Single
    Dim vw As Word.View
    Dim bCurVw As Boolean

    Set vw = ActiveDocument.ActiveWindow.View
    'In Word 2002 können Grafiken falsch positioniert werden,
    'wenn die oberen und unteren Seitenränder ausgeblendet sind.
    'Die Benutzereinstellung festhalten und am Schluss wieder herstellen
    'WORD 2000: die beiden folgenden Zeilen entfernen!
    bCurVw = vw.DisplayPageBoundaries
    If bCurVw = False Then vw.DisplayPageBoundaries = True

    aEckPunkte(1, 1) = 25
    aEckPunkte(1, 2) = 25
    aEckPunkte(2, 1) = 50
    aEckPunkte(2, 2) = 50
    aEckPunkte(3, 1) = 50
    aEckPunkte(3, 2) = 100
    aEckPunkte(4, 1) = 0
    aEckPunkte(4, 2) = 100
    aEckPunkte(5, 1) = 0
    aEckPunkte(5, 2) = 50
    aEckPunkte(6, 1) = 25
    aEckPunkte(6, 2) = 25
    Set shp = ActiveDocument.Shapes.AddPolyline(aEckPunkte)
    shp.RelativeHorizontalPosition = wdRelativeHorizontalPositionMargin
```

**Listing 6.88** Ein Polygon erstellen und anschließend die Eckpunkte ändern (*Fortsetzung*)

```

shp.Left = wdShapeRight
shp.RelativeVerticalPosition = wdRelativeVerticalPositionParagraph
shp.Top = 0
shp.WrapFormat.Type = wdWrapSquare
shp.Fill.ForeColor = 255

Dim o_shp As Object
Dim aPunkte As Variant
Dim x As Single
Dim y As Single
Set o_shp = shp
With o_shp.Nodes
    aPunkte = .Item(3).Points
    x = aPunkte(1, 1)
    y = aPunkte(1, 2)
    .SetPosition 3, x, y - 15
    aPunkte = .Item(4).Points
    x = aPunkte(1, 1)
    y = aPunkte(1, 2)
    .SetPosition 4, x, y - 15
End With

'WORD 2000: die folgende Zeile entfernen!
vw.DisplayPageBoundaries = bCurVw
End Sub

```

## Text in AutoFormen ansprechen

Den meisten AutoFormen kann Text hinzugefügt werden. Leider ist es nicht möglich, AutoFormen oder Textfeldern generell eine Formatvorlage oder andere Formatierung zuzuweisen. Neue AutoFormen werden immer mit der Formatvorlage *Standard* formatiert. Während es möglich ist, AutoFormen zu kopieren, stellen wir zunehmend Probleme in Word 2002 und 2003 fest. Word kann offensichtlich eine eingefügte AutoForm nicht zuverlässig vom kopierten Original unterscheiden, was zu mühsamen Vorgehensweisen bei der Dokumentbearbeitung führt.

Es ist also ratsam, jede AutoForm und Textfeld einzeln zu erstellen und zu formatieren. Beinhaltet das Dokument viele solche Objekte, ist dieser Vorgang kaum weniger mühsam. Eine programmtechnische Lösung drängt sich also auf. In Listing 6.89 finden Sie Beispielcode, der alle Textfelder (und ausschließlich diese) mit Textinhalt im Dokumenttext gleich formatiert.

**Listing 6.89** Den Text in allen Textfeldern des Dokumentkörpers mit Arial 10, fett, formatieren

```

Sub AlleTextBereicheFormatieren()
    Dim shp As Word.Shape

    For Each shp In ActiveDocument.Shapes
        If shp.Type = msoTextBox Then
            With shp.TextFrame
                If .HasText Then
                    .TextRange.Font.Name = "Arial"
                    .TextRange.Font.Size = 10
                    .TextRange.Bold = True
                End If
            End With
        End If
    Next shp
End Sub

```



**Listing 6.89** Den Text in allen Textfeldern des Dokumentkörpers mit Arial 10, fett, formatieren (*Fortsetzung*)

```
End With
End If
Next shp
End Sub
```



Sie finden den Code in der Beispieldatei *Bsp06\_05\_Graf.doc* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap06*.

## Abschnitte im Dokument: Das Section-Objekt

Jedes Dokument umfasst grundsätzlich einen Abschnitt. Weitere Abschnitte werden benötigt, wenn innerhalb des Dokuments ein unterschiedliches Seitenlayout benötigt wird (beispielsweise einige Seiten im Dokument werden im Querformat gedruckt).

Die grundlegenden Eigenschaften eines Dokuments werden pro Abschnitt definiert. Dazu zählen unter anderem die Einstellungen für

- Papierformat und Seitenausrichtung
- Seitenränder und Bundsteg
- Abstand der Kopf- und Fußzeilen zum Text
- Anzahl der Zeitungsspalten
- Papierzufuhr für den Ausdruck

Alle diese Eigenschaften beziehen sich immer auf den definierten Abschnitt. Eine Besonderheit bieten die Kopf- und Fußzeilen. Hier wird beispielsweise eine Kopfzeile automatisch mit der Kopfzeile des folgenden Abschnitts verknüpft, sofern dies nicht explizit unterbunden wird. Somit kann sich eine Änderung der Darstellung innerhalb einer Kopf- bzw. Fußzeile auch auf andere Abschnitte auswirken. Weitere Informationen zu den Kopf- und Fußzeilen sind im Abschnitt »Die Seite gestalten: Das HeaderFooter-Objekt« in diesem Kapitel aufgeführt.

Manuell wird ein Abschnittswechsel durch Aufruf des Menübefehls *Einfügen/Manueller Umbruch* eingefügt.

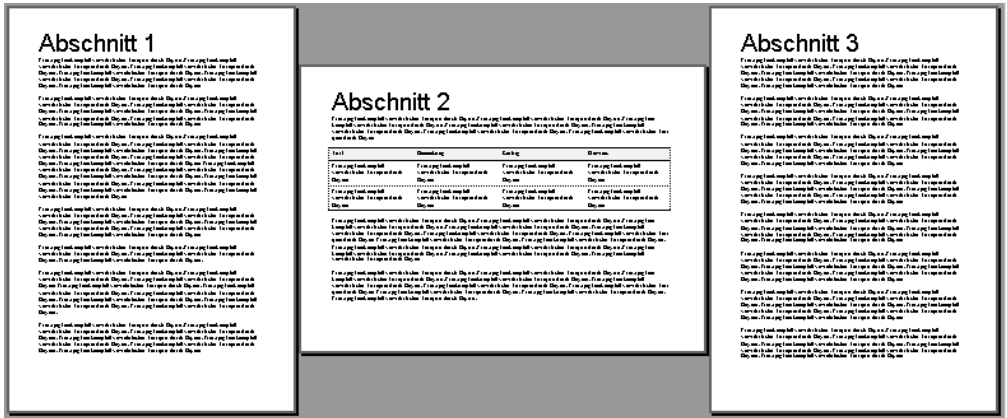
Innerhalb eines Dokuments können vier verschiedene Arten von Abschnittswechsel eingefügt werden. Der eigentliche Abschnittswechsel ist eine Methode eines beliebigen Range-Objekts:

```
ActiveDocument.Range.InsertBreak Type:=wdSectionBreakNextPage
```

Eine komplette Liste aller möglichen Umbrüche ist in der Tabelle 6.10 zusammengefasst.

Wie in Abbildung 6.55 und in Abbildung 6.56 dargestellt, lassen sich Abschnittswechsel zur Gestaltung von großen Dokumenten einsetzen. So können die einzelnen Bereiche über ein unterschiedliches Seitenlayout verfügen. Oder es kann beispielsweise mit einem Abschnittswechsel vom Typ `wdSectionBreakOddPage` sichergestellt werden, dass ein neuer Abschnitt stets auf der rechten Seite eines Buches beginnt.

**Abbildg. 6.55** Unterschiedliches Seitenlayout: Die Abschnitte 1 und 3 werden im Hochformat, der Abschnitt 2 wird hingegen im Querformat ausgedruckt

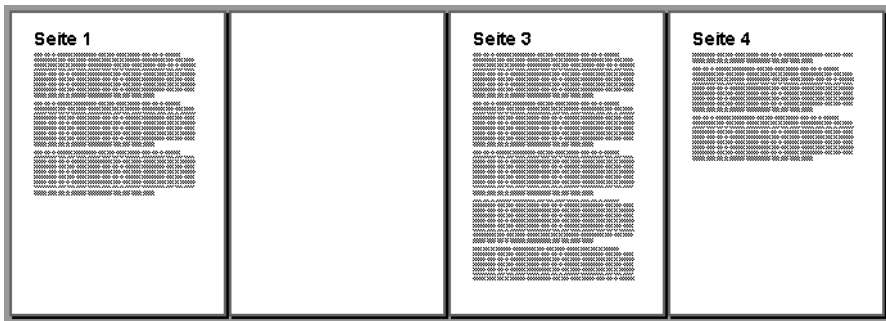


### TIPP

Die Abschnittswechsel innerhalb eines Dokuments sind sehr fragile Strukturen. Vor allem in großen Dokumenten mit mehreren Abschnittswechseln kann es vorkommen, dass der gespeicherte Inhalt eines Dokuments »durcheinander« gerät und so das Dokument beschädigt wird. In vielen Fällen können bei einem beschädigten Dokument der Text und die Formatierungen dennoch gerettet werden.

Jeder einzelne Abschnitt wird mittels der Zwischenablage in ein neues Dokument kopiert. Bei diesem Vorgang muss sichergestellt werden, dass der ganze Inhalt eines Abschnitts *ohne* den Abschnittswechsel (Linie mit feinen Doppelpunkten) kopiert wird. Beim letzten Abschnitt muss der ganze Inhalt *ohne* die letzte Absatzmarke kopiert werden. Anschließend kann das Dokument neu aufgebaut werden.

**Abbildg. 6.56** Die Seite 2 wird als leere Seite gedruckt, wenn ein Abschnittswechsel vom Typ `wdSectionBreakOddPage` eingefügt wird



Headers  
Footers

Grundsätzlich verfügt jeder Abschnitt über eigene Kopf- und Fußzeilen. Dies gilt auch für einen Abschnittswechsel vom Typ `wdSectionBreakContinuous` (vgl. Abbildung 6.58). Hier werden die zugehörigen Kopf- bzw. Fußzeilen erst sichtbar, wenn der betreffende Abschnitt größer als eine Seite ist.

Die Seitennummerierung kann für jeden Abschnitt individuell festgelegt werden. Somit ist es möglich, dass die Seitennummerierung eines jeden Abschnitts erneut bei »1« beginnt. Obwohl sich die Seitennummerierung auf einen bestimmten Abschnitt bezieht, müssen die betreffenden Einstellungen im zugehörigen HeaderFooter-Objekt vorgenommen werden (vgl. Listing 6.100).

Die Headers- und Footers-Eigenschaften geben je eine HeadersFooters-Auflistung zurück. Die Eigenschaften und Methoden dieser Auflistung werden im Abschnitt »Die Seite gestalten: Das HeaderFooter-Objekt« in diesem Kapitel beschrieben.

Page- Die PageSetup-Eigenschaft repräsentiert das Seitenlayout des Abschnitts. Die Eigenschaft gibt ein Setup PageSetup-Objekt zurück. Die Eigenschaften und Methoden dieser Auflistung sind im Abschnitt »Die Seite definieren: Das PageSetup-Objekt« in diesem Kapitel erläutert.

Protec- Jedes Dokument kann für die Texteingabe geschützt werden. Dieser Dokumentschutz umfasst ver tedFor- verschiedene Stufen. Eine dieser Schutzstufen lässt das Erfassen von Daten innerhalb von Formularfeldern zu. Mit der ProtectedForForms-Eigenschaft wird festgelegt, in welchen Abschnitten die Formularfelder gesperrt bzw. zur Erfassung freigeschaltet sind:

```
ActiveDocument.Sections(2).ProtectedForForms = False
```

Mehr zu diesem Thema lesen Sie im Abschnitt »Formulare: Das FormField-Objekt« in diesem Kapitel.

## Bereiche im Dokument: Das StoryRanges-Objekt

Ein StoryRange ist eine Dokumentkomponente innerhalb eines Dokuments. Diese einzelnen Dokumentkomponenten sind voneinander getrennt und müssen individuell behandelt werden. Werden in einem Dokument neben dem Haupttext noch Kopfzeilen und Fußnoten eingefügt, so beinhaltet dieses Dokument eine eigene Dokumentkomponente für den Haupttext, die Kopfzeilen und die Fußnoten.

Die separate Behandlung der einzelnen Dokumentkomponenten muss dann angewandt werden, wenn beispielsweise alle Felder eines Dokuments aktualisiert oder alle Shape-Objekte bearbeitet werden müssen:

```
MsgBox ActiveDocument.Shapes.Count 'Beispiel Ergebnis: 3
```

Die vorstehende Programmzeile liefert den Wert 3 zurück, obwohl im Beispieldokument (*Bsp06\_01a.doc*) vier Shape-Objekte eingefügt wurden. Dies liegt daran, dass mit diesem Aufruf nur der Haupttext des Dokuments überprüft wurde. Die eben aufgeführte Programmzeile ist äquivalent zur nachstehenden Zeile:

```
MsgBox ActiveDocument.StoryRanges(wdMainTextStory).ShapeRange.Count 'Ergebnis: 3
```

Da die anderen Dokumentkomponenten vom Haupttext getrennt behandelt werden, konnten nicht alle vorhandenen Shape-Objekte bei der Zählung berücksichtigt werden. Diesem Umstand trägt das

Listing 6.90 Rechnung. Hier werden alle vorhandenen StoryRanges des Dokuments durchforstet und die gefundenen Shape-Objekte aufaddiert.

**Listing 6.90** Zählen der *Shape*-Objekte im ganzen Dokument, also unter Berücksichtigung aller *StoryRanges*

```
Sub Shapes_Zählen_StoryRanges()
    Dim rng As Word.Range
    Dim intZähler As Integer

    For Each rng In ActiveDocument.StoryRanges
        intZähler = intZähler + rng.ShapeRange.Count
    Next rng

    MsgBox intZähler 'Beispiel Ergebnis: 4
End Sub
```

**WICHTIG** Das Überprüfen aller StoryRanges-Objekte, wie dies in Listing 6.90 umgesetzt wurde, ist jedoch nur ein erster Schritt zur kompletten Lösung. Einzelne Dokumentkomponenten bestehen nämlich aus mehreren Teilen. Verfügt beispielsweise ein Dokument über zwei Abschnitte, und die Kopfzeile im zweiten Abschnitt ist nicht mit der vorherigen verknüpft, so besteht das StoryRanges-Objekt vom Typ *wdPrimaryHeaderStory* aus zwei Elementen. Also müssten beide Elemente bei der Zählung berücksichtigt werden. Dies wird im Listing 6.90 jedoch nicht durchgeführt.

Um festzustellen, ob die aktuelle Dokumentkomponente über weitere Elemente verfügt, steht die *NextStoryRange*-Eigenschaft zur Verfügung. In Tabelle 6.23 wurden die möglichen Werte dieser Eigenschaft zusammengestellt.

Mehrere Programmbeispiele, die alle Dokumentkomponenten und deren zugehörigen Elemente berücksichtigen, sind im aktuellen Kapitel im Abschnitt »Die Nadel im Heuhaufen: *Find/Replace* einsetzen« aufgeführt. Gerade beim Suchen und Ersetzen müssen nämlich ebenfalls alle StoryRanges-Objekte und deren Elemente berücksichtigt werden, damit sichergestellt ist, dass alle Textstellen im Dokument bearbeitet wurden.

In der Tabelle 6.22 sind die Elemente der StoryRanges-Auflistung zusammengefasst. Die einzelnen Dokumentkomponenten werden innerhalb eines jeden Dokuments dynamisch angelegt, sobald diese bearbeitet werden. Jedes Dokument beinhaltet mindestens ein StoryRanges-Objekt vom Typ *wdMainTextStory*.

**Tabelle 6.22** Die einzelnen Elemente der *StoryRanges*-Auflistung

WdStoryType-Enum	Wert	Bedeutung
wdMainTextStory	1	Haupttext des Dokuments
wdFootnotesStory	2	Fußnoten
wdEndnotesStory	3	Endnoten
wdCommentsStory	4	Kommentare
wdTextFrameStory	5	Textrahmen

**Tabelle 6.22** Die einzelnen Elemente der *StoryRanges*-Auflistung (Fortsetzung)

WdStoryType-Enum	Wert	Bedeutung
wdEvenPagesHeaderStory	6	Kopfzeilen auf den Seiten mit gerader Seitenzahl, sofern <i>Gerade/Ungerade anders</i> aktiviert wurde. <sup>a</sup>
wdPrimaryHeaderStory	7	Standardkopfzeile
wdEvenPagesFooterStory	8	Fußzeilen auf den Seiten mit gerader Seitenzahl, sofern <i>Gerade/Ungerade anders</i> aktiviert wurde. <sup>a</sup>
wdPrimaryFooterStory	9	Standardfußzeile
wdFirstPageHeaderStory	10	Kopfzeile auf der ersten Seite des Dokuments, sofern <i>Erste Seite anders</i> aktiviert wurde. <sup>a</sup>
wdFirstPageFooterStory	11	Fußzeile auf der ersten Seite des Dokuments, sofern <i>Erste Seite anders</i> aktiviert wurde. <sup>a</sup>
wdFootnoteSeparatorStory	12	Fußnotentrennlinie
wdFootnoteContinuationSeparatorStory	13	Fußnoten-Fortsetzungstrennlinie
wdFootnoteContinuationNoticeStory	14	Fußnoten-Fortsetzungshinweis
wdEndnoteSeparatorStory	15	Endnotentrennlinie
wdEndnoteContinuationSeparatorStory	16	Endnoten-Fortsetzungstrennlinie
wdEndnoteContinuationNoticeStory	17	Endnoten-Fortsetzungshinweis

- a. Die Optionen *Gerade/Ungerade anders* bzw. *Erste Seite anders* stehen im Zusammenhang mit den möglichen Einstellungen für die Kopf- und Fußzeilen. Diese Eigenschaften können über den Menübefehl *Datei/Seite einrichten* auf der Registerkarte *Layout* geändert werden.

Bei der *StoryRanges*-Auflistung handelt es sich um eine Auflistung von *Range*-Objekten. Aus diesem Grunde verfügt das Objekt, mit Ausnahme der *NextStoryRange*-Eigenschaft, über keine besonderen Eigenschaften, auf die hier näher eingegangen werden müsste. Die Eigenschaften und Methoden des *Range*-Objekts wurden bereits im Abschnitt »Mit Bereichen arbeiten: Das *Range*-Objekt« in diesem Kapitel detailliert aufgeführt.

Die *StoryRanges*-Auflistung kann auch nicht erweitert werden. Die *Add*-Methode wird nicht unterstützt, denn die möglichen Elemente der Auflistung sind, wie in Tabelle 6.22 dargestellt, begrenzt.

In der Tabelle 6.23 wurden die möglichen Werte für die einzelnen Dokumentkomponenten zusammengestellt, die von der *NextStoryRange*-Eigenschaft zurückgegeben werden. Dies ist entweder *Nothing* oder wiederum ein *Range*-Objekt, welches dem nächsten Element der gleichen Dokumentkomponente entspricht. Als Beispiel veranschaulicht das Listing 6.91 bzw. das Listing 6.92, wie durch den Aufruf der *NextStoryRange*-Methode die nächste primäre Kopfzeile festgelegt wird, sofern das Dokument über mehrere Abschnitte verfügt und die einzelnen Kopfzeilen nicht miteinander verbunden sind.

Next-  
Story-  
Range

**Tabelle 6.23** Zusammenstellung der Elemente, die von der *NextStoryRange*-Methode zurückgegeben werden

Dokumentkomponente	Element, das zurückgegeben wird
wdMainTextStory, wdFootnotesStory, wdEndnotesStory, wdCommentsStory	Gibt immer <b>Nothing</b> zurück, da diese Dokumentkomponente <i>nie</i> über zusätzliche Elemente verfügen kann.
wdTextFrameStory	Das nächste Element von verknüpften Textfeldern oder <b>Nothing</b> .
wdEvenPagesHeaderStory, wdPrimaryHeaderStory, wdEvenPagesFooterStory, wdPrimaryFooterStory, wdFirstPageHeaderStory, wdFirstPageFooterStory	Die Dokumentkomponente des nächsten Abschnitts der gleichen Art, wobei sich »nächster Abschnitt« auf eine nicht verknüpfte Kopf- bzw. Fußzeile bezieht, oder <b>Nothing</b> .

**Listing 6.91** Alle primären Kopfzeilen mittels der *NextStoryRange*-Methode ermitteln

```

Sub Demo_NextStoryRanges()
    Dim doc As Word.Document
    Dim hdr As Word.HeaderFooter
    Dim rng As Word.Range

    Set doc = Documents.Add

    'Abschnitt Zwei und Drei erzeugen
    doc.Range.InsertBreak wdSectionBreakNextPage
    doc.Range.InsertBreak wdSectionBreakNextPage

    'Kopfzeile Abschnitt Eins bis Drei setzen
    Set hdr = doc.Sections(1).Headers(wdHeaderFooterPrimary)
    hdr.Range.Text = "Kopfzeile A"

    'Kopfzeile Abschnitt Drei setzen
    Set hdr = doc.Sections(3).Headers(wdHeaderFooterPrimary)
    hdr.LinkToPrevious = False
    hdr.Range.Text = "Kopfzeile B"

    'Alle Kopfzeilen ausgeben
    Set rng = doc.Sections(1).Headers(wdHeaderFooterPrimary).Range
    MsgBox rng.Text
    While Not (rng.NextStoryRange Is Nothing)
        Set rng = rng.NextStoryRange
        MsgBox rng.Text
    Wend
End Sub

```

**Listing 6.92** (.NET): Die C#-Version. Die *Sections*- sowie *Headers*-Auflistungen werden wie Datenfelder behandelt.

```

private void Demo_NextStoryRanges_CS(Word.Document doc)
{
    //Abschnitt Zwei und Drei erzeugen
    object objSectionNextPage = Word.WdBreakType.wdSectionBreakNextPage;
    doc.Content.InsertBreak(ref objSectionNextPage);
    doc.Content.InsertBreak(ref objSectionNextPage);
}

```

**Listing 6.92** (.NET): Die C#-Version. Die *Sections*- sowie *Headers*-Auflistungen werden wie Datenfelder behandelt. (Fortsetzung)

```
//Kopfzeile Abschnitt Eins bis Drei setzen

wd.HeaderFooter hdr1 = doc.Sections[1].Headers[
    wd.WdHeaderFooterIndex.wdHeaderFooterPrimary];
hdr1.Range.Text = "Kopfzeile A";

//Kopfzeile Abschnitt Drei setzen
wd.HeaderFooter hdr3 = doc.Sections[3].Headers[
    wd.WdHeaderFooterIndex.wdHeaderFooterPrimary];
hdr3.LinkToPrevious = false;
hdr3.Range.Text = "Kopfzeile B";

//Alle Kopfzeilen ausgeben
wd.Range rng = hdr1.Range;
MessageBox.Show(rng.Text);
while (rng.NextStoryRange != null)
{
    rng = rng.NextStoryRange;
    MessageBox.Show(rng.Text);
}
```

Das Beispiel legt ein neues Dokument an und erzeugt zwei zusätzliche Abschnitte. Der primären Kopfzeile im ersten Abschnitt wird ein Text zugewiesen. Die anderen beiden Abschnitte übernehmen die Kopfzeile, da diese standardmäßig mit der vorherigen verknüpft sind. Die Verknüpfung zur vorherigen Kopfzeile wird im dritten Abschnitt aufgelöst und gleichzeitig wird eine eigene Kopfzeile definiert. Anschließend wird mit einer Schleife das StoryRanges-Objekt (wdPrimaryHeaderStory) durchforstet. Die Schleife wird so lange durchlaufen, bis kein nachfolgendes Element in dieser Dokumentkomponente mehr gefunden wird. Beachten Sie dabei, dass zweimal eine Meldung (der Inhalt der Kopfzeile) am Bildschirm ausgegeben wird. Dies deshalb, weil das Dokument zwar über drei Abschnitte, aber nur über zwei unterschiedliche Kopfzeilen verfügt.



Die in diesem Abschnitt dargestellten Beispiele finden Sie in der Beispieldatei *Bsp06\_01a.doc* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap06*.

## Die Seite definieren: Das *PageSetup*-Objekt

Das PageSetup-Objekt ist eigentlich eine Eigenschaft des Section-Objekts. Hier wird für jeden Abschnitt des Dokuments das Layout festgelegt. Dazu gehört unter anderem das

- Festlegen des Papierformat und dessen Ausrichtung,
- Bestimmen der Seitenränder und des Bundstegs,
- Definieren der verschiedenen Kopf- und Fußzeilen sowie deren Abstand zur beschreibbaren Textfläche des Dokuments,
- und Zuweisen der Papierzufuhr für den Ausdruck.

Alle diese Einstellungen sollten bereits während des Anlegens der Dokumentvorlage als Standardeinstellungen definiert werden. Dazu steht das Dialogfeld *Seite einrichten* zur Verfügung. Dieses

kann durch Aufruf des entsprechenden Menübefehls *Datei/Seite einrichten* eingeblendet werden. Somit ist sichergestellt, dass das Grundlayout der einzelnen Dokumente immer gleich ist. Mehr zum Aufbau einer Dokumentvorlage ist in Kapitel 12 beschrieben. Steht dem Entwickler keine entsprechende Dokumentvorlage zur Verfügung, müssen diese Parameter gezwungenermaßen dynamisch erstellt werden.

Innerhalb des Objektmodells kann das PageSetup-Objekt als Eigenschaft zweier unterschiedlicher Objekte angesprochen werden:

```
ActiveDocument.PageSetup
ActiveDocument.Sections(1).PageSetup
```

Solange das aktuelle Dokument nur aus einem Abschnitt besteht, können beide Varianten problemlos genutzt werden, und das Resultat der Programmanweisung wird das gleiche bleiben. Probleme treten auf, wenn im Dokument zusätzliche Abschnittswechsel eingefügt und den einzelnen Einstellungen unterschiedliche Werte zugewiesen werden.

Um die Problematik genauer zu erläutern, soll ein Dokument mit zwei Abschnitten als Basis dienen. Im ersten Abschnitt ist die Ausrichtung auf Hoch- und im zweiten Abschnitt auf Querformat eingestellt:

```
MsgBox ActiveDocument.Sections(1).PageSetup.Orientation 'Ergebnis: 0 = wdOrientPortrait
MsgBox ActiveDocument.Sections(2).PageSetup.Orientation 'Ergebnis: 1 = wdOrientLandscape
```

Der Zugriff auf die Orientation-Eigenschaft erfolgt eindeutig. Dies bedeutet in Bezug auf den entsprechenden Abschnitt, die entsprechenden Werte werden korrekt ausgegeben. Ganz anders sieht das Resultat bei der Verwendung der PageSetup-Eigenschaft des Dokuments aus. Hier ist der Bezug nicht mehr eindeutig:

```
MsgBox ActiveDocument.PageSetup.Orientation 'Ergebnis: 9999999 = wdUndefined
```

Als Resultat wird der Wert »9999999« (wdUndefined) ausgegeben. Dies bedeutet, dass der Wert nicht eindeutig definiert werden kann. Wäre die Ausrichtung in beiden Abschnitten beispielsweise ein Hochformat, würde als Resultat der Wert »0« (wdOrientPortrait) ausgegeben.

### WICHTIG

Wir Autoren empfehlen Ihnen dringend, dass das PageSetup-Objekt nur als Eigenschaft des Section-Objekts angesprochen und verwendet wird. Dies hilft Ihnen Fehler zu vermeiden, die nur in speziellen Dokumentkonstellationen vorkommen und deshalb nicht auf Anhieb reproduziert werden können.

Paper-  
Size  
Page-  
Height  
Page-  
Width

Die PaperSize-Eigenschaft dient zum Festlegen des Papierformats. In Tabelle 6.24 sind die wichtigsten Konstanten für das Papierformat aufgeführt.

Wird die Eigenschaft auf wdPaperCustom gesetzt, müssen zusätzlich die Höhe (PageHeight) und die Breite (PageWidth) des gewünschten Papierformats festgelegt werden. Beide Werte müssen in Punkten eingegeben werden. Zum Umrechnen der Maßeinheit steht unter anderem die CentimetersToPoints-Funktion zur Verfügung.



Tabelle 6.24 Zusammenstellung der wichtigsten Konstanten für das Papierformat

Konstante	Wert	Bedeutung
wdPaperCustom	41	Benutzerdefiniertes Papierformat
wdPaperA5	9	DIN A5
wdPaperA4	7	DIN A4
wdPaperA3	6	DIN A3
wdPaperLetter	2	Letter, amerikanisches Papierformat

Orienta-  
tion

Die Orientation-Eigenschaft dient zum Festlegen der Dokumentausrichtung innerhalb des Abschnitts. Das Dokument kann entweder im Hochformat (wdOrientPortrait) oder im Querformat (wdOrientLandscape) genutzt werden.

Margin-  
Top

Anhand der Eigenschaften MarginTop (Rand oben), MarginBottom (unten), MarginLeft (links) und MarginRight (rechts) wird der effektiv beschreibbare Bereich des Abschnitts festgelegt.

Margin-  
Bottom  
Margin-  
Left  
Margin-  
Right

In Listing 6.93 wird ein spezielles Kärtchen aufgebaut. Der beschreibbare Bereich soll zentriert auf dem Blatt stehen und 15 x 15 cm betragen. Für das neue Dokument werden die benötigten Seitenränder berechnet und zugewiesen. Die Methode ShowTextBoundaries des View-Objekts zeigt die Rändereinstellungen in der Seitenlayout-Ansicht als fein gepunktete Linien, wie in Abbildung 6.57 ersichtlich.

Listing 6.93

Satzspiegel für ein Kärtchen berechnen und zuweisen

```

Sub Demo_KärtchenErstellen()
    Const sngSEITE As Single = 15 'Seitenlänge des Kärtchens

    Dim doc As Word.Document
    Dim sngEinzugH As Single      'Einzug Horizontal
    Dim sngEinzugV As Single      'Einzug Vertikal

    'Neues Dokument erzeugen
    Set doc = Documents.Add

    With doc.Sections(1).PageSetup
        .PaperSize = wdPaperA4
        .Orientation = wdOrientPortrait
    End With

    'Vertikalen bzw. horizontalen Einzug berechnen
    sngEinzugV = (PointsToCentimeters(.PageHeight) - sngSEITE) / 2
    sngEinzugH = (PointsToCentimeters(.PageWidth) - sngSEITE) / 2

    'Seitenränder zuweisen
    .TopMargin = CentimetersToPoints(sngEinzugV)
    .BottomMargin = CentimetersToPoints(sngEinzugV)
    .LeftMargin = CentimetersToPoints(sngEinzugH)
    .RightMargin = CentimetersToPoints(sngEinzugH)
    .Gutter = CentimetersToPoints(0)
End With

'Textbegrenzungen einblenden
doc.ActiveWindow.View.ShowTextBoundaries = True
End Sub

```

**Listing 6.94** (.NET): Die C#-Version

```
private void Demo_KärtchenErstellen_CS(wd.Document doc)
{
    const float SEITE = 15; //Seitenlänge des Kärtchens
    float EinzugH; //Einzug Horizontal
    float EinzugV; //Einzug Vertikal

    wd.PageSetup pageSetup = doc.Sections[1].PageSetup;
    pageSetup.PaperSize = wd.WdPaperSize.wdPaperA4;
    pageSetup.Orientation = wd.WdOrientation.wdOrientPortrait;

    //Vertikalen bzw. horizontalen Einzug berechnen
    EinzugV = (wdApp.PointsToCentimeters(pageSetup.PageHeight) - SEITE) / 2;
    EinzugH = (wdApp.PointsToCentimeters(pageSetup.PageWidth) - SEITE) / 2;

    //Seitenränder zuweisen
    pageSetup.TopMargin = wdApp.CentimetersToPoints(EinzugV);
    pageSetup.BottomMargin = wdApp.CentimetersToPoints(EinzugV);
    pageSetup.LeftMargin = wdApp.CentimetersToPoints(EinzugH);
    pageSetup.RightMargin = wdApp.CentimetersToPoints(EinzugH);
    pageSetup.Gutter = wdApp.CentimetersToPoints(0);

    //Textbegrenzungen einblenden
    doc.ActiveWindow.View.ShowTextBoundaries = true;
    doc.Activate();
}
```

**Gutter** Die Gutter-Eigenschaft dient zum Festlegen des Bundstegs. Der Bundsteg entspricht dem zusätzlichen Abstand, der zum Binden des Dokuments zum Seitenrand hinzugefügt wird (vgl. Abbildung 6.57).

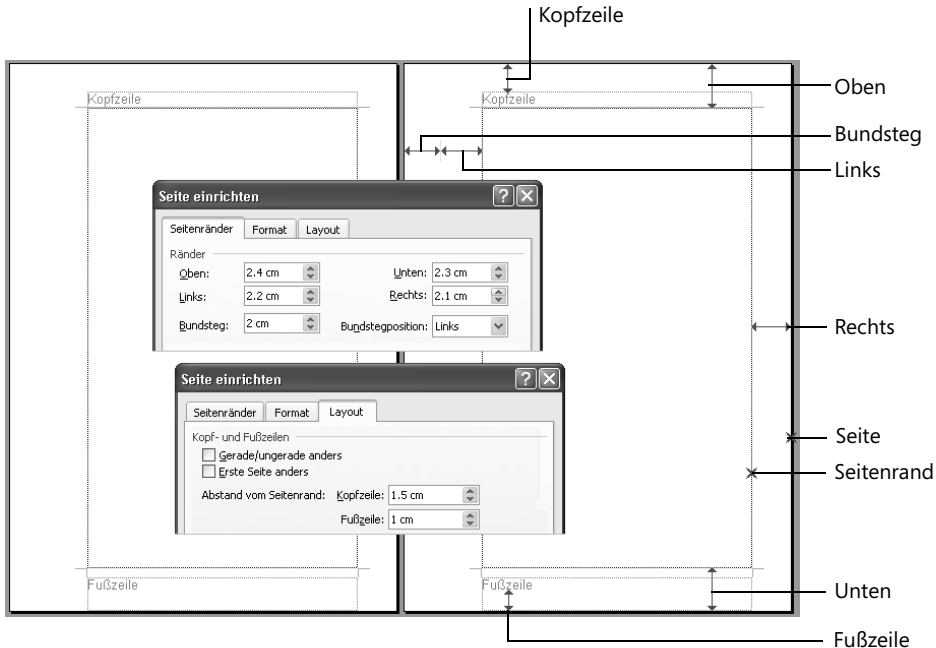
**Header-Distance** Die HeaderDistance- und FooterDistance-Eigenschaften definieren den Abstand von der Seite bis zur Oberkante der Kopfzeile bzw. von der Seite bis Unterkante der Fußzeile.

**Footer-Distance** **HINWEIS** Enthält die Kopfzeile mehr Text, als zwischen der Oberkante der Kopfzeile und dem Seitenrand eingefügt werden kann, wird der beschreibbare Bereich des Abschnitts automatisch verkleinert. Der Wert der MarginTop-Eigenschaft bleibt bestehen. Das gleiche Verhalten gilt analog für die Fußzeile.

Alle Maßeinheiten zum Festlegen des Satzspiegels werden in Punkten festgelegt.

**HINWEIS** In Abbildung 6.57 sind zusätzlich die Positionen der »Seite« und des »Seitenrands« markiert. Diese beiden Ränder können beim Positionieren von Shape-Objekten auch als Ursprungskoordinaten verwendet werden (vgl. Abbildung 6.46).

Abbildg. 6.57 Die Einstellungen des Satzspiegels und deren Auswirkungen



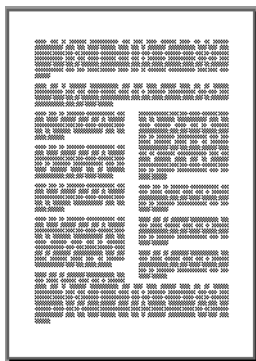
Text-  
Columns

Die TextColumns-Eigenschaft gibt eine TextColumns-Auflistung zurück. Diese Auflistung repräsentiert die Textspalten (»Zeitungsspalten«) des Abschnitts. Wie in Abbildung 6.58 ersichtlich, kann innerhalb der gleichen Seite eines Dokuments eine unterschiedliche Spaltenzahl realisiert werden.

#### HINWEIS

Um eine unterschiedliche Anzahl von Spalten auf der gleichen Seite des Dokuments realisieren zu können, muss mindestens ein fortlaufender Abschnittswechsel (wdSectionBreakContinuous) eingefügt werden.

Abbildg. 6.58 Eine unterschiedliche Anzahl von Spalten kann auf der gleichen Seite zum Einsatz kommen



**HINWEIS** Obwohl es sich bei der `TextColumns`-Eigenschaft um eine Eigenschaft des `PageSetup`-Objekts handelt, muss für das manuelle Ändern der separate Menübefehl *Format/Spalten* angewählt werden.

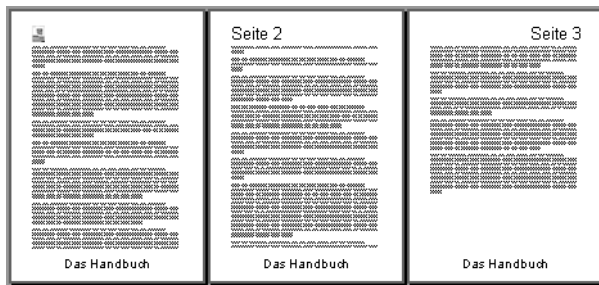
Alle Eigenschaften, die bis jetzt vorgestellt wurden, hatten einen direkten Einfluss auf das Seitenlayout des entsprechenden Abschnitts. Dies ist bei den nachfolgenden Eigenschaften nicht mehr der Fall.

Jeder Abschnitt kann über drei verschiedene Kopf- bzw. Fußzeilen verfügen. So steuert die `DifferentFirstPageHeaderFooter`-Eigenschaft, dass die erste Seite des Abschnitts über eine eigene Kopf- bzw. Fußzeile verfügt.

Die `OddAndEvenPagesHeaderFooter`-Eigenschaft steuert, dass die Seiten mit einer geraden bzw. jene mit einer ungeraden Seitennummer über getrennte Kopf- bzw. Fußzeilenbereiche verfügen.

Werden beide Eigenschaften aktiviert, wird der Inhalt der primären Kopf- bzw. Fußzeile ab der dritten Seite auf allen Seiten mit ungerader Seitennummer dargestellt. Informationen zum Ansteuern der Kopf- und Fußzeilen werden im Abschnitt »Die Seite gestalten: Das `HeaderFooter`-Objekt« in diesem Kapitel dargestellt.

Abbildg. 6.59 Drei unterschiedliche Kopfzeilen können pro Abschnitt erzeugt werden



**HINWEIS** Damit das Dokument wie in Abbildung 6.59 auf allen Seiten über eine einheitliche Fußzeile verfügt, müssen diese manuell oder mittels eines Makros wie in Listing 6.95 synchronisiert werden.

Listing 6.95 Synchronisieren der restlichen Fußzeilen anhand der Fußzeile auf der ersten Seite

```
Sub Demo_FusszeileKopieren()
    Dim ftrF As Word.HeaderFooter 'Erste Seite
    Dim ftrE As Word.HeaderFooter 'Gerade Seiten
    Dim ftrP As Word.HeaderFooter 'Primäre
    Dim i As Integer

    Set ftrF = ActiveDocument.Sections(1).Footers(wdHeaderFooterFirstPage)
    Set ftrE = ActiveDocument.Sections(1).Footers(wdHeaderFooterEvenPages)
    Set ftrP = ActiveDocument.Sections(1).Footers(wdHeaderFooterPrimary)

    With ftrE
        .Range.FormattedText = ftrF.Range.FormattedText
        i = .Range.Paragraphs.Count
    End With
End Sub
```

**Listing 6.95** Synchronisieren der restlichen Fußzeilen anhand der Fußzeile auf der ersten Seite (*Fortsetzung*)

```

        .Range.Paragraphs(i).Range.Delete
    End With

    With ftrP
        .Range.FormattedText = ftrF.Range.FormattedText
        i = .Range.Paragraphs.Count
        .Range.Paragraphs(i).Range.Delete
    End With
End Sub

```

Durch die verwendete Art des direkten Zuweisens des formatierten Textes von einer Fußzeile an eine andere wird diese zwar ersetzt, doch bleibt die letzte Absatzmarke der ehemaligen Fußzeile bestehen. Aus diesem Grunde wird der letzte Absatz nachträglich aus der Fußzeile entfernt.

Die letzten interessanten Eigenschaften des PageSetup-Objekts haben eigentlich nur einen indirekten Einfluss auf das Seitenlayout des Abschnitts. Hier geht es um die Steuerung des Papierschachts für den Ausdruck.

Jedem Abschnitt kann für den Ausdruck eine andere Papierquelle (Papierschacht) zugewiesen werden. Zusätzlich kann innerhalb eines jeden Abschnitts für die erste Seite das zu bedruckende Papier aus einem anderen Papierschacht eingezogen werden, als für die restlichen Seiten.

First-  
PageTray  
Other-  
PagesTray

Zusammen mit einer anderen Kopf- und Fußzeile auf der ersten Seite lassen sich die heute üblichen Anforderungen an das Corporate Design umsetzen. Als Beispiel dient das Listing 6.96. Nur die erste Seite des Dokuments soll das spezielle Briefkopfpapier mit aufgedrucktem Logo verwenden. Die restlichen Seiten werden auf normalem weißem Papier ausgegeben.

**Listing 6.96** Sicherstellen, dass nur die erste Seite des Dokuments auf Briefkopfpapier gedruckt wird

```

Sub Demo_NurErsteSeiteBriefkopfpapier()
    Const intPAPIER_BRIEFKOPF As Integer = wdPrinterUpperBin
    Const intPAPIER_WEISS As Integer = wdPrinterLowerBin
    Dim doc As Word.Document
    Dim sec As Word.Section

    Set doc = Documents.Add
    doc.Range.InsertBreak wdSectionBreakNextPage

    'Alle Abschnitte/alle Seiten verwenden weißes Papier
    For Each sec In doc.Sections
        With sec.PageSetup
            .FirstPageTray = intPAPIER_WEISS
            .OtherPagesTray = intPAPIER_WEISS
        End With
    Next sec

    'Erste Seite im ersten Abschnitt verwendet Briefkopfpapier
    doc.Sections(1).PageSetup.FirstPageTray = intPAPIER_BRIEFKOPF
End Sub

```

Den beiden Eigenschaften kann ein Wert aus der in Tabelle 6.25 zusammengefassten Konstanten oder ein gültiger Integer-Wert zugewiesen werden.

**HINWEIS** Die gültigen Werte für einen bestimmten Druckertreiber müssen im Druckerhandbuch des Geräts nachgeschlagen werden.

Eine zweite Möglichkeit besteht darin, ein Makro aufzuzeichnen. Dazu müsste so lange der Menübefehl *Datei/Seite einrichten* aufgerufen werden, bis alle benötigten Papierfächer einmal für die Papierzufuhr zugewiesen wurden. In einem zweiten Schritt werden die generierten Programmzeilen analysiert und die entsprechenden Werte ausgelesen.

Tabelle 6.25 Zusammenstellung der Konstanten für die Papierschachtssteuerung

WdPaperTray-Konstante	Wert	Bedeutung
wdPrinterDefaultBin	0	Standardpapierschacht gemäß den Einstellungen des Druckertreibers bzw. der Papierschacht, der in den Optionen festgelegt wurde.
wdPrinterOnlyBin	1	Einziger Papierschacht (entspricht Oberer Papierschacht)
wdPrinterUpperBin	1	Oberer Papierschacht
wdPrinterLowerBin	2	Unterer Papierschacht
wdPrinterMiddleBin	3	Mittlerer Papierschacht
wdPrinterManualFeed	4	Manuelle Papierzufuhr, in den meisten Fällen wartet der Drucker, bis das Papier eingelegt und vom Anwender bestätigt wird.
wdPrinterEnvelopeFeed	5	Briefumschlag
wdPrinterManualEnvelopeFeed	6	Manuelle Papierzufuhr für Briefumschlag
wdPrinterAutomaticSheetFeed	7	Einzelblatteinzug für Matrixdrucker
wdPrinterTractorFeed	8	Endlos garnitur für Matrixdrucker
wdPrinterSmallFormatBin	9	Papierschacht mit kleinem Papierformat
wdPrinterLargeFormatBin	10	Papierschacht mit großem Papierformat
wdPrinterLargeCapacityBin	11	Zusätzlicher optionaler Papierschacht
wdPrinterPaperCassette	14	Zusätzlicher Papierschacht
wdPrinterFormSource	15	Automatisch auswählen

**HINWEIS** Die aktuellen Werte für die Papierzufuhr werden im Dokument gespeichert. Dies hat den Vorteil, dass bei einem wiederholten Ausdruck des gleichen Dokuments die gewünschte Papierzufuhr nicht erneut ausgewählt werden muss.

Dieser Vorteil kann jedoch auch ein Nachteil sein. Werden Dokumente intern oder auch extern ausgetauscht und auf einem anderen Drucker ausgegeben, können die gespeicherten Einstellungen dazu führen, dass der Ausdruck auf einen falschen Papierschacht zugreift.

Wird der FirstPageTray- bzw. OtherPagesTray-Eigenschaft ein Wert zugewiesen, der vom aktuellen Druckertreiber nicht unterstützt wird, so wird automatisch auf den Papierschacht »Automatisch auswählen« (wdPrinterFormSource) gewechselt. Dies bedeutet, dass die Standardeinstellungen des Druckertreibers berücksichtigt werden.



Die in diesem Abschnitt dargestellten Beispiele finden Sie in der Beispieldatei *Bsp06\_02a.doc* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap06*.

## Die Seite gestalten: Das *HeaderFooter*-Objekt

Das HeaderFooter-Objekt ist eigentlich ein ganz spezielles Objekt. Die Bezeichnung deutet darauf hin, dass es sich um ein gemeinsames Objekt für die Kopf- und die Fußzeile eines Abschnitts handelt. Dies ist aber nicht der Fall. Wie in Listing 6.97 ersichtlich, verfügt das PageSetup-Objekt über zwei getrennte Eigenschaften. Die Headers- bzw. die Footers-Eigenschaft.

Bei der Kopf- und Fußzeile handelt es sich um einen Bereich des Abschnitts, der auf jeder Seite ausgegeben wird. Diese Bereiche beinhalten pro Abschnitt immer die gleiche Information. Es gilt jedoch zu beachten, dass es sich bei dieser Information nicht grundsätzlich auf jeder Seite auch um den gleichen Inhalt handeln muss. Als Beispiel kann die Seitennummer aufgeführt werden. Die eigentliche Information ist, wie es der Name schon sagt, die Seitennummer. Der Inhalt der Information ist auf jeder Seite unterschiedlich, da sich der Wert von Seite zu Seite ändert.

### HINWEIS

Dynamische Informationen in Kopf- und Fußzeilen können grundsätzlich nur durch die Verwendung von Feldfunktionen erreicht werden. Diejenigen Felder, die regelmäßig in Kopf- und Fußzeilen zur Anwendung kommen, sind in der Tabelle 6.26 kurz zusammengefasst.

**Tabelle 6.26** Besonders geeignete Felder für den Einsatz in Kopf- und Fußzeilen

Feldbezeichnung	Bedeutung
Page	Seitenzahl des Dokuments
NumPages	Gesamtzahl der Seiten im Dokument
SaveDate	Letztes Speicherdatum des Dokuments
PrintDate	Aktuelles Datum beim Ausdrucken des Dokuments
CreateDate	Erstellungsdatum des Dokuments
FileName	Dateiname des Dokuments
StyleRef	Text des Absatzes einer bestimmten Formatvorlage (beispielsweise die Stichworte bei einem Wörterbuch)
If	Bedingte Ausgabe eines Textes (beispielsweise kein Seitenfolgezeichen auf der letzten Seite)
Section	Nummer des aktuellen Abschnitts
SectionPages	Gesamtzahl der Seiten im aktuellen Abschnitt
DocProperty	Wert einer Dokumenteigenschaft
DocVariable	Wert einer Dokumentvariablen

Um eine Kopf- bzw. Fußzeile manuell in das bestehende Dokument aufzunehmen, muss der Menübefehl *Ansicht/Kopf- und Fußzeile* aufgerufen werden. Die Dokumentansicht wechselt während der Bearbeitung der Kopf- bzw. Fußzeile in die Ansicht *Seitenlayout*. Gleichzeitig wird die Symbolleiste *Kopf- und Fußzeile* eingeblendet. Diese Symbolleiste stellt die wichtigsten Funktionen zum Navigieren innerhalb der Kopf- und Fußzeilen zur Verfügung.

**HINWEIS** Ein Beispiel für das Erstellen und Bearbeiten von Kopf- und Fußzeilen finden Sie in Kapitel 23.

Word kennt drei verschiedene Arten von Kopf- bzw. Fußzeilen. Welche dieser drei Arten im Dokument zur Anwendung kommt, wird mit der *DifferentFirstPageHeaderFooter*- und *OddAndEvenPagesHeaderFooter*-Eigenschaft aus dem *PageSetup*-Objekt gesteuert. Mehr zum *PageSetup*-Objekt ist im Abschnitt »Die Seite definieren: Das *PageSetup*-Objekt« in diesem Kapitel erläutert.

**Tabelle 6.27** Zusammenstellung der Konstanten für den Zugriff auf die Kopf- bzw. Fußzeilen

WdHeaderFooter-Enum	Wert	Bedeutung
wdHeaderFooterEvenPages	3	Bei aktivierter <i>OddAndEvenPagesHeaderFooter</i> -Eigenschaft wird die Kopf- bzw. Fußzeile auf allen Seiten mit gerader Seitennummer ausgegeben.
wdHeaderFooterFirstPage	2	Bei aktivierter <i>DifferentFirstPageHeaderFooter</i> -Eigenschaft wird die Kopf- bzw. Fußzeile auf der ersten Seite des Abschnitts ausgegeben.
wdHeaderFooterPrimary	1	Sind beide erwähnten Eigenschaften aktiv, wird die Kopf- bzw. Fußzeile ab der dritten Seite auf allen Seiten mit ungerader Seitennummer ausgegeben.  Werden eine oder auch beide Eigenschaften deaktiviert, wird die primäre Kopf- bzw. Fußzeile auch an deren Stelle dargestellt.

**Listing 6.97** Kopf- und Fußzeile sind getrennte Objekte vom Datentyp *HeaderFooter*

```

Sub Demo_KopfFusszeile()
    Dim doc As Word.Document
    Dim hdr As Word.HeaderFooter
    Dim ftr As Word.HeaderFooter

    Set doc = Documents.Add

    With doc.Sections(1)
        Set hdr = .Headers(wdHeaderFooterPrimary)
        Set ftr = .Footers(wdHeaderFooterPrimary)
    End With

    hdr.Range.Text = "Dies ist die Kopfzeile"
    ftr.Range.Text = "Dies ist die Fußzeile"
End Sub

```

Unabhängig von den gesetzten Eigenschaften des *PageSetup*-Objekts können die Kopf- und Fußzeilen jederzeit über deren *Range*-Objekt angesprochen werden. Somit kann beispielsweise die Kopfzeile für die erste Seite definiert werden, auch wenn die *DifferentFirstPageHeaderFooter*-



Eigenschaft auf False gesetzt ist. Word speichert diese Werte innerhalb der Datei, auch wenn diese im Dokument nicht sichtbar sind.

**HINWEIS**

Kopf- und Fußzeilen sollten immer über deren Range-Objekt bearbeitet werden. Auf die Verwendung des durch den Markorekorder aufgezeichneten Codes sollte unbedingt verzichtet werden. Diese Programmzeilen bauen auf dem Selection-Objekt auf. Dies hat zur Folge, dass während der Laufzeit des Makros ein unschönes Bildschirmflackern sichtbar ist.

Zusätzlich muss berücksichtigt werden, dass nicht sichergestellt werden kann, dass die gewünschte Kopf- bzw. Fußzeile auch tatsächlich bearbeitet wird. Dies kann anhand eines einfachen Beispiels verdeutlicht werden.

Ist die `DifferentFirstPageHeaderFooter`-Eigenschaft auf True gesetzt und die Kopfzeile wird durch Aufruf des Menübefehls *Ansicht/Kopf- und Fußzeile* bearbeitet, dann macht es einen enormen Unterschied, ob sich die Einfügemarke vor dem Start des Makros auf der ersten oder einer anderen Seite befindet.

In Listing 6.98 werden alle drei Kopfzeilen des Abschnitts bearbeitet. Damit die unterschiedlichen Inhalte sichtbar werden, müssen die beiden Eigenschaften manuell gesetzt und zusätzliche Seitenumbrüche eingefügt werden.

**Listing 6.98** Alle drei Kopfzeilenarten als *Range*-Objekt bearbeiten

```
Sub Demo_AlleDreiKopfzeilen()
    Dim doc As Word.Document

    Set doc = Documents.Add

    With doc.Sections(1)
        With .PageSetup
            .DifferentFirstPageHeaderFooter = False
            .OddAndEvenPagesHeaderFooter = False
        End With

        With .Headers
            .Item(wdHeaderFooterPrimary).Range.Text = "Primäre Kopfzeile"
            .Item(wdHeaderFooterEvenPages).Range.Text = "Kopfzeile auf geraden Seiten"
            .Item(wdHeaderFooterFirstPage).Range.Text = "Kopfzeile auf ersten Seiten"
        End With
    End With
End Sub
```

LinkTo-  
Previous

Mit der `LinkToPrevious`-Eigenschaft wird festgelegt, ob die angegebene Kopf- bzw. Fußzeile mit der entsprechenden Kopf- bzw. Fußzeile aus dem vorherigen Abschnitt verknüpft ist.

Wird ein neuer Abschnitt in das Dokument eingefügt, so sind dessen Kopf- und Fußzeilen bereits mit jenen des vorherigen Abschnitts verknüpft. Dieses Verhalten hat den Vorteil, dass innerhalb des Dokuments auf jeder Seite unabhängig vom Abschnitt die gleiche Kopf- bzw. Fußzeile aufgebaut wird.

Die `LinkToPrevious`-Eigenschaft ist individuell gültig. Wie in Listing 6.99 dargestellt, kann beispielsweise die gleiche Fußzeile in allen Abschnitten verwendet werden. Die Kopfzeile wird in jedem Abschnitt anders definiert.

**Listing 6.99** Unterschiedliche Kopfzeilen bei gleich bleibenden Fußzeilen erstellen

```

Sub Demo_KopfFusszeile_ZweiAbschnitte()
    Dim doc As Word.Document

    Set doc = Documents.Add

    With doc.Sections(1)
        .Headers(wdHeaderFooterPrimary).Range.Text = "Kopfzeile Eins"
        .Footers(wdHeaderFooterPrimary).Range.Text = "Fußzeile Eins"
    End With

    doc.Range.InsertBreak wdSectionBreakNextPage

    With doc.Sections(2).Headers(wdHeaderFooterPrimary)
        .LinkToPrevious = False
        .Range.Text = "Kopfzeile Zwei"
    End With
End Sub

```

**Exists** Mit der Exists-Eigenschaft kann festgestellt werden, ob ein spezifisches Objekt der HeaderFooters-Auflistung bereits vorhanden ist. Die beiden nachstehenden Programmzeilen führen die gleiche Prüfung aus.

```

If ActiveDocument.Sections(1).Headers(wdHeaderFooterFirstPage).Exists Then
If ActiveDocument.Sections(1).PageSetup.DifferentFirstPageHeaderFooter Then

```

Bei dieser Prüfung wird nur getestet, ob das Objekt vorhanden ist. Dies entspricht einer Prüfung des Status des entsprechenden Kontrollkästchens im Dialogfeld *Seite einrichten*. Eine Kontrolle, ob bereits ein Text in die Kopfzeile eingetragen wurde, findet auf diese Weise nicht statt.

**Page-Numbers** Die PageNumbers-Eigenschaft gibt eine PageNumbers-Auflistung zurück. Diese Auflistung entspricht allen Seitenzahlfeldern, die in der entsprechenden Kopf- bzw. Fußzeile vorhanden sind.

Aus der Sicht der Autoren kann auf das Einfügen eines PageNumber-Objekts ins Dokument verzichtet werden, denn das Einfügen einer Seitennummer mit dem Menübefehl *Einfügen/Seitenzahlen* erzeugt die Seitennummer innerhalb eines Positionsrahmens. Diesen Positionsrahmen mittels VBA zu bearbeiten, ist weit aufwändiger als das Einfügen und Bearbeiten eines Page-Felds.

Trotzdem verfügt das PageNumber-Objekt über zwei besonders interessante Aspekte, deren Nutzen kurz aufgezeigt werden soll.

- Das Neustarten der Seitennummerierung in einem bestimmten Abschnitt wie dies in Listing 6.100 dargestellt wird.
- Zum Formatieren des *Page*-Felds (Zahlenformat, Kapitelnummer usw.) wie dies in Listing 6.102 dargestellt wird.

In der Benutzerschnittstelle wird das in Abbildung 6.60 abgebildete Dialogfeld durch Anwählen des Menübefehls *Einfügen/Seitenzahlen* und Betätigen der Schaltfläche *Format* erreicht. Beim Verlassen der Dialogfelder ist es wichtig, dass Sie im Dialogfeld *Seitenzahlen* nicht auf *OK*, sondern auf *Schließen* klicken. Sonst wird eine Seitenzahl in einen Positionsrahmen zusätzlich eingefügt.

Muss die Seitennummerierung in einem Abschnitt bei einer bestimmten Zahl beginnen, so muss der entsprechende Wert der StartingNumber-Eigenschaft zugewiesen werden.

**HINWEIS** Damit die Seitennummerierung tatsächlich mit der gewünschten Zahl beginnt, muss beim entsprechenden Abschnitt zusätzlich die `RestartNumberingAtSection`-Eigenschaft auf `True` gesetzt werden.

**Listing 6.100** Festlegen des Startwerts für die Seitennummer im zweiten Abschnitt

```
Sub Demo_Seitenzahl_NeuStarten()
    Dim doc As Word.Document
    Dim rng As Word.Range

    Set doc = Documents.Add
    Set rng = doc.Sections(1).Headers(wdHeaderFooterPrimary).Range

    rng.Fields.Add Range:=rng, Type:=wdFieldPage

    doc.Range.InsertBreak wdSectionBreakNextPage

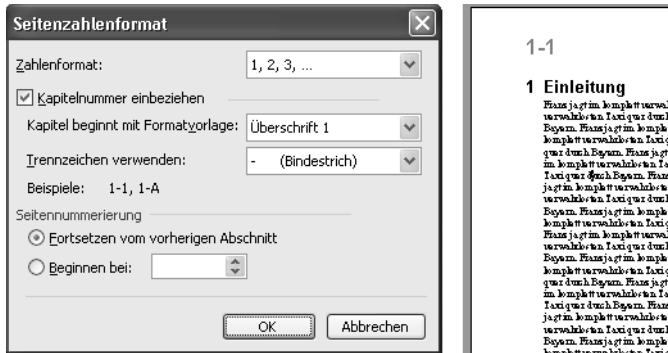
    With ActiveDocument.Sections(2).Headers(wdHeaderFooterPrimary).PageNumbers
        .RestartNumberingAtSection = True
        .StartingNumber = 100
    End With
End Sub
```

**Listing 6.101** (.NET): Die C#-Version

```
private void Demo_Seitenzahl_NeuStarten_CS()
{
    object objMissing = System.Reflection.Missing.Value;
    wd.Document doc = wdApp.Documents.Add(ref objMissing, ref objMissing,
        ref objMissing, ref objMissing);
    doc.ActiveWindow.View.DisplayPageBoundaries = true;
    wd.WdHeaderFooterIndex HFPrimary = wd.WdHeaderFooterIndex.wdHeaderFooterPrimary;
    wd.Range rng = doc.Sections[1].Headers[HFPrimary].Range;
    object objFieldType = (object) wd.WdFieldType.wdFieldPage;
    object objFalse = false;
    rng.Fields.Add(rng, ref objFieldType, ref objMissing, ref objFalse);
    object objBreakNextPage = (object) wd.WdBreakType.wdSectionBreakNextPage;
    doc.Content.InsertBreak(ref objBreakNextPage);
    wd.HeaderFooter HFAbschnitt2 = doc.Sections[2].Headers[HFPrimary];
    HFAbschnitt2 .PageNumbers.RestartNumberingAtSection = true;
    HFAbschnitt2 .PageNumbers.StartingNumber = 100;
}
```

In Abbildung 6.60 sind die verschiedenen Möglichkeiten zum Formatieren der Seitennummer dargestellt. Das Resultat der dargestellten Einstellungen ist ebenfalls ersichtlich. Die gleichen Einstellungen werden anschließend im Listing 6.102 umgesetzt.

Abbildg. 6.60 Legen Sie die Eigenschaften für das Seitenzahlenformat fest



Listing 6.102 Festlegen des Formats für die Seitennummer

```
Sub Demo_SeitennummerFormatieren()
    Dim doc As Word.Document
    Dim hdr As Word.HeaderFooter

    Set doc = Documents.Add
    Set hdr = doc.Sections(1).Headers(wdHeaderFooterPrimary)

    With hdr.PageNumbers
        .NumberStyle = wdPageNumberStyleArabic      '1. 2. 3.
        .IncludeChapterNumber = True                 'Kapitelnummer verwenden
        .HeadingLevelForChapter = 0                   'Überschrift 1
        .ChapterPageSeparator = wdSeparatorHyphen    'Bindestrich einfügen
        .RestartNumberingAtSection = False
        .StartingNumber = 0

        .Add PageNumberAlignment:=wdAlignPageNumberLeft, FirstPage:=True
    End With
End Sub
```

#### HINWEIS

Die zugewiesenen Eigenschaften des PageNumbers-Objekts werden direkt im Page-Feld umgesetzt. Das Format der Seitennummer kann pro Abschnitt geändert werden, da das PageNumbers-Objekt eine Eigenschaft des HeaderFooter-Objekts ist.

IsHeader  
IsFooter

Die IsHeader- und IsFooter-Eigenschaft als die beiden letzten interessanten Eigenschaften des HeaderFooter-Objekts müssten eigentlich gar nicht aufgeführt werden. Mit diesen Eigenschaften kann die Position der Einfügemarke ermittelt werden. So kann überprüft werden, ob sich die Einfügemarke innerhalb der Kopf- bzw. der Fußzeile befindet.

Diese beiden Eigenschaften können nur mit dem Selection-Objekt verwendet werden. Wie bereits im Abschnitt »Die gegenwärtige Markierung: Selection und ähnliche Objekte« in diesem Kapitel erläutert wurde, raten wir Ihnen jedoch vom Abreiten mit dem Selection-Objekt ab. Aus diesem Grunde werden diese beiden Eigenschaften nicht näher beschrieben.



Die dargestellten Beispiele finden Sie in der Beispieldatei *Bsp06\_02a.doc* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap06*.

## Lange Dokumente

Word dient vielen Bedürfnissen, die von kurzen Memos über Briefe und Berichte bis zu Büchern und Handbücher reichen. In den vorangehenden Abschnitten wurden Objekte vorgestellt, die in allen Dokumentarten benutzt werden können. In diesem Abschnitt werden wir einige Aspekte und Objekte diskutieren, die primär mit der Dokumentverwaltung und der Erstellung von langen Dokumenten zu tun haben.

Mit der Einführung eines XML-Dateiformats in Word 2003 (und dem angekündigten Ausbau dieses Formats in Word 12 zum Standardformat) verliert die über das Objektmodell automatisierte Erstellung und Verwaltung von langen Dokumenten an Bedeutung. Die Aufgabe ist allgemein schneller mit XML zu erreichen, mit verringerter Gefahr der Dokumentbeschädigung. Eine Einführung in die XML-Funktionalität in Word 2003 finden Sie in Teil VI dieses Buchs.

Dank schnelleren, stabileren Rechnern und Betriebssystemen sind in den letzten Jahren viele der Beschränkungen für die Arbeit mit langen Dokumenten entfallen. Word-Dokumente können, vom Blickwinkel der Word-Anwendung aus betrachtet, gut und gern mehr als eintausend Seiten umfassen. Die Frage ist jetzt eher, ob es für Autoren oder Anwender wünschenswert ist, den gesamten Text eines Werks in einem einzigen Dokument zu verwalten. Einige Beispiele für eine gegenteiligen Entscheidung:

- Word unterstützt die gleichzeitige Bearbeitung eines einzelnen Dokuments nicht. Arbeiten mehrere Personen am gleichen Werk, ist es sinnvoll, dies in mehrere Dokumente aufzuteilen.
- In manchen Werken bleiben einige Teile statisch, während andere häufiger bearbeitet oder ausgetauscht werden. Es ist vorteilhaft, wenn der dynamische Text getrennt verwaltet wird.
- Viele Dokumentationen unserer globalen Welt müssen mehrsprachig vorliegen. Es kann wünschenswert sein, die verschiedenen Sprachversionen getrennt zu speichern, und diese gleichzeitig nebeneinander im gleichen Dokument zu sehen oder auszudrucken.

## Zentraldokumente

Für die Bewältigung solcher Aufgaben stellt Word zwei Funktionalitäten zur Verfügung: Zentral- und Filialdokumente sowie die Dokumentenverknüpfung (*IncludeText*-Feldfunktion). Die Zentraldokument-Funktionalität ermöglicht das Einfügen von »Filialdokumenten« mit automatischer Erhaltung der Seitenlayout-Eigenschaften jedes einzelnen Dokuments. Dies wird erreicht durch das Einfügen von zwei Abschnittswechsels zwischen jedem Dokument, zusammen mit dem Ausschalten der Verknüpfungen zwischen Kopf- und Fußzeilen.

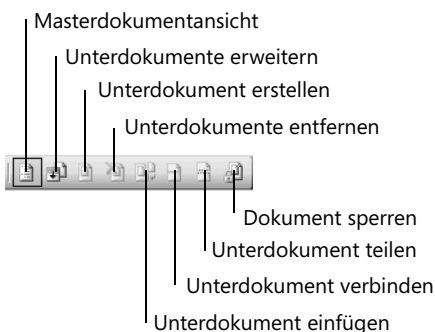
Allgemein raten Word-Kenner von Zentraldokumenten ab. Unsachgemäß eingesetzt, führen sie zu Dokumentbeschädigungen und kostspieligem Daten- und Zeitverlust. Vorausgesetzt einer korrekten Handhabung ist es jedoch sinnvoll, sich ihrer Vorteile zu bedienen. Wenn Sie sich für eine Zentraldokument-Lösung entscheiden, achten Sie auf die folgenden Punkte:

- Die Zentral- und Filialdokumente sollten alle von der gleichen Dokumentvorlage erstellt werden, die ihrerseits von einer neuen Kopie der *Normal.dot* stammen soll. Diese liefert alle Grundeinstellungen für alle Dokumente, die Formatvorlagen inbegriffen.
- Da sich Formatvorlagen der Definition von Formatvorlagen gleichen Namens im Zieldokument annehmen, müssen abweichende Formatierungen in einem Filialdokument mit einer eigens dafür in diesem Dokument erstellten Formatvorlage erfolgen, die in keinem anderen der Dokumente vorhanden ist.
- Filialdokumente sollen als einzelne Dateien nur bei geschlossenem Zentraldokument geöffnet und bearbeitet werden.
- Filialdokumente sollen niemals innerhalb des Zentraldokuments verschoben werden. Stattdessen soll der Bereich (samt Abschnittswechsel) gelöscht und die Datei neu eingefügt werden.
- Bevor mit einem Zentraldokument gearbeitet wird, sind Sicherungskopien aller Filialdokumente zu erstellen.
- Filialdokumente sollen niemals im Zentraldokument bearbeitet werden. Das Zentraldokument ist einzig da, um Dokument-übergreifende Elemente wie Listenummerierungen, Verzeichnisse, Verweise und Indexes zur Verfügung zu stellen.
- Querverweise sowie Inhaltsverzeichnisse und Indexes werden im Zentraldokument erstellt und verwaltet, und haben nur in diesem Umfeld Gültigkeit (sind im einzelnen Filialdokument bedeutungslos).

Zentraldokumente werden in der Gliederungsansicht mit den Werkzeugen der »Masterdokumentansicht« der Symbolleiste *Gliederung* (Abbildung 6.61) verwaltet. Filialdokumente können aus dem Text des Zentraldokuments oder durch Einfügen eines gespeicherten Dokuments erstellt werden. Allgemeine Angaben liefert die Word-Hilfdatei. In diesem Abschnitt werden wir lediglich einige wichtige Punkte kurz vorstellen.

Ein Filialdokument kann in zwei geteilt werden; mehrere Filialdokumente dürfen zu einem vereint werden. Die Schaltfläche *Unterdokument entfernen* löscht nicht den Text eines Filialdokuments aus dem Zentraldokument, sondern entfernt lediglich die Filialdokumentstrukturen (der Text wird ins Zentraldokument integriert).

Abbildung. 6.61 Die Symbolleiste für die Arbeit mit Zentral- und Filialdokumenten



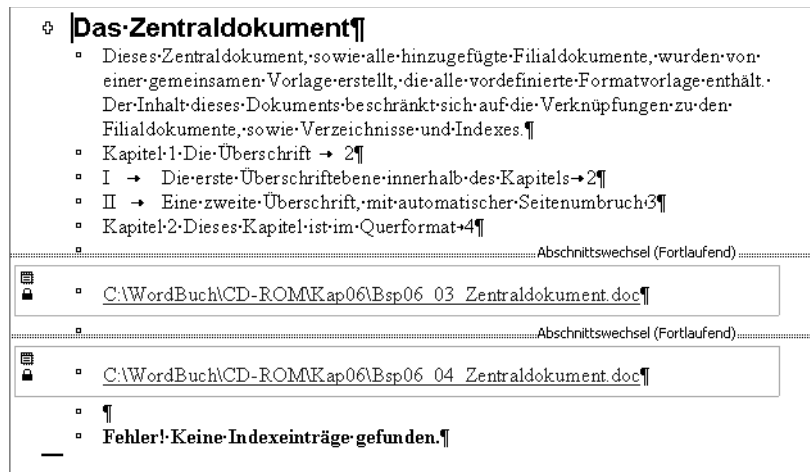
**HINWEIS**

Den Übersetzern der Benutzerschnittstelle der neueren Word-Versionen war es offensichtlich nicht klar, dass die Funktionalität seit Jahren Zentral- und Filialdokumente heißt. Sie haben etwas willkürlich, an die englische Terminologie anlehnd, die Symbolleistenschaltflächen mit »Masterdokument« und »Unterdokument« beschriftet. Da in der Word-Hilfe immer noch »Zentraldokument« und »Filialdokument« gebraucht werden, behalten wir diese Ausdrücke bei.

Wird ein Filialdokument auf Grund eines Bereichs im Zentraldokument erstellt, muss der erste Absatz dieses Bereichs mit einer Word-Überschriftenformatvorlage formatiert sein.

Beim Öffnen eines Zentraldokuments erscheinen alle Filialdokumente als Hyperlinks mit absolutem Pfad, wie in Abbildung 6.62 ersichtlich. In Word 2003 werden die Pfadangaben jedoch als relative Pfade verwaltet. In früheren Versionen ist dies nicht immer der Fall. Diese Pfadangaben können nicht geändert oder bearbeitet werden, auch nicht über das Objektmodell.

Abbildg. 6.62 Noch nicht erweiterte Filialdokumente in der Gliederungsansicht



Die Beispieldatei *Bsp06\_01\_Zentraldokument.dot* ist eine Vorlage für das Beispiel in Abbildung 6.62. Die Beispieldatei *Bsp06\_02\_Zentraldokument.doc* ist das eigentliche Zentraldokument; *Bsp06\_03\_Zentraldokument.doc* sowie *Bsp06\_04\_Zentraldokument.doc* die Filialdokumente. Sie finden alle auf der CD-ROM zum Buch im Ordner `\Beispiele\Kap06`.

Beim Erstellen eines Filialdokuments aus einem Bereich des Zentraldokuments muss das neue Filialdokument in ein eigenes Fenster geöffnet werden. Dort wird es über *Datei/Speichern unter* unter einem eigenen Dateinamen gespeichert. Wird dies nicht getan, speichert Word beim Schließen des Zentraldokuments das Filialdokument automatisch in den gleichen Ordern. Dabei wird keine Aufforderung zur Eingabe eines Dateinamens eingeblendet.

Im Objektmodell wird mit den Eigenschaften `IsMasterDocument` und `IsSubDocument` festgestellt, ob ein Dokument ein Zentral- bzw. Filialdokument ist. Die beiden Eigenschaften sind nur lesbar.

Ist ein Dokument ein Zentraldokument, werden seine Filialdokumente über die `SubDocuments`-Eigenschaft angesprochen. Wichtige Eigenschaften und Methoden sind `AddFromFile` (Filialdokument aus

SubDocu-  
ment

einer Datei hinzufügen), AddFromRange (Filialdokument aus einem Bereich im Zentraldokument erstellen), Count, Delete (die Filialdokumentstrukturen entfernen), Expanded (der Text des Filialdokuments ist im Zentraldokument sichtbar) sowie Merge (mehrere Filialdokumente in eines zusammenführen).

Das SubDocument-Objekt seinerseits hat Eigenschaften und Methoden wie HasFile (ein aus einem Bereich erstelltes Filialdokument wurde bereits als Dokument gespeichert), Level (die oberste Überschriftsebene eines Filialdokuments), Locked (gibt an, ob das Filialdokument gesperrt ist), Open (öffnet das Filialdokument in einem eigenen Word-Fenster), Path (die Pfadangabe), Range (der Bereich), und Split (ein Filialdokument in zwei Filialdokumente aufteilen).

Das Listing 6.103 zeigt, wie aus einem gewöhnlichen Dokument durch Einfügen von Dateien als Filialdokumente ein Zentraldokument wird. Bitte beachten Sie, dass

- das Dokument in der Zentraldokumentansicht sein muss,
- diese Funktionalität auf der gegenwärtigen Markierung im Dokument basiert: das Filialdokument wird an dieser Stelle eingefügt, und ersetzt eventuell markierten Text.

Falls das Filialdokument auf einer anderen Vorlage als das Zentraldokument basiert, blendet Word eine entsprechende Meldung ein, die nicht unterdrückt werden kann.

**Listing 6.103** Alle *doc*-Dateien eines Ordners, die mit dem Zeichen »Test« anfangen, werden als Filialdokumente eingefügt

```
Sub FilialDokumentEinbinden()
    Dim strDateiname As String
    Dim strPfad As String
    Dim docZentral As Word.Document

    Application.DisplayAlerts = wdAlertsNone
    strPfad = "C:\Test\"
    strDateiname = Dir$(strPfad & "Test*.doc")
    Set docZentral = ActiveDocument
    docZentral.ActiveWindow.View = wdMasterView
    Selection.Find.ClearFormatting
    Selection.Find.Execute FindText:="Filialdokumente hier einfügen."
    Do While Not strDateiname = ""
        docZentral.Subdocuments.AddFromFile Name:=strPfad & strDateiname
        strDateiname = Dir
    Loop
    Application.DisplayAlerts = wdAlertsAll
End Sub
```



Die Beispieldatei *Bsp06\_05\_Zentraldokument.doc* mit der Prozedur finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap06*.

## Dokumente verknüpfen

Die alternative Methode ist, Dokumente über *Einfügen/Datei* mit einer Verknüpfung in ein »Zentraldokument« einzufügen. Im Gegensatz zur Zentraldokument-Funktionalität gehen die Seitenlayouteigenschaften sowie Kopf- und Fußzeilen des eingefügten Dokuments verloren, es sei denn, diese werden durch Abschnittswechsel im Quelldokument »geschützt«. Meist bedeutet dies, dass Abschnittswechsel am Dokumentanfang und -ende vorhanden sind, die Informationen speichern.



Da das Einfügen von Dokumenten in der Regel ohne zusätzliche Abschnittwechsel vonstatten geht, neigen solche »Zentraldokumente« weniger zu Dokumentbeschädigung.

Solche »Zentraldokumente« öffnen immer mit dem gesamten Text sichtbar. Zudem sind, da die Verknüpfungen über *IncludeText*-Feldfunktionen verwaltet werden, relative Pfadangaben möglich, und die Pfadangaben können problemlos angepasst werden (siehe auch die Erläuterung »Verknüpfungen, Tabellen und Berechnungen« im Abschnitt »Feldfunktionen« in diesem Kapitel).

Es ist möglich, den im »Zentraldokument« verknüpften Text im »Zentraldokument« zu bearbeiten. Änderungen werden mit der Tastenkombination **Strg** + **↕** + **F7** zurück ans Quelldokument geschickt. Im Objektmodell entspricht dies der Methode *UpdateSource* des *Field*-Objekts.

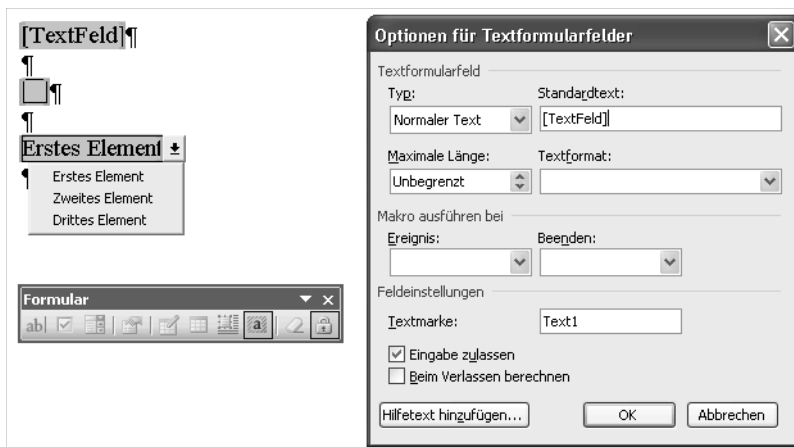
## Formulare: Das *FormField*-Objekt


Viele Dokumentarten wie Memos und Faxnachrichten muss der Anwender wiederholt erstellen. Um den Arbeitsvorgang zu beschleunigen und zu vereinfachen, werden hierfür Dokumentvorlagen bereitgestellt. Diese können sogar mit vorhandenen Daten über den Seriendruck oder eine automatisierte Lösung bestückt werden, um noch effektiver zu arbeiten. Eines haben diese Methoden gemeinsam: der Anwender könnte das Dokumentlayout oder -inhalt bearbeiten und ändern. Manchmal ist dies jedoch nicht erwünscht.

Um diesem Bedürfnis entgegenzukommen, bietet Word den Dokumentschutz und Formularfelder an. In der Benutzerschnittstelle befindet sich diese Funktionalität in der *Formular*-Symbolleiste sowie unter dem Menübefehl *Extras/Dokument schützen*.

Es gibt drei Arten von Formularfeldern: Text, Kontrollkästchen und Dropdown-Liste (siehe Abbildung 6.63). Ihnen können Namen zugewiesen werden, die gleichzeitig als Textmarken dienen. Es ist auch möglich, dafür einen Hilfetext zu definieren, der in der Statusleiste oder durch Drücken von **F1** eingeblendet wird. Als Automatisierungsschnittstelle bieten sie Ereignisse beim Eintreten und beim Verlassen des Feldes. Formularfelder können auch begrenzt formatiert und für die Benutzereingabe gesperrt werden. Alle diese Einstellungen befinden sich im Dialogfeld *Optionen für ...*, das per Doppelklick auf ein Formularfeld erscheint.

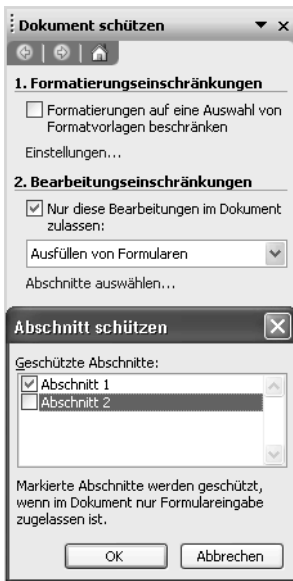
Abbildg. 6.63 Formularfelder, die *Formular*-Symbolleiste sowie ein *Optionen*-Dialogfeld



Formularfelder können nur als solche benutzt werden, wenn das Dokument als ein Formular geschützt ist. Nur dann kann durch Drücken der -Taste oder mit der Maus zwischen den Formularfeldern navigiert werden. Um Teile eines Dokuments für die normale Bearbeitung freizustellen, können Abschnittswchsel eingefügt und bestimmte Abschnitte als ungeschützt deklariert werden (Abbildung 6.64).

**HINWEIS** Bestimmte Funktionalitäten bleiben auch in ungeschützten Abschnitten eines als Formular geschützten Dokuments gesperrt, wie etwa Kopf- und Fußzeilen sowie die Zeichnungsebene. Eine Übersicht der gesperrten Funktionalität steht im Knowledge Base-Artikel »WD: Some Menu Commands Unavailable (Document Protection)« unter <http://support.microsoft.com/default.aspx?scid=kb;en-us;105697> beschrieben.

**Abbildg. 6.64** Der Aufgabenbereich für den Dokumentschutz in Word 2002 und 2003. Der Abschnitt 2 bleibt ungeschützt.



**HINWEIS** Word unterstützt auch ActiveX-Steuerelemente für die Informationseingabe, welche gegenüber Formularfeldern Vor- und Nachteile haben. Mehr Informationen dazu finden Sie in Kapitel 11.

Aus Sicht des Entwicklers können Formularfelder als Alternative zu Textmarken betrachtet werden: Es ist möglich, Daten in die Felder zu schreiben sowie diese auszulesen. Formularfelder haben den weiteren Vorteil, dass sie vom Anwender nicht versehentlich gelöscht werden können, wie dies häufig bei Textmarken der Fall ist. Im Objektmodell wird ein Formularfeld mit dem `Formfield`-Objekt dargestellt. Ein Formularfeld kann mit dem Indexwert im Dokument oder über seinen Namen angesprochen werden.

**ACHTUNG** Standardmäßig wird ein Formularfeld beim Einfügen mit einem Namen wie »Text1«, »Text2«, usw. versehen. Da diese Namen gleichzeitig Textmarken sind, müssen die Bezeichner gezwungenermaßen einmalig sein. Wird ein Formularfeld kopiert und ins gleiche Dokument eingefügt, geht ein eventuell identischer Name entweder des Originals oder der Kopie verloren. Achten Sie also darauf, dass alle Formularfelder, die Ihr Code bearbeiten muss, gültige Namen haben. Falls Sie Formularfeldnamen während des Code-Ablaufs zuweisen müssen, muss das Dokument im ungeschützten Zustand sein.

Formular-  
feldwerte

Um den Textinhalt eines Textfeldes zu lesen oder zu schreiben wird die Result-Eigenschaft benutzt (im Gegensatz zur Result-Eigenschaft einer Feldfunktion gibt ein Formularfeld Result eine Zeichenkette und keinen Bereich zurück):

```
doc.FormFields("FormfeldName").Result = "Textinhalt"
strFormularfeldInhalt = doc.FormFields("FormularfeldName").Result
```

Auch ein Dropdown-Feld unterstützt die Result-Eigenschaft, die sich auf den sichtbaren Text bezieht. Zudem kann über die Dropdown.Value-Eigenschaft der Indexwert gelesen oder geschrieben werden:

```
doc.FormFields("Dropdown1").Dropdown.Value = 2
lGewählterIndex = doc.FormFields("Dropdown1").Dropdown.Value
```

Der Zustand eines Kontrollkästchens wird ebenfalls über die Result-Eigenschaft abgefragt bzw. festgelegt:

```
doc.FormFields("Kontrollkästchen1").Result = 1 'Aktiviert. 0 = Nicht aktiviert.
lKontrollkästchenWert = doc.FormFields("Kontrollkästchen1").Result
```

Wenn Sie sicherstellen wollen, dass ein Formularfeld ein Kontrollkästchen oder Dropdownfeld ist, können Sie entweder die Type- oder die Valid- Eigenschaft prüfen. Erstere gibt ein WdFieldType zurück. Die zweite wird wie folgt verwendet:

```
If Formfield.CheckBox.Valid Then 'Es ist ein Kontrollkästchen
If Formfield.Dropdown.Valid Then 'Es ist ein Dropdownfeld
```

Das Listing 6.104 bzw. das Listing 6.105 veranschaulicht die Verwendung der Eigenschaft in einer Funktion, die zurückgibt, ob der Inhalt eines Bereichs ein Kontrollkästchen ist. Gleichzeitig zeigt sie, wie der Name eines Formularfelds ermittelt wird: über das zugehörige Bookmark-Objekt.

**Listing 6.104** Prüfen, ob das markierte Formfeld ein Kontrollkästchen ist

```
Sub Test()
    Dim rng As Word.Range

    Set rng = Selection.Range
    MsgBox IstKontrollkästchen(rng)
End Sub
```

**Listing 6.104** Prüfen, ob das markierte Formfeld ein Kontrollkästchen ist (*Fortsetzung*)

```
Function IstKontrollkästchen(rng As Word.Range) As Boolean
    Dim strFeldName As String
    Dim ffld As Word.FormField

    IstKontrollkästchen = False
    'Prüfen, ob eine Textmarke vorhanden ist.
    If rng.Bookmarks.Count >= 1 Then
        'Wenn ja, enthält sie ein Formularfeld?
        If rng.Bookmarks(1).Range.FormFields.Count = 1 Then
            'Dann ist der Textmarkenname der Feldname
            strFeldName = rng.Bookmarks(1).Name
            Set ffld = rng.Document.FormFields(strFeldName)
            If ffld.CheckBox.Valid Then IstKontrollkästchen = True
        End If
    End If
End Function
```

**Listing 6.105** Listing 6.105 (.NET): Die C#-Version. Das *FormField*-Objekt verlangt *get\_Item*

```
private bool IstKontrollkästchen(wd.Range rng)
{
    bool test = false;
    //Prüfen, ob eine Textmarke vorhanden ist.
    if(rng.Bookmarks.Count >= 1)
    {
        //Wenn ja, enthält sie ein Formularfeld?
        object objIndex1 = (object) 1;
        if (rng.Bookmarks.get_Item(ref objIndex1).Range.FormFields.Count == 1)
        {
            //Dann ist der Textmarkenname der Feldname
            string fieldName = rng.Bookmarks.get_Item(ref objIndex1).Name;
            object objFieldName = (object) fieldName;
            wd.FormField ffld = rng.Document.FormFields.get_Item(ref objFieldName);
            if (ffld.CheckBox.Valid)
            {test = true;}
        }
    }
    return test;
}
```



Die Beispieldatei *Bsp06\_01\_Form.doc* mit der Prozedur finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap06*.

Doku-  
ment  
schützen

Wie oben erwähnt, sind die Bearbeitungsmöglichkeiten eingeschränkt, wenn ein Dokument als ein Formular geschützt ist. Möchten Sie dem Anwender gewisse Befehle wie die Zeichenformatierung trotzdem zur Verfügung stellen, ist eine Automatisierungslösung notwendig. Diese ermittelt die gewünschte Handlung, hebt den Dokumentschutz auf, führt die Handlung aus und stellt anschließend den Dokumentschutz wieder her.

Um den Dokumentschutz aufzuheben, wird die Methode *Unprotect* eingesetzt, die ein optionales Argument *Password* akzeptiert, falls der Dokumentschutz mit einem Kennwort gesichert wurde:

```
doc.Unprotect "Kennwort"
```

in C#

```
object objKennwort = (object) "Kennwort"
doc.Unprotect(ref objKennwort);
```

Die Methode `Protect`, um den Dokumentschutz zu aktivieren, hat mehrere Argumente. Die Syntax in Word 2003 lautet:

```
Protect(Type, NoReset, Password, UseIRM, EnforceStyleLock)
```

`Type` legt die Art Dokumentschutz fest und erwartet einen `WdProtectionType`-Konstantwert. `wdAllowOnlyComments` (nur Kommentare), `wdAllowOnlyFormFields` (nur Formulareingabe), `wdAllowOnlyReading` (nur lesbar), `wdAllowOnlyRevisions` (nur Änderungen verfolgen) sowie `wdNoProtection` (kein Schutz). `wdAllowOnlyReading` wird nur von Word 2003 unterstützt, und ist das Gegenstück zur Auswahl »Keine Änderungen (schreibgeschützt)« in der Liste *Bearbeitungseinschränkungen* des Aufgabenbereichs *Dokument schützen* (Abbildung 6.64).

`NoReset` bestimmt, ob der Inhalt der Formularfelder zurückgesetzt werden soll. Meistens setzt man es auf `True`, um die Benutzereingaben zu behalten.

Mit `Password` wird ein Kennwort festgelegt, so dass der Anwender nicht ohne weiteres den Schutz aufheben kann.

#### WICHTIG

Der Formularschutz ist einfach zu umgehen. Mit dem Befehl *Einfügen/Datei* kann ein Formular, auch ein per Kennwort geschütztes, jeder Zeit ungeschützt in einem anderen Dokument »geöffnet« werden.

Das Argument `UseIRM` ist nur in Word 2003 vorhanden und weist Word an, das »Information Rights Management« einzuschalten. (Mehr über IRM erfahren Sie im TechNet-Artikel »Microsoft Office 2003 – Informationen schützen mit den Diensten für die Windows-Rechteverwaltung und der Verwaltung von Informationsrechten« unter <http://www.microsoft.com/germany/technet/datenbank/articles/600336.mspx>)

Auch `EnforceStyleLock` ist Word 2003-spezifisch und schaltet die Formatierungseinschränkungen ein (siehe auch den Abschnitt »Formatieren mit Stil: Das *Style*-Objekt« in diesem Kapitel). Wenn Sie dieses Argument auf `True` setzen, wird `Type` meist auf `wdNoProtection` festgelegt.

Um einen Abschnitt aus dem Formularschutz auszuschließen, muss die `ProtectedForForms`-Eigenschaft des `Section`-Objekts auf `False` gesetzt werden. Beim Schützen des Dokuments wird diese Einstellung dann berücksichtigt:

```
doc.Sections(2).ProtectedForForms = False
```

Als Beispiel zeigt das Listing 6.106 bzw. das Listing 6.107, wie die gegenwärtige Markierung innerhalb eines Textformularfelds fett formatiert wird. Beachten Sie, wie mit der `ProtectionType`-Eigenschaft geprüft wird, ob der Dokumentschutz aktiviert ist, da die `Unprotect`-Methode einen Laufzeitfehler hervorruft, falls das Dokument in einem ungeschütztem Zustand vorliegt.

**Listing 6.106** Die Markierung innerhalb eines Formularfelds fett formatieren

```

Sub FettFormatieren()
    Dim doc As Word.Document
    Dim rng As Word.Range

    Set doc = ActiveDocument
    Set rng = Selection.Range
    If doc.ProtectionType <> wdNoProtection Then
        doc.Unprotect
    End If
    Selection.Font.Bold = True
    doc.Protect Type:=wdAllowOnlyFormFields, Noreset:=True
    'Aktivierung des Dokumentschutzes markiert das ganze Formularfeld.
    'Am Schluss die ursprüngliche Markierung wieder herstellen.
    rng.Select
End Sub

```

**Listing 6.107** (.NET): Die C#-Version

```

private void FettFormatieren(Wd.Range rng)
{
    Wd.Document doc = (Wd.Document) rng.Parent;
    object objKennwort = "";
    if (doc.ProtectionType != Wd.WdProtectionType.wdNoProtection)
    { doc.Unprotect(ref objKennwort); }
    rng.Font.Bold = -1;
    doc.Protect(Wd.WdProtectionType.wdAllowOnlyFormFields, ref objTrue,
        ref objKennwort, ref objFalse, ref objFalse);
    //Aktivierung des Dokumentschutzes markiert das ganze Formularfeld.
    //Am Schluss die ursprüngliche Markierung wieder herstellen.
    rng.Select();
}

```

**WICHTIG** Diese Methode funktioniert nicht mit Formularfeldern, die sich in einer Tabellenzelle befinden, da der Anwender einzelne Zeichen nicht markieren kann.



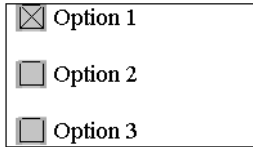
Die Beispieldatei *Bsp06\_01\_Form.doc* mit der Prozedur finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap06*.

Die Ereignisse

Jedes Formularfeld hat eine `EntryMacro`- sowie eine `ExitMacro`-Eigenschaft, der der Name einer Prozedur zugewiesen werden kann. Es darf sich nur um öffentliche Sub-Prozeduren in einem Standardmodul handeln, die keine Argumente haben. Bei der Auslösung eines solchen Makros ist das aktuelle Formularfeld immer dasjenige, das die Prozedur ausgelöst hat.

Als Beispiel dient eine Lösung, die Kontrollkästchen wie eine Gruppe Optionfelder benutzen lässt (Abbildung 6.65). Die Kontrollkästchen müssen sich in einem definierbaren Bereich befinden, wie eine Tabellenzelle oder ein Positionsrahmen. Beim Verlassen und Betreten eines Kontrollkästchens werden alle anderen deaktiviert, falls das aktuelle aktiviert ist.

Abbildg. 6.65 Kontrollkästchen verhalten sich wie Optionfelder dank der Ereignis-Makros



Der Code hierfür befindet sich in Listing 6.108. Der EntryMacro-Eigenschaft jedes Kontrollkästchens wurde »KontrollkästchenEintreten« zugewiesen und der ExitMakro-Eigenschaft »KontrollkästchenVerlassen«. Die Lösung setzt voraus, dass zwei Variable-Objekte im Dokument schon definiert sind – »AktuellesFeld« und »VorherigesFeld«. Beim Eintreten in ein Formularfeld, dessen EntryMacro-Eigenschaft auf »KontrollkästchenEintreten« festgelegt ist, wird der Wert der »AktuellesFeld«-Variable in die »VorherigesFeld«-Variable geschrieben, und der »AktuellesFeld«-Variable der Name des gerade aktuell gewordenen Formularfeldes zugewiesen. Der abzusuchende Bereich (in diesem Beispiel eine Tabellenzelle) wird bestimmt, dann die Funktion *kkAktiviert* aufgerufen.

Diese prüft das Resultat des aktuellen Felds. Falls es »1« ist, ist das Kontrollkästchen aktiviert, und alle anderen im angegebenen Bereich müssen deaktiviert sein. Es wird also durch alle Formularfelder im Bereich geschleift und, falls es sich nicht um das aktuelle handelt, werden deren Resultate auf »0« gesetzt.

Beim Verlassen eines dieser Kontrollkästchen wird *KontrollkästchenVerlassen* ausgeführt. Auch diese Prozedur führt *kkAktiviert* aus. Der Grund dafür ist, dass die Einfügemarke außerhalb des »Optionenbereichs« landen könnte, was bedeuten würde, dass *KontrollkästchenEintreten* nicht ausgeführt wird. Falls der Anwender mit der Tastatur arbeitet, könnten gleichzeitig zwei Kontrollkästchen aktiviert sein. *KontrollkästchenVerlassen* sorgt dafür, dass dieser Zustand aufgehoben wird.

**HINWEIS**

Um nach einer Gültigkeitsprüfung zu einem Problemfeld zurückkehren zu können, muss nach dem gleichen Prinzip gearbeitet werden: mit einem Eintreten/Verlassen-Makro-Paar. Beim Eintreten wird der Formularfeldname in einer Dokument-Variablen gespeichert. Beim Verlassen wird die Gültigkeitsprüfung durchgeführt und deren Ergebnis in einer weiteren Dokument-Variablen gespeichert. Beim Eintreten in das nächste Feld wird im Falle eines ungültigen Ergebnisses zurück zum ersten Feld gesprungen, ansonsten wird der Name des nächsten Feldes in die Dokument-Variable geschrieben.

Listing 6.108 Kontrollkästchen wie Optionfelder in einem Bereich präsentieren

```
Sub KontrollkästchenEintreten()
    Dim doc As Word.Document
    Dim ffld As Word.FormField
    Dim strAktuellesFeld As String
    Dim strVorherigesFeld As String
    Dim rng As Word.Range

    Set doc = ActiveDocument
    Set ffld = Selection.Bookmarks(1).Range.FormFields(1)
    strVorherigesFeld = doc.Variables("AktuellesFeld").Value
    strAktuellesFeld = ffld.Name
    doc.Variables("VorherigesFeld").Value = strVorherigesFeld
    doc.Variables("AktuellesFeld").Value = strAktuellesFeld
    Set rng = doc.FormFields(strAktuellesFeld).Range.Cells(1).Range
```

**Listing 6.108** Kontrollkästchen wie Optionenfelder in einem Bereich präsentieren (Fortsetzung)

```

    Debug.Print kkAktiviert(rng, ffld, strAktuellesFeld, strVorherigesFeld)
End Sub

Sub KontrollkästchenVerlassen()
    Dim doc As Word.Document
    Dim strAktuellesFeld As String
    Dim ffld As Word.FormField
    Dim rng As Word.Range

    Set doc = ActiveDocument
    strAktuellesFeld = doc.Variables("AktuellesFeld").Value
    Set ffld = doc.FormFields(strAktuellesFeld)
    Set rng = doc.FormFields(strAktuellesFeld).Range.Cells(1).Range
    Debug.Print kkAktiviert(rng, ffld, strAktuellesFeld, _
        doc.Variables("VorherigesFeld").Value)
End Sub

Function kkAktiviert(rng As Word.Range, ffld As Word.FormField, _
    strAktuellesFeld As String, strVorherigesFeld As String) As Boolean

    kkAktiviert = False
    If ffld.Parent.FormFields(strAktuellesFeld).Result = 1 Then
        kkAktiviert = True
    End If
    If kkAktiviert Then
        For Each ffld In rng.FormFields
            If ffld.Name <> strAktuellesFeld And ffld.CheckBox.Valid Then
                ffld.Result = 0
            End If
        Next
    End If
End Function

```



Die Beispieldatei *Bsp06\_01\_Form.doc* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap06*.

**HINWEIS**

In Kapitel 26 finden Sie ein Beispiel, wie Formularfelder für eine Rechnung oder Offerte eingesetzt werden können. Es veranschaulicht den Umgang mit dem Dokumentschutz, wie mit Formularfeldern gerechnet wird und wie sie dynamisch erstellt und angepasst werden.

## Der Seriendruck: Das *MailMerge*-Objekt

Als Entwickler stehen Ihnen mehrere Möglichkeiten zur Verfügung, Daten in ein Word-Dokument zu schreiben. Am effizientesten ist es, ein im XML-Format gespeichertes Word 2003-Dokument mit XML-Werkzeugen zu bearbeiten. Eine relativ schnörkellose Methode ist, Daten automatisiert in das Dokument einzufügen, wie im Abschnitt »Zielscheibe Textmarke: Das *Bookmark*-Objekt« in diesem Kapitel oder in Kapitel 11 im Abschnitt über ADO erklärt. Der Seriendruck jedoch bietet seit frühen Versionen eine dem Anwender vertraute Schnittstelle an, womit Datenquellen ausgewählt und deren Feldinhalt gezielt ins Dokument eingefügt werden können.



**HINWEIS**

Die Grundlagen des Seriendrucks sind in der Datei *Seriendruck.doc* auf der CD-ROM zum Buch im Ordner *\Beilagen\Zusatzmaterial Seriendruck* erklärt.

In Word 2002 wurde nicht nur eine überarbeitete Benutzerschnittstelle eingeführt, auch die Automatisierungsschnittstelle wurde der neuen Funktionalität angepasst. Die wichtigsten Änderungen sind folgende:

- Zu den unterstützten Verbindungsmethoden DDE und ODBC gesellt sich OLE DB.
- Neue Sicherheitsmaßnahmen erschweren das Öffnen von Seriendruck-Hauptdokumenten.
- Der eingeführte Aufgabenbereich lässt sich teilweise steuern.
- Neue »MailMerge«-Ereignisse ermöglichen einen Eingriff in das Zusammenführen des Seriendrucks (siehe Kapitel 7).

In diesem Abschnitt werden die wichtigsten Aspekte der Automatisierung vorgestellt.

Der  
Makrore-  
korder

Der Makrorekorder leistet beim Erforschen der Seriendruckfunktionalität nur begrenzt Hilfe. Er ist vor allem beim Erkunden der Verbindungszeichenkette nützlich, solange es sich um keine OLE DB-Verbindung handelt.

**HINWEIS**

Am Ende dieses Abschnitts finden Sie eine eingehende Diskussion über das Zusammenstellen von »Connection Strings«.



Serien-  
druck-  
Hauptdo-  
kument

Jedes Dokument hat eine MailMerge-Eigenschaft, die ein MailMerge-Objekt zurückgibt und worüber alles, was mit dem Seriendruck zu tun hat, angesprochen wird.

Ein Seriendruck-Hauptdokument ist ein gewöhnliches Dokument, dessen MailMergeType-Eigenschaft auf einen anderen Wert als wdNotAMergeDocument festgelegt wurde. Diese Eigenschaft kann abgefragt werden, um herauszufinden, um welche Art von Seriendruckdokument es sich handelt. Oder sie kann festgelegt werden, um aus einem gewöhnlichen Dokument ein Seriendruckdokument zu machen. Mögliche weitere Werte sind wdCatalog bzw. wdDirectory (Verzeichnis), wdEmail, wdEnvelopes (Umschläge), wdFax, wdFormletters (Briefe) sowie wdMailingLabels (Etiketten).

Um ein Seriendruckdokument in ein gewöhnliches Dokument zurückzuwandeln:

```
ActiveDocument.MailMerge.MainDocumentType = wdNotAMergeDocument
```

In C#

```
doc.MailMerge.MainDocumentType = wd.WdMailMergeMainDocType.wdNotAMergeDocument;
```

Diese Handlung entfernt lediglich die Datenverbindung, Seriendruckfelder werden davon nicht tangiert.

Daten-  
sätze  
anspre-  
chen

Um mit den Datensätzen zu arbeiten, wird die DataSource-Eigenschaft des MailMerge-Objekts benutzt. Da diese Funktionalität aus den Urzeiten von Word stammt, stützt sie sich auf den aktuellen Zustand des Dokuments. Dies bedeutet, dass immer mit dem aktuellen Datensatz (ActiveRecord) gearbeitet wird und, um mit einem anderen Datensatz zu arbeiten, dieser angewählt werden muss.

Ein bestimmter Datensatz wird durch Festlegen der ActiveRecord-Eigenschaft ausgewählt:

```
ActiveDocument.MailMerge.DataSource.ActiveRecord = 10
```

Beachten Sie, wie in C# die Ganzzahl zum Typ `WdMailMergeActiveRecord` konvertiert werden muss:

```
wd.MailMergeDataSource ds = doc.MailMerge.DataSource;
ds.ActiveRecord = (wd.WdMailMergeActiveRecord) (records - 2);
```

Für die relative Navigation zwischen Datensätzen gibt es zusätzlich die in Tabelle 6.28 aufgeführten `WdMailMergeActiveRecord`-Konstanten. Bitte beachten Sie, dass Sie zuerst kontrollieren sollen, ob die beabsichtigte Verschiebung auch möglich ist. Wenn nicht, wird ein Laufzeitfehler ausgelöst. Beispiel: Der erste Datensatz ist der aktuelle. Die Anweisung `ActiveRecord = wdPreviousRecord` würde Word mit dem Laufzeitfehler »5853 'Ungültiger Parameter.'« quittieren. `ActiveRecord = wdPreviousDataSourceRecord` würde den Laufzeitfehler »5852 'Das angeforderte Objekt ist nicht verfügbar.'« verursachen.

**Tabelle 6.28** Konstantenwerte für die Datensatz-Navigation

<code>WdMailMergeActiveRecord-Enum</code>	Wert	Beschreibung
<code>wdNoActiveRecord</code>	-1	Keine Datensatz ist im Dokument eingebunden bzw. es handelt sich nicht um ein Seriendruckdokument.
<code>wdNextRecord</code>	-2	Nächster verfügbarer Datensatz
<code>wdPreviousRecord</code>	-3	Vorhergehender verfügbarer Datensatz
<code>wdFirstRecord</code>	-4	Erster verfügbarer Datensatz
<code>wdLastRecord</code>	-5	Letzter verfügbarer Datensatz
<i>Folgende Konstantenwerte gelten nur für Word 2002 und später. Sie beziehen sich auf die Kontrollkästchen-Einstellungen im Dialogfeld Seriendruckempfänger.</i>		
<code>wdFirstDataSourceRecord</code>	-6	Der erste Datensatz
<code>wdLastDataSourceRecord</code>	-7	Der letzte Datensatz
<code>wdNextDataSourceRecord</code>	-8	Der nächste Datensatz
<code>wdPreviousDataSourceRecord</code>	-9	Der vorhergehende Datensatz

Die Anzahl der Datensätze gibt die Eigenschaft `RecordCount` zurück.

**HINWEIS** In Word 2000 gibt es die `RecordCount`-Eigenschaft nicht. In dieser Version muss der letzte Datensatz ausgewählt und `ActiveRecord` abgefragt werden.



Mit Einführung des Dialogfeldes *Seriendruckempfänger* (Abbildung 6.66) in Word 2002 erhielt der Anwender mehr Flexibilität. Dadurch wurde jedoch die Navigierung und Auswertung von Datensätzen durch das Objektmodell komplizierter.

**Abbildg. 6.66** Das Dialogfeld *Seriendruckempfänger* ermöglicht es dem Anwender, einzelne Datensätze aus dem Seriendruck auszulassen



Die vom Anwender ausgeschlossenen Datensätze bleiben Teil der DataSource; das RecordCount-Ergebnis ändert sich nicht. Wo in Word 2000 und früher mit `wdNextRecord` und `wdPreviousRecord` durch die Datensätze geschleift wurde, werden ab Word 2002 `wdNextDataSourceRecord` sowie `wdPreviousDataSourceRecord` benutzt und die `Included`-Eigenschaft abgefragt, um festzustellen, ob es sich um einen ausgeschlossenen Datensatz handelt, wie in Listing 6.109 und Abbildung 6.67 ersichtlich.

**Listing 6.109** Durch alle Datensätze Schleifen und feststellen, ob sie ausgeschlossen sind

```
Sub DurchDatensätzeSchleifen()
    Dim lAnzahlDS As Long
    Dim lZaehler As Long
    Dim lAnzahlAktiverDS As Long
    Dim ds As Word.MailMergeDataSource

    If ActiveDocument.MailMerge.MainDocumentType = wdNotAMergeDocument _
        Then Exit Sub
    Set ds = ActiveDocument.MailMerge.DataSource
    lAnzahlDS = ds.RecordCount
    lAnzahlAktiverDS = 0
    ds.ActiveRecord = wdFirstRecord
    For lZaehler = 1 To lAnzahlDS
        Debug.Print lZaehler, ds.ActiveRecord, ds.Included
        'Falls das Kontrollkästchen aktiviert ist, wird der Datensatz zusammengeführt
        If ds.Included Then
            lAnzahlAktiverDS = lAnzahlAktiverDS + 1
        End If
        'Den Laufzeitfehler 5852 vermeiden
        If ds.ActiveRecord <> wdLastDataSourceRecord Then
            ds.ActiveRecord = wdNextDataSourceRecord
        End If
    Next
    Debug.Print lAnzahlAktiverDS & " Datensätze werden zusammengeführt."
    'Den letzten sichtbaren Datensatz anwählen
    ds.ActiveRecord = wdLastRecord
End Sub
```

Abbildg. 6.67 Das Ergebnis von Listing 6.109 basiert auf den Einstellungen in Abbildung 6.66

Direktbereich		
1	1	Wahr
2	2	Wahr
3	3	Falsch
4	4	Wahr
5	5	Wahr
6	6	Falsch
7	7	Wahr
8	8	Wahr
9	9	Falsch
6 Datensätze werden zusammengeführt.		

### TIPP

Sie können mit der `Included`-Eigenschaft sowie der `SetAllIncludedFlags`-Methode Datensätze aus- und einschließen, analog zum Dialogfeld.



Die Beispieldatei *Bsp06\_01\_SD.doc* finden Sie auf der CD-ROM zum Buch im Ordner `\Beispiele\Kap06`. Sie ist mit der *Nordwind.mdb*-Datenbank im `\Datenbank`-Ordner verbunden.

Daten-  
sätze  
filtern

Um auf die Datenfelder zuzugreifen, stehen die `DataFields`- sowie `Fieldnames`-Auflistungen bereit. Der Datenfeldinhalt kann nur gelesen und nicht beschrieben werden.

Das *Empfänger*-Dialogfeld bzw. die `Included`-Eigenschaft ist eine Möglichkeit, in Word 2002/2003 Datensätze aus dem Seriendruckergebnis auszuschließen. Das `MailMerge.DataSource`-Objekt bietet zudem die `QueryString`-Eigenschaft an, womit über eine SQL-Select-Anweisung Datensätze nach Kriterien gefiltert und sortiert zur Verfügung gestellt werden. In Word 2000 ist diese die einzige Methode, um Datensätze auszuschließen.

### HINWEIS

In der Word 2002/2003-Benutzerschnittstelle wird diese Funktionalität durch Anklicken eines Pfeils neben einem Feldnamen und Auswahl des Eintrags *Weitere Optionen* erreicht. In Word 2000 befindet sie sich im Dialogfeld *Mail Merge...*; die Schaltfläche heißt *Abfrageoptionen*.

Eine mit der `QueryString`-Eigenschaft festgelegte Filtrierung ersetzt die mit der `OpenDataSource` Argumente `SQLStatement` und `SQLStatement1` (siehe »Wie *OpenDataSource* funktioniert«) festgelegte. Die SQL-Anweisung der `QueryString`-Eigenschaft darf höchstens 256 Zeichen betragen.

### HINWEIS

Das `SQLStatement`-Argument wird im Abschnitt »Wie *OpenDataSource* funktioniert« in diesem Kapitel erläutert. Mehr über SQL-Anweisungen und wie sie gekürzt werden können, finden Sie in der Datei *SQL.doc* auf der CD-ROM zum Buch im Ordner `\Beilagen\Zusatzmaterial Seriendruck`.

Um aus der *Personal*-Tabelle von *Nordwind.mdb* nur diejenigen auszuwählen, die nicht in den USA wohnen:

```
doc.MailMerge.DataSource.QueryString = "SELECT * FROM [Personal] WHERE Land<> 'USA'"
```

Im Gegensatz zur Included-Eigenschaft erscheinen filtrierte Datensätze nicht im Dialogfeld *Empfänger*, werden von RecordCount nicht beachtet und können über ActiveRecord nicht angesprungen werden.

Um alle Datensätze der Datenquelle wieder verfügbar zu machen:

```
doc.MailMerge.DataSource.QueryString = "SELECT * FROM [Personal]"
```



Daten-  
satz  
suchen

Die Methode FindRecord entspricht der Symbolschaltfläche *Eintrag suchen* (Ferngläser) in der Benutzeroberfläche. In Word 97 und Word 2000 hatten beide Schnittstellen mit einem Problem zu kämpfen, dass der Seriendruck nach einer erfolgreichen Suchaktion nicht zusammengeführt werden konnte; es musste zuerst eine nicht erfolgreiche Suche durchgeführt werden, um den Seriendruck wieder zu »befähigen«.

Dieses Problem wurde zwar in der Benutzeroberfläche von Word 2002 behoben, aber leider für die Automatisierungsschnittstelle vollkommen kaputt gemacht. Die Syntax sollte wie folgt aussehen, aber die Suche wird entweder nicht ausgeführt oder gibt eine Fehlermeldung zurück: »Laufzeitfehler 5852. Das angeforderte Objekt ist nicht verfügbar.«:

```
ActiveDocument.MailMerge.DataSource.FindRecord FindText:="Peter", Field:="Vorname"
```

Die Lösung? Sie können den früheren Befehl – der neuerdings FindRecord2000 heißt und als verborgenes Element im Objektkatalog geführt wird – weiterhin einsetzen:

```
ActiveDocument.MailMerge.DataSource.FindRecord2000 FindText:="Peter", Field:="Vorname"
```

Dieser Befehl leidet selbstverständlich unter dem gleichen Fehlverhalten, wie in früheren Word-Versionen, weshalb der Code für seinen Einsatz etwas komplizierter ausfällt, als zuerst angenommen. Das Prinzip des Beispielcodes in Listing 6.110 funktioniert in allen Versionen von Word, nur muss die richtige Methode verwendet werden.

Der gegenwärtige Datensatz wird in einer Variablen festgehalten, so dass im Fall eines Fehlschlags dieser Datensatz wieder angezeigt werden kann. Dann wird zum ersten Datensatz gesprungen, weil die Suche nur von dort aus funktioniert. Die Suche wird anhand der Argumente, die aus der aufrufenden Prozedur übergeben wurden, ausgeführt. Die Funktion *DS\_Suchen2000* gibt bei Erfolg »Wahr« zurück, sonst »Falsch«. Nach einer erfolgreichen Suche muss eine erfolglose durchgeführt werden, um den Seriendruck wieder zu befähigen. (Das ANSI-Zeichen 07 verwendet Word für Tabellenstrukturen, wird aber kaum als Text im Seriendruck vorkommen.) Sonst wird zum ursprünglichen Datensatz zurückgesprungen.

#### ACHTUNG

Die Suche funktioniert nur, wenn der Feldname in der *MailMerge*-Feldfunktion nicht von Anführungszeichen umgeben ist. Da spätere Versionen von Word Feldnamen so einführen, müssen Sie unter Umständen zuerst die Feldcodes bearbeiten, um die Anführungszeichen zu entfernen.

**Listing 6.110** Einen Datensatz suchen in Word bis einschließlich Version 2003

```
Function DS_Suchen2000(mm As Word.MailMerge,
    strText As String, strFeld As String) As Boolean
    Dim lDS As Long
    Dim bGefunden As Boolean

    lDS = mm.DataSource.ActiveRecord
    mm.DataSource.ActiveRecord = wdFirstRecord
    bGefunden = mm.DataSource.FindRecord2000(FindText:=strText, Field:=strFeld)
    If bGefunden Then
        mm.DataSource.FindRecord2000 FindText:=Chr$(7), Field:=strFeld
    Else
        mm.DataSource.ActiveRecord = lDS
    End If
    DS_Suchen2000 = bGefunden
End Function
```

#### HINWEIS

In der Beispieldatei werden die Suchtext- und Feld-Angaben über ein UserForm festgelegt. Die Version von Word wird ermittelt und die Informationen an die passende Funktion übergeben.



Die Beispieldatei *Bsp06\_01\_SD.doc* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap06*. Sie ist mit der *Nordwind.mdb*-Datenbank im *\Datenbank*-Ordner verbunden.

Seriendruckdokument öffnen

Eigentlich lässt sich ein Seriendruckdokument wie jedes andere Dokument öffnen. Etwas umständlicher wurde dies jedoch in Word 2002 mit Service Pack und 2003 durch die Einführung neuer Sicherheitsmaßnahmen. In diesen Versionen wird standardmäßig beim automatisierten Öffnen eines Seriendruck-Hauptdokuments mit verbundener Datenquelle die Datenverbindung lautlos gekappt und die Verbindungs-Information aus dem Dokument entfernt. Der Code (oder der Anwender) muss jedes Mal die Daten neu einbinden. Dieser Umstand ist nicht immer zufriedenstellend. Das Verhalten kann durch Festlegung eines Registry-Eintrags ausgeschaltet werden. Dieser Vorgang ist im Knowledge Base-Artikel »"Opening This Will Run the Following SQL Command" Message When You Open a Word Document – 825765« unter <http://support.microsoft.com?kbid=825765/en-us> beschrieben.

Noch eine Änderung gibt es seit Word 2002: Es wird keine Seriendruckschnittstelle – weder die Symbolleiste noch der neue Assistent *Aufgabenbereich* – beim Öffnen eines Seriendruck-Hauptdokuments eingeblendet. Wenn Sie dem Anwender eine der beiden anbieten wollen, können Sie das mit einem AutoOpen- oder Document\_Open-Makro tun, wie in Listing 6.111.

Der Seriendruck-Assistent

Im Allgemeinen stellt VBA keine Schnittstelle für die Änderung oder Anpassung der Aufgabenbereiche zu Verfügung. Bekanntlich bestätigt die Ausnahme die Regel: In Word 2002 und 2003 kann der Seriendruck-Assistent begrenzt angepasst werden.

Mit der ShowWizard-Methode ist es möglich, den Seriendruck-Assistent, voreingestellt mit einem bestimmten Schritt, einzublenden sowie festzulegen, welche Schritte überhaupt zur Verfügung stehen.

Das Ereignis MailMergeWizardStateChange wird bei jedem Wechsel zwischen den Schritten des Aufgabenbereichs ausgelöst. Somit können mit VBA zusätzliche Handlungen durchgeführt werden.

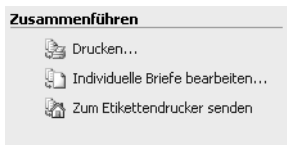
Der Inhalt des Aufgabenbereichs jedoch kann nur im sechsten und letzten Schritt beeinflusst werden: Die Methode ShowSendToCustom ermöglicht die Hinzufügung eines zusätzlichen, vom Entwickler definierten Eintrags. Und das Ereignis MailMergeWizardSendToCustom wird beim Anklicken des Eintrags ausgeführt, um die Seriendruckzusammenführung abzufangen und umzuleiten (beispielsweise zu einem bestimmten Fax- oder anderen Drucker).

Nehmen wir beispielsweise an, es liegt eine Seriendruck-Hauptdokumentvorlage vor. Die Datenquelle ist bereits verknüpft und darf nicht geändert werden. Es bleibt dem Benutzer lediglich die Datenauswahl sowie die Zusammenführung. Damit erübrigen sich die ersten vier Schritte des Seriendruck-Assistenten.

Dieser Aufgabenbereich soll also im letzten Schritt eingeblendet werden und nur der fünfte steht zusätzlich noch zur Verfügung; die übrigen sind gesperrt. Zudem wollen wir die Seriendruck-Symboleiste auch sperren, so dass der Benutzer dazu angehalten wird, nur die Funktionalität zu benutzen, die wir ihm zur Verfügung stellen.

Mit der Methode ShowSendToCustom wird dem sechsten Schritt der zusätzliche »Menüpunkt« in Abbildung 6.68 hinzugefügt, den der Benutzer auswählen soll, um den Seriendruck zum richtigen Drucker zu senden.

**Abbildg. 6.68** Der Aufgabenbereich wurde mittels VBA um den letzten Eintrag ergänzt



Es liegt auf der Hand, dass diese Handlungen beim Erstellen eines neuen oder beim Öffnen eines vorhandenen Dokuments auszuführen sind. Deshalb wird die Prozedur in Listing 6.111 sowohl vom Document\_New- als auch vom Document\_Open-Ereignis aufgerufen.

**Listing 6.111** Den Seriendruck-Assistent nur mit dem fünften und sechsten Schritt einblenden, der sechste ist ausgewählt

```
Public Sub SeriendruckAssistentEinblenden(Doc As Word.Document)

    If doc.MailMerge.MainDocumentType <> wdNotAMergeDocument Then
        Doc.MailMerge.ShowWizard InitialState:=6, ShowDocumentStep:=False, _
            ShowTemplateStep:=False, ShowDataStep:=False, ShowWriteStep:=False, _
            ShowPreviewStep:=True, ShowMergeStep:=True
        Doc.MailMerge.ShowSendToCustom = "Zum Etikettendrucker senden"
        Application.CustomizationContext = Doc
        Application.CommandBars("Mail merge").Enabled = False
    End If
End Sub
```

#### WICHTIG

Die ShowWizard-Methode wird bei der Einstellung InitialState auf 4 oder höher fehlschlagen, falls das Dokument mit keiner Datenquelle verbunden ist. Die Schritte 4 bis 6 stehen nur in einem Seriendruck-Hauptdokument mit verknüpfter Datenquelle zur Verfügung.



Die Beispieldatei *Bsp06\_02\_SD.dot* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap06*. Sie ist mit der *Nordwind.mdb*-Datenbank im *\Datenbank*-Ordner verbunden.



Serien-  
druck  
zusam-  
menfüh-  
ren

Ein Seriendruck kann in ein neues Dokument, direkt zum Drucker, zur Fax-Ausgabe oder zur E-Mail-Ausgabe zusammengeführt werden (vorausgesetzt, dass das Ausgabezielgerät vorhanden und verfügbar ist). Im Objektmodell wird die *Destination*-Eigenschaft des *MailMerge*-Objekts festgelegt. Die entsprechenden *WdMailMergeDestination*-Werte sind *wdSendToNewDocument*, *wdSendToPrinter*, *wdSendToFax* sowie *wdSendToEmail*. Die Zusammenführung wird mit der *Execute*-Methode ausgelöst. Ein Beispiel hierfür sehen Sie in Listing 6.113.

#### PROFITIPP

Wenn der Seriendruck in ein neues Dokument zusammengeführt wird, gibt die *Execute*-Methode kein *Document*-Objekt zurück, womit das Ergebnisdokument direkt angesprochen werden kann. Meist ist das aktuelle Dokument (*ActiveDocument*) das Seriendruckresultat, sicher ist es jedoch nicht. Eine nützliche Kontrolle bietet eine im Seriendruck-Hauptdokument gespeicherte *Dokument-Variable*. Das Resultat »erbt« diese. Somit kann ihr Vorhandensein getestet werden.

Wird Word 2002 oder 2003 automatisiert, besteht die Möglichkeit, mit Seriendruckereignissen zu arbeiten. Das *MailMergeAfterMerge*-Ereignis stellt ein entsprechendes *Document*-Objekt zur Verfügung. Mehr über Seriendruckereignisse steht in Kapitel 7 beschrieben.

Nach der Zusammenführung hat das Seriendruckresultat-Dokument die gleiche Dokumentvorlage wie das Hauptdokument. Damit müsste es Zugang zu den gleichen Symbolleisten und Makros haben. Diese Verknüpfung ist zu diesem Zeitpunkt jedoch nicht vollständig. Falls der Anwender das Resultat mit Makros bearbeiten soll, muss nach der Zusammenführung die Verknüpfung zur Vorlage explizit hergestellt werden:

```
ActiveDocument.AttachedTemplate = doc.AttachedTemplate
```

Hauptdo-  
kument  
aufstellen

Es kommt eher selten vor, dass Seriendruck-Hauptdokumente für Briefe von Grund auf über eine Automatisierungsschnittstelle erstellt werden. Meistens werden sie als Vorlage bereitgestellt oder vom Anwender für eine einmalig vorkommende Aufgabe erstellt. Die automatische Erstellung gestaltet sich jedoch als nicht besonders schwierig, da ein Seriendruck-Hauptdokument in Wirklichkeit nichts anderes als ein gewöhnliches Dokument mit *Mergefield*-Feldfunktionen ist (der Umgang mit Feldfunktionen ist im Abschnitt »Feldfunktionen« in diesem Kapitel beschrieben). Ausnahmen bilden die Erstellung von Umschlägen und Etiketten.

Um-  
schläge

Nicht gerade intuitiv für den Entwickler ist das Erstellen von Umschlägen für den Seriendruck. Word macht es für den Anwender relativ einfach: er wählt Umschläge als Seriendruck-Hauptdokument aus, legt im automatisch folgenden Dialogfeld das Umschlagformat fest, und Word stellt ihm ein Blatt im richtigen Format und in der korrekten Ausrichtung zur Verfügung. Ein Absatz ist bereits mit der Formatvorlage *Umschlagadresse* formatiert; diese positioniert die Adresse mittels eines Positionsrahmens. Der Anwender muss darin lediglich die Seriendruckfelder einfügen.

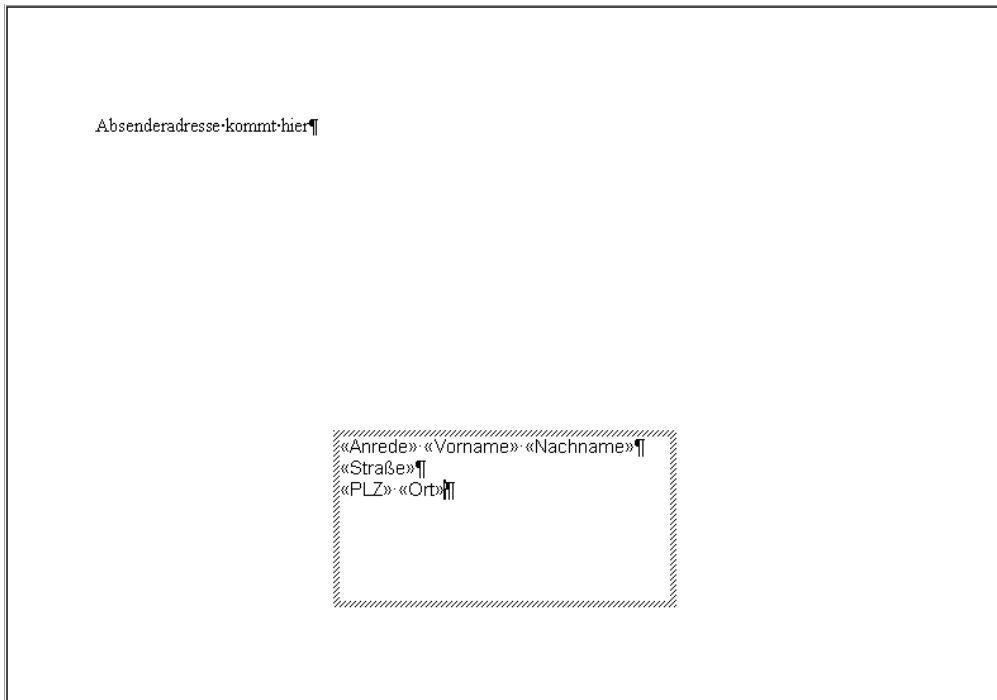
Der Entwickler muss alles selbst erledigen, und der Makrorekorder leistet bei der Suche nach den nötigen Befehlen keinen großen Dienst. Er nimmt lediglich die Festlegung der Art des Seriendruck-Hauptdokuments auf, aber nicht das Umschlagformat. Das Listing 6.112 zeigt, was Sie alles so benötigen, um einen Umschlagseriendruck »nach Maß« zu erstellen.



Ein neues Dokument wird erstellt und die Papiergröße sowie Ausrichtung für den Umschlag werden festgelegt. Die Absenderadresse kommt oben links, wo die Einfügemarke steht. Das geht ganz einfach.

Etwas schwieriger ist die Frage der Empfängeradresse. Word benutzt die Formatvorlage *Umschlagadresse*, um sie in einem Positionsrahmen zu positionieren. Das können Sie auch tun, oder Sie könnten eine eigene Formatvorlage erstellen und einsetzen, oder mit direkter Formatierung arbeiten. Wir entschieden uns für die Formatvorlage *Umschlagadresse* und formatierten einen zweiten, eingefügten Absatz damit. Die Seriendruckfelder werden in diesen Bereich eingefügt, wie die Abbildung 6.69 veranschaulicht.

Abbildg. 6.69 Die Adressfelder wurden in einen Positionsrahmen einzeln eingefügt



#### TIPP

Es wäre auch möglich, einem Seriendruck-Hauptdokument einen Umschlag hinzuzufügen: `ActiveDocument.Envelope.Insert`. Die Seriendruckfelder werden entsprechend dem Listing 6.112 eingefügt.

Listing 6.112 Einen Umschlag als Seriendruck-Hauptdokument erstellen

```
Sub UmschlagSeriendruck()
    Dim doc As Word.Document, rng As Word.Range
    Dim strDatenPfad As String

    strDatenPfad = "..\Datenbank\Nordwind.mdb"
    Set doc = Documents.Add
```

**Listing 6.112** Einen Umschlag als Seriendruck-Hauptdokument erstellen (*Fortsetzung*)

```

doc.PageSetup.PaperSize = wdPaperEnvelopeC5
doc.PageSetup.Orientation = wdOrientLandscape

Selection.Range.Text = "Absenderadresse kommt hier"
Set rng = doc.Range
rng.InsertAfter vbCr
rng.Collapse wdCollapseEnd
rng.Style = "Umschlagadresse"
With doc.MailMerge
    .OpenDataSource Name:=strDatenPfad, _
        Connection:="TABLE Personal", SQLStatement:="SELECT * FROM 'Personal'"
    .MainDocumentType = wdEnvelopes
    FeldEinfuegen "Anrede", rng
    rng.Text = " "
    FeldEinfuegen "Vorname", rng
    rng.Text = " "
    FeldEinfuegen "Nachname", rng
    rng.Text = vbCr
    FeldEinfuegen "Straße", rng
    rng.Text = vbCr
    FeldEinfuegen "PLZ", rng
    rng.Text = " "
    FeldEinfuegen "Ort", rng
    'Direkt auf den Drucker ausgeben
    .Destination = wdSendToPrinter
    .Execute
End With
End Sub

Sub FeldEinfuegen(strFeld As String, ByRef rng As Word.Range)
    Dim fld As Word.Field

    rng.Collapse Direction:=wdCollapseEnd
    Set fld = rng.Fields.Add(Range:=rng, Type:=wdFieldEmpty, _
        Text:="Mergefield " & strFeld, PreserveFormatting:=False)
    Set rng = fld.Result
    rng.TextRetrievalMode.IncludeFieldCodes = False
    rng.Collapse Direction:=wdCollapseEnd
    rng.MoveStart wdCharacter, 2
End Sub

```



Etiketten

Die Beispieldatei *Bsp06\_03\_SD.doc* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap06*.

Auch für das Herstellen von Etiketten zeichnet der Makrorekorder lediglich die Festlegung der Seriendruckart als Etikett auf, gibt jedoch keine Auskünfte über die Auswahl des Etiketts oder die Erstellung eines Etikettenblatts.

Es stellt sich heraus, dass keine eigene Methode für Seriendrucketiketten im Word Objektmodell vorgesehen ist. Der Programmierer muss sich der `MailingLabel.CreateNewDocument`-Methode für gewöhnliche Etiketten bedienen. Ebenso wenig steht eine Liste der vorhandenen Etikettenbezeichnungen zur Verfügung. Auch eine entsprechende Methode für die neue Word 2002-Funktionalität

*Alle Etiketten aktualisieren* fehlt. Stattdessen muss man den internen Word-Befehl über das Word-Basic-Objektmodell aufrufen.

Wir legen deshalb Wert darauf, in Listing 6.113 als Beispiel die Technik zu veranschaulichen. Die Etikettenbezeichnung eines bestimmten Etiketts entnehmen Sie dem Feld *Bestellnummer* im Dialogfeld *Etiketten einrichten*, das über die Befehlsfolge *Extras/Briefe und Sendungen/Umschläge und Etiketten/Etiketten/Optionen* erreicht wird. Meistens wird nur die Nummer, und nicht die Beschreibung, gebraucht.

Wenn Sie Word 97 oder Word 2000 automatisieren, müssen Sie mühsam jedes Etikett in einer eigenen UserForm auflisten, falls der Anwender die Etikettenart auswählen soll. Ab Word 2002 ist jedoch die neue Methode `Application.MailingLabel.LabelOptions` vorhanden. Damit kann dem Anwender das Word-Dialogfeld zur Auswahl der Etikettenart eingeblendet werden. Die Auswahl der Etikettenart hat einen direkten Einfluss auf das Seriendruck-Hauptdokument, wir können sie nicht abfangen.

Das Beispiel bedient sich der in Word 2002 eingeführten `AddressBlock`-Feldfunktion für den Seriendruck. Die Zusammenstellung der Adresselemente wird in der Zeichenkette-Variable `strAdresse` festgelegt.

Als Datenquelle haben wir die Outlook-Kontaktliste gewählt und in der SQL-Anweisung die Felder, wo nötig, umbenannt, so dass sie automatisch von der `AddressBlock`-Funktion erkannt werden.

Da in diesem Beispiel wir, und nicht der Anwender über das Dialogfeld, das Etikett festlegen, müssen wir dafür ein neues Etiketten-Dokument erstellen:

```
Application.MailingLabel.CreateNewDocument(Name:= strEtikettenname)
```

Es ist zu beachten, dass dieses Dokument nicht automatisch vom Typ Seriendruck-Etikett ist. Dies müssen wir ausdrücklich mit `MainDocumentType = wdMailingLabels` festlegen.

Da die Einfügemarke im ersten Etikett steht, ist es kein Problem, die `AddressBlock`-Feldfunktion einfach in den `Selection.Range` einzufügen. Weitere Seriendruckfelder werden nicht benötigt. Der Inhalt des ersten Etiketts wird mit `WordBasic.MailMergePropagateLabel` in alle übrigen Etiketten kopiert.

---

**ACHTUNG** Unter dem Tablet PC-Betriebssystem funktioniert `WordBasic.MailMergePropagateLabel` fehlerhaft; die Felder werden nicht in alle Tabellenzellen des Etikettenblatts kopiert. Wenn eine Lösung auf einem solchen System eingesetzt wird, muss sie wie für Word 2000 entwickelt werden.

---



---

**HINWEIS** Dieses Beispiel ist für Word 2002 und 2003 beschrieben. Etiketten für Word 97 und 2000 werden mit VBA ähnlich erstellt; nur fehlt in diesen Versionen die `AddressBlock`-Feldfunktion sowie die Funktionalität, das erste Etikett in allen anderen Tabellenzellen automatisch zu kopieren. Dies macht die Erstellung etwas aufwändiger.

---

**Listing 6.113** Seriendruck-Etiketten erstellen

```

Sub SeriendruckEtikettenErstellen()
    Dim doc As Word.Document
    Dim strAdresse As String
    Dim strSQL As String
    Dim strEtikettenname

    strEtikettenname = "J8160"
    strAdresse = "\f ""<< TITLE0 >><< FIRST0 >><< LAST0 >>" & vbCrLf & _
        "<< COMPANY " & vbCrLf & ">><< STREET1 " & vbCrLf & ">><< STREET2 " & _
        "& vbCrLf & ">><< POSTAL >><< CITY >><<, STATE >><<" & vbCrLf & _
        "COUNTRY >>" & "\l 1031 \c 2\e ""Deutschland""
    strSQL = "Select Vorname, Nachname, Firma, [Position] as Anrede, " & _
        "[Straße] as Address1, Ort, Bundesland, [PLZ] as PostalCode, Land From [Kontakte]"

    'Die Etiketteneinteilung wird in ein neues Dokument erstellt
    Set doc = Application.MailingLabel.CreateNewDocument(Name:= strEtikettenname)
    'Etiketten werden immer als eine Tabelle aufgestellt
    'Erste Zeile der Etiketten oben (statt in der Mitte) ausrichten
    doc.Tables(1).Range.Cells.VerticalAlignment = wdCellAlignVerticalTop
    With doc.MailMerge
        'Dokument als Seriendruck-Hauptdokument bezeichnen
        .MainDocumentType = wdMailingLabels
        'Um die Auswahl der Art des Etiketts dem Benutzer zu überlassen,
        'folgende Zeile einsetzen statt MailingLabel.CreateNewDocument
        .Application.MailingLabel.LabelOptions
        'Datenquelle einbinden
        .OpenDataSource Name:="Kontakte", SQLStatement:=strSQL, _
            SubType:=wdMergeSubTypeOutlook
        'Eine Addressblock Feldfunktion einfügen
        doc.Fields.Add Range:=Selection.Range, Type:=wdFieldAddressBlock, Text:=strAdresse
        'Erstes Etikett, mit Adressblock, zu allen anderen kopieren
        WordBasic.MailMergePropagateLabel
        'Seriendruck in ein neues Dokument zusammenführen
        .Destination = wdSendToNewDocument
        .Execute
        'Seriendruck-Hauptdokument schließen, ohne es zu speichern
        doc.Close SaveChanges:=wdDoNotSaveChanges
    End With
End Sub

```



Die Beispieldatei *Bsp06\_04\_SD.doc* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap06*.



Daten-  
quelle  
einbinden

Betrachten wir zunächst, welche Informationen benötigt werden, um eine Verbindung zur Datenquelle herzustellen. Dafür ist die `OpenDataSource`-Methode verantwortlich, die die folgende Syntax hat. Die einzelnen Parameter werden in Tabelle 6.29 erläutert, und die Angaben aus der Hilfe zum Thema ergänzt.

```

OpenDataSource(Name, [Format], [ConfirmConversions], [ReadOnly], [LinkToSource], _
    [AddToRecentFiles], [PasswordDocument], [PasswordTemplate], [Revert], _
    [WritePasswordDocument], [WritePasswordTemplate], [Connection], [SQLStatement], _
    [SQLStatement1], [OpenExclusive], [SubType])

```

**HINWEIS** Subtype und OpenExclusive wurden in Word 2002 eingeführt und werden von Word 2000 nicht unterstützt.

Tabelle 6.29 Parameter der Methode *OpenDataSource*

Parameter-Name	Bemerkungen
Name	Die Hilfe bezeichnet diesen Parameter als »Erforderlicher String-Wert. Der Dateiname der Datenquelle«. In diesem Fall bedeutet »Erforderlich« lediglich, dass der Parameter vorhanden sein muss, er darf aber eine »leere« Zeichenkette übergeben. Bei ODBC-Verbindungen mit Benutzer- oder System-DSN muss er eine leere Zeichenkette enthalten.
Format	* Optionaler Variant-Wert. Standardmäßig wird von Word <b>wdOpenFormatAuto</b> verwendet.
ConfirmConversions	* Optionaler Variant-Wert.
ReadOnly	* Optionaler Variant-Wert.
LinkToSource	Optionaler Variant-Wert. <b>True</b> , um die Abfrage-SQL-Anweisung der <b>Connection</b> - und <b>SQLStatement</b> -Parameter bei jedem Öffnen des Dokuments auszuführen.
AddToRecentFiles	* Optionaler Variant-Wert.
PasswordDocument	Optionaler Variant-Wert. Falls die Datenquelle ein Word-Dokument mit Kennwortschutz ist, das Kennwort hier übergeben, um das Datenquellen-Dokument zu öffnen. Fehlt der Parameter oder wird nur eine leere Zeichenkette übergeben, blendet Word eine Eingabeaufforderung ein. Wird das falsche Kennwort übergeben, schlägt die <b>OpenDataSource</b> -Methode fehl.
PasswordTemplate	* Optionaler Variant-Wert
Revert	* Optionaler Variant-Wert
WritePasswordDocument	Optionaler Variant-Wert. Falls ein Word-Dokument mit einem Kennwort zum Ändern gespeichert wurde, das Kennwort hier übergeben, um das Datenquellen-Dokument zu öffnen. Fehlt der Parameter oder wird nur eine leere Zeichenkette übergeben, blendet Word eine Eingabeaufforderung ein. Wird das falsche Kennwort übergeben, schlägt die <b>OpenDataSource</b> -Methode fehl.
WritePasswordTemplate	* Optionaler Variant-Wert.

**Tabelle 6.29** Parameter der Methode *OpenDataSource* (Fortsetzung)

Parameter-Name	Bemerkungen
Connection	<p>Optionaler Variant-Wert. Der Bereich, auf dem die Abfrage in den <b>SQLStatement</b>-Parametern ausgeführt wird. Wie der Bereich festzulegen ist, kommt auf die Datenverbindungsmethode an. Beispiele:</p> <p>Für Daten, die über eine ODBC-Verbindung eingebunden werden, ist ein »Connection String« mit gültigem DSN erforderlich.</p> <p>Für Excel-Daten, die mit einer DDE-Verbindung eingelesen werden, gibt man einen benannten Arbeitsblatt-Bereich ein.</p> <p>Eine DDE-Verbindung mit Access erfordert den Namen einer Tabelle oder Abfrage.</p> <p>Die Grundlagen für diesen Parameter werden am besten durch Aufzeichnung eines Makros ermittelt. Weitere Informationen folgen weiter unten.</p>
SQLStatement	Optionaler Variant-Wert. Bestimmt eine Abfrage, um die Daten zu filtern.
SQLStatement1	Optionaler Variant-Wert. Falls die SQL-Anweisung länger als 255 Zeichen ist, kann sie mit diesem Parameter um zusätzliche 255 Zeichen erweitert werden.
OpenExclusive (nicht verfügbar in Word 2000 und früher)	Optionaler Variant-Wert. Öffnet die Datenquelle angeblich »im exklusiven Modus«. Unsere Tests mit Access haben jedoch keine Wirkung gezeigt. Wird vom Makrorekorder nicht aufgezeichnet.
SubType (nicht verfügbar in Word 2000 und früher)	<p>Optionaler Variant-Wert. In der VBA-Hilfe nicht dokumentiert, wird aber vom Makrorekorder aufgezeichnet. Akzeptiert einen der folgenden <b>WdMergeSubType</b>-Werte und beeinflusst, wie Word die Verbindung einer Datenquelle gestaltet.</p> <p><b>wdMergeSubTypeAccess</b></p> <p><b>wdMergeSubTypeOAL</b></p> <p><b>wdMergeSubTypeOLEDBText</b></p> <p><b>wdMergeSubTypeOLEDBWord</b></p> <p><b>wdMergeSubTypeOther</b></p> <p><b>wdMergeSubTypeOutlook</b></p> <p><b>wdMergeSubTypeWord</b></p> <p><b>wdMergeSubTypeWord2000</b></p> <p><b>wdMergeSubTypeWorks</b></p>
<p>* Diese Parameter stammen aus der <b>Open</b>-Methode des <b>Document</b>-Objekts, haben für die Seriendruck-<b>OpenDataSource</b>-Methode jedoch keine Wirkung. Sie können ruhig weggelassen werden.</p>	

## Wie *OpenDataSource* funktioniert

Die Angaben für den Parameter **Connection** sind ausschlaggebend für den Erfolg der **OpenDataSource**-Methode, und es gibt dafür unzählige Permutationen, je nach System, Datenquelle und Verbindungsmethode. Vereinfacht ausgedrückt lauten die Faustregeln:

- Zuerst wird der Parameter **Name** ausgewertet, der entweder den Pfadnamen einer Datei enthält oder eine leere Zeichenkette. Falls eine leere Zeichenkette vorliegt, erwartet Word im **Connection** Parameter einen gültigen ODBC DSN, entweder vom Typ »System« oder »Benutzer«.

- Steht am Anfang des Connection-Parameters ein gültiger DSN, versucht der Seriendruck eine ODBC-Verbindung herzustellen.
- Sonst wird für Excel und Access versucht, mit den Angaben des Name-Parameters eine DDE-Verbindung herzustellen, falls der Code Word 97 oder Word 2000 automatisiert. In Word 2002 und 2003 ebenfalls, wenn SubType:=wdMergeSubType2000.
- Für alle andere Datenbankarten wird Word mangels ODBC DSN versuchen, in Word 97 und 2000 sowie in Word 2002 und 2003 bei SubType:=wdMergeSubType2000, die Daten über einen Konvertierfilter bereitzustellen (was voraussichtlich nur mit Word-Dokumenten, zeichengetrennten Textdateien und Tabellenkalkulationsblättern gelingen wird).
- In Word 2002 und 2003 ohne SubType:=wdMergeSubType2000 wird versucht, eine OLE DB-Verbindung zu erstellen.

Allgemein ist zu erwarten, dass sich die OpenDataSource-Methode in Word 2000 anders verhält als in Word 2002 oder 2003. Diese letzteren sind sich in dieser Beziehung sehr ähnlich, falls alle Service Packs installiert sind. Neben der Version, inkl. Service Packs können die folgenden Umstände das Resultat beeinflussen:

- Installierte Version von MDAC (Microsoft Data Access)
- Installierte Versionen von ODBC-Treibern und OLE DB-Provider
- Installierte Versionen der Datenquellen-Anwendungen (beispielsweise Excel oder Access) und deren Einstellungen (ob ANSI 89 oder ANSI 92 beispielsweise in Access aktiviert ist)

Es ist also äußerst wichtig, dass Lösungen, die OpenDataSource einsetzen, gründlich mit den Zielversionen von Word, Windows und der Datenquelle getestet werden.

**HINWEIS**

Wir machen darauf aufmerksam, dass nicht alle Beispiele auch in Word 2002 getestet wurden. Angaben zur Beispieldatei *Bsp06\_05\_SD.doc* für diesen Abschnitt finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap06*.

## Datenverbindungen unter Word 2000, 2002 sowie 2003

Alle Beispiele bis zum Abschnitt über OLE DB sollten unter Word 2000, 2002, sowie 2003 funktionieren. Da OLE DB erst ab Word 2002 unterstützt wird, laufen diese Beispiele nicht unter früheren Versionen. Meistens werden Sie für Word 2002 und 2003 die OpenDataSource-Kommandozeile mit dem Argument Subtype:=wdMergeSubtypeWord2000 ergänzen müssen.

Beispiel: Der folgende Code in Word 2000

```
ActiveDocument.MailMerge.OpenDataSource _
    Name:="", Connection:="DSN=wbdb-excel-us;", _
    SQLStatement:="SELECT * FROM 'Bestellungen'"
```

muss für die späteren Versionen wie folgt aussehen:

```
ActiveDocument.MailMerge.OpenDataSource _
    Name:="", Connection:="DSN=wbdb-excel-us;", _
    SQLStatement:="SELECT * FROM 'Bestellungen'", _
    Subtype:= wdMergeSubtypeWord2000
```

## Ohne ODBC oder OLE DB: Textdateien, Dokumente usw.

Die in diesem Abschnitt vorgestellten Beispiele für die OpenDataSource-Methode sollten in allen Word-Versionen ab 97 gleich funktionieren.

Falls ein Benutzer- oder System-DSN für den Text-ODBC-Treiber auf dem Rechner vorhanden ist, wird Word bei Textdateien versuchen, eine ODBC-Verbindung aufzubauen, anstatt seinen internen Textdatei-Konvertierfilter zu benutzen. Deshalb haben die Dateinamen dieser Beispiele die Endung *.dat*, die der ODBC-Treiber nicht automatisch erkennt. Ab Word 2002 können Sie mit dem Argument *wdMergeSubtypeOther* eine Verbindung mit dem Textkonvertierfilter erzwingen, ohne eine dem ODBC-Treiber unbekannte Endung benutzen zu müssen.

Die OpenDataSource-Methode verfügt nicht über Argumente, die die Festlegung der Feld- und Datensatztrennzeichen ermöglichen. Wenn Word die Trennzeichen nicht ermitteln kann, wird das Dialogfeld *Trennzeichen im Steuersatz* eingeblendet. Vor allem scheinen die folgenden Umstände das Einblenden des Dialogfelds zu provozieren:

- Die Textdatenquelle besteht aus nur zwei Zeilen: Feldnamen und einem Datensatz. Im Knowledge Base-Artikel »212362: Steuersatztrennzeichen auswählen bei Steuersatzquelle« (<http://support.microsoft.com/default.aspx?scid=kb;de;212362>) finden Sie mehr Informationen dazu.
- Mehr als eines der üblichen Trennzeichen befindet sich in den Angaben der ersten Zeile (ob Feldnamen oder Daten) und diese sind nicht mit Anführungszeichen umgeben. Dies kommt vor allem bei Datumsangaben vor oder wenn ein Komma als Dezimaltrennzeichen dient.
- Die Zeilen sind sehr lang.

### HINWEIS

In den Beispieldatenquellen für diesen Abschnitt stehen Feldinhalte, die aus mehreren Zeilen bestehen, zwischen Anführungszeichen, so dass Word die korrekte Anzahl der Felder für jeden Datensatz erkennt. Die einzige Ausnahme sind Word-Tabellen, wo diese Maßnahme nicht notwendig ist. (Die Anführungszeichen würden in diesem Fall als Teil der Feldinhalte betrachtet und im Seriendruckresultat erscheinen.)

## Word-Dokument sowie RTF- und HTM-Dateien – einfachste Version

(Die RTF- und HTML-Dateien sind als Word-Dokument zu öffnen.)

```
ActiveDocument.MailMerge.OpenDataSource Name:="c:\wbdb\bestellungen-tab-1-fn.doc"
```

## Word-Dokument sowie RTF- und HTM-Dateien – mit Abfrage

Das folgende Beispiel wählt alle Bestellungen für den Kunden »CHOPS« aus. Bitte beachten Sie, dass

- die in der FROM-Klausel angegebene »Tabelle« nicht unbedingt eine Beziehung zum Dateinamen haben muss, aber die Klausel sowie ein Name müssen gleichwohl in der SELECT-Anweisung vorhanden sein. Falls ein vom Dateinamen abweichender Name in der SELECT-Anweisung steht, stürzt Word 2002/3 beim Versuch, das Dialogfeld *Empfängerliste* anzuzeigen, ab.



- der Feldname im Dokument »Kunden-Code« lautet, Word ihn jedoch in »KundenCode« umwandelt. Letztere Version muss in der Abfrage benutzt werden; »Kunden-Code« würde fehlschlagen.

```
ActiveDocument.MailMerge.OpenDataSource Name:="c:\wdbdb\bestellungen-tab-1-fn.rtf", _
SQLStatement:=" SELECT * FROM x WHERE KundenCode = 'CHOPS'"
```

### Textdateien – einfachste Version

Solange die oben erwähnten Bedingungen erfüllt sind, soll das folgende Beispiel die Anzeige des Dialogfelds *Trennzeichen im Steuersatz* nicht auslösen:

```
ActiveDocument.MailMerge.OpenDataSource Name:="c:\wdbdb\bestellungen-tab-1-fn.txt", _
SubType:=wdMergeSubTypeOther
```

Um auch eine Steuersatzdatei einzubinden:

```
ActiveDocument.MailMerge.OpenHeaderSource Name:="c:\wdbdb\bestellungen-tab-1-nur-fn.txt"
ActiveDocument.MailMerge.OpenDataSource Name:="c:\wdbdb\bestellungen-tab-1.txt"
```

### Textdateien – mit Abfrage

Wie im Beispiel für Dokumente, alle Datensätze mit dem »KundenCode« (und nicht »Kunden-Code«!) CHOPS:

```
ActiveDocument.MailMerge.OpenDataSource Name:="c:\wdbdb\bestellungen-tab-1.txt", _
SQLStatement:=" SELECT * FROM x WHERE KundenCode = 'CHOPS'"
```

### Codierte Textdateien

Falls Word den Eindruck hat, eine Textdatei ist codiert, zeigt es das Dialogfeld an. Nur wenn Word von sich aus die Codierung erkennt (oder sie zu erkennen meint), wird das Dialogfeld nicht eingeblendet. Dessen Anzeige kann mit der `DisplayAlerts`-Methode unterdrückt werden (denken Sie daran, Meldungen nachher wieder zu aktivieren):

```
Application.DisplayAlerts = wdAlertsNone
ActiveDocument.MailMerge.OpenDataSource
Name:="c:\wdbdb\bestellungen-semikolon-1-fn-unicode.dat"
```

### XML-Dateien

Es gibt keinen Weg, eine XML-Datei direkt als Datenquelle in Word einzubinden. Unter Umständen kann die XML-Datei in Excel geöffnet, als Excel-Datei gespeichert werden und diese als Datenquelle eingebunden werden.

### Adressbücher

Über die Jahre gab es mehrere »Adressbücher« in der Windows-Umgebung, und nicht alle können (oder konnten) automatisiert werden. Auch für Outlook allein ist die Lage nicht immer klar. Das Verhalten gestaltet sich anders, je nach Version von Outlook und MAPI, dem Modus (als »Internet

Mail Only« oder »Workgroup« in Zusammenhang mit Exchange) und Verknüpfung mit Outlook Express usw.

Allgemein ist es vorteilhaft, den Seriendruck von Outlook aus zu initialisieren, um das breitmöglichste Spektrum von Feldern und der Datenauswahl zu haben. Genügen jedoch die Informationen, die über eine von Word aus erstellte Verbindung kommen, können Sie mit Word-VBA ein Adressbuch wie folgt für den Seriendruck einbinden.

#### Word 2000 sowie Word 2002

Auch für Outlook-Adressbücher gibt es mehrere mögliche Verbindungsmethoden: einen Konvertierfilter oder das Exchange/Outlook-IISAM, das über DDE, ODBC oder OLE DB (nur in Word 2002/3) angesprochen werden kann.

Word 2000 bedient sich immer des Konvertierfilters, wenn der Benutzer eine Verbindung zu einem Adressbuch herstellt. Eine Aufzeichnung dieses Vorgangs in einem Makro zeigt, dass nicht die `OpenDataSource`, sondern die `UseAddressBook`-Methode in VBA für die Verbindung sorgt:

```
ActiveDocument.MailMerge.UseAddressBook Type:="olk"
```

Es ist auch möglich, den Konvertierfilter über `OpenDataSource` anzusprechen, um so Abfragekriterien festzulegen, was mit `UseAddressBook` nicht möglich ist. Allerdings ist es mit einem Konvertierfilter nur möglich, eine `WHERE`-Klausel zu definieren; eine Liste von Datenfeldern wird für `SELECT` mit einer Fehlermeldung vom Seriendruck abgewiesen. Dazu brauchen Sie eine Textdatei mit der Endung `.olk`. Die Datei darf leer sein, Word wird sowieso die Verbindung direkt mit dem Adressbuch herstellen. Bitte beachten Sie, dass der Dateiname mit Endung in der `FROM`-Klausel stehen muss.

#### WICHTIG

Diese Technik ist übrigens nicht dokumentiert. Sie setzt das Vorhandensein der Datei `SCHDMMAPI.DLL` auf dem Rechner aus, die sich in einem Ordner befinden muss, der in der `PATH`-Angabe des Systems referenziert ist. Diese Datei ist nicht Teil des Office 2003-Lieferumfangs, kann aber von einer Office XP-Installation kopiert werden. Das sagt sich leicht ;-) Was ist, wenn ich keine Office XP-Installation (mehr) habe?

CM: Dann hat man halt Pech...

```
ActiveDocument.MailMerge.OpenDataSource Name:="C:\wdbd\ab.olk", _
SQLStatement:="SELECT * FROM ab.olk WHERE Nachname > 'M'"
```

Diese Methode hat den weiteren Vorteil, dass sie keine temporäre `.olk`-Datei auf der Festplatte zurücklässt, wie es manchmal mit dem Adressbuch der Fall ist. Der Anwender wird jedoch, wie während einer manuellen Verbindung, unter Umständen aufgefordert, ein Profil und einen Kontakte-Ordner auszuwählen.

#### HINWEIS

Die Outlook-Feldnamen sind sprachenspezifisch. Statt `Nachname` setzt die englische Version beispielsweise `Last_Name` ein.

## Datenquellen mit DDE

Lesen Sie bitte die allgemeinen Informationen zu DDE-Verbindungen sowie DDE-Verbindungen im Bezug auf Access und Excel in der Datei *Seriendruck.doc* auf der CD-ROM zum Buch im Ordner *\Beilagen\Zusatzmaterial Seriendruck*.

### Microsoft Access mit DDE

Wir weisen darauf hin, dass sich ein *AutoOpen*-Makro in einer Access-Datenbank (was beispielsweise den Splashscreen der Nordwind-Beispieldatenbank anzeigt) auf die Herstellung einer DDE-Verbindung störend auswirken könnte. Um mit Word 2002/2003 und VBA eine solche Verbindung aufzubauen, brauchen Sie die SubType-Parameter.

#### Einfache Verbindung zu einer Tabelle, ohne SQL-Anweisung

```
ActiveDocument.MailMerge.OpenDataSource Name:="C:\wbdb\Nordwind.mdb", _
Connection:="TABLE Bestellungen", Subtype:= wdMergeSubtypeWord2000 _
```

#### Einfache Verbindung zu einer Abfrage, ohne SQL-Anweisung

```
ActiveDocument.MailMerge.OpenDataSource Name:="C:\wbdb\Nordwind.mdb", _
Connection:="QUERY Die zehn teuersten Artikel"
```

#### Verbindung zu einer Tabelle, mit SQL-Anweisung

Bezeichnen Sie Feldnamen, die spezielle Zeichen enthalten, in der Abfrage mit eckigen Klammern [ ]:

```
ActiveDocument.MailMerge.OpenDataSource Name:="C:\wbdb\Nordwind.mdb", _
SQLStatement:="SELECT [Bestell-Nr], [Kunden-Code], Straße " & "FROM Bestellungen"
```

#### Verbindung zu einer Abfrage, mit SQL-Anweisung

```
ActiveDocument.MailMerge.OpenDataSource Name:="C:\wbdb\Nordwind.mdb", _
SQLStatement:="SELECT EinzelPreis FROM [Die zehn teuersten Artikel]" _
```

#### Ein komplexeres Beispiel mit SQL-Anweisung

Mit DDE steht Ihnen die gesamte Funktionalität von Access-Abfragen zur Verfügung, inklusive benutzerdefinierte VBA-Funktionen, Parameterabfragen, usw. Das folgende Beispiel verbindet zwei Tabellen, benutzt die LEFT-Funktion, um den Kundennamen zu kürzen, und die COUNT-Funktion, um die Anzahl der Bestellungen zu ermitteln.

---

**TIPP** Mehr über die Erstellung von komplexen Abfragen lesen Sie in der Datei *SQL.doc* auf der CD-ROM zum Buch im Ordner *\Beilagen\Zusatzmaterial Seriendruck* nach.

---

```
ActiveDocument.MailMerge.OpenDataSource _
    Name:="C:\wdbd\Nordwind.mdb", _
    SQLStatement:="SELECT B.[Kunden-Code], LEFT(K.[Firma],10) AS 'Abkürzung', " & _
        "COUNT(B.[Kunden-Code]) AS 'AnzahlBestellungen'" & _
        " FROM Bestellungen B, Kunden K" & _
        " WHERE B.[Kunden-Code] = K.[Kunden-Code]" & _
        " GROUP BY B.[Kunden-Code], LEFT(K.[Firma],10)"
```

Um zu sehen, wie eine benutzerdefinierte Funktion erstellt und eingebunden wird, fügen Sie Ihrem Datenbankprojekt ein Modul namens »Beispiel« hinzu, und geben Sie dort die folgenden Codezeilen ein:

```
Function left10(s As String) As String
    left10 = Left(s,10)
End Function
```

Diese Funktion ersetzt die LEFT-Funktion im vorherigen Beispiel. Die SQL-Anweisung in Word-VBA sieht dann folgendermaßen aus:

```
ActiveDocument.MailMerge.OpenDataSource _
    Name:="C:\wdbd\Nordwind.mdb", _
    SQLStatement:=" SELECT B.[Kunden-Code], left10(K.[Firma]) AS 'Firma'," & _
        "COUNT(B.[Kunden-Code]) AS 'Orders'" & _
        " FROM Bestellungen B, Kunden K" & _
        " WHERE B.[Kunden-Code] = K.[Kunden-Code]" & _
        " GROUP BY B.[Kunden-Code],left10(K.[Firma])"
```

### Daten aus anderen Datenbanken über Access einbinden

Jet (Access Datenbank-Engine) unterstützt eine spezielle Syntax, die Verbindungen zu anderen, unabhängigen, nicht verknüpften Datenbanken ermöglicht. Diese Verbindungen werden über den Name-Parameter erstellt. Die folgenden Verbindungen sind theoretisch möglich, aber nicht unter allen Konfigurationen erfolgreich. Wir stellen diese Informationen also eher der Vollständigkeit halber zur Verfügung:

- Zu anderen Access-.mdb-Tabellen oder Abfragen, mit der Syntax *[Volle Pfadangabe zur Access-mdb].[Tabellen (oder Abfrage)-Name]*. Tabellen und Abfragen in .mde- oder .adp-Datenbanken werden nicht unterstützt.
- Zu ODBC-Daten mit der Syntax *[ODBC;ODBC Verbindungsstring].[Tabellenname]*. Diese Methode kann jedoch keine Verbindung zu Tabellen in Jet-Datenbanken herstellen.
- Daten, die mit den Jet-IISAMs (Excel, dBase, Paradox, Outlook usw.) ansprechbar sind, mit der Syntax *[IISAM Name;IISAM Verbindungsstring].[Tabellenname]*.

---

**HINWEIS** IISAM = »Installable Indexed Sequential Access Method-Treiber«, was so viel wie installierbare »einfache Datendateien mit Index-Treiber« heißt.

---

Solche Verbindungen können Teil einer Abfrage sein, die an Access über DDE übergeben wird. Auch ODBC sowie OLE DB können damit umgehen, wenn die Datenbanksicherheit diese Zugangsart erlaubt. Falls Sie jedoch Einlog-Informationen bereitstellen müssen, kommt diese Technik weniger in Frage.

Hier ein Beispiel für die Verwendung dieser Methode. Sie haben eine Access-*.mdb*-Datei ohne Inhalt (keine Tabellen oder andere Objekte). Sie können diese Datenbank im Name-Parameter angeben, um Tabellen aus zwei anderen Datenbanken in einer Abfrage zu verknüpfen, und diese Daten in den Seriendruck zu übernehmen:

```
ActiveDocument.MailMerge.OpenDataSource Name:="C:\wbdb\leer.mdb", _
SQLStatement:=" SELECT * " &
" FROM [c:\wbdb\Artikel.mdb].[Artikel] A, " &
" [c:\wbdb\Nordwind.mdb].[Kategorien] K" & _
" WHERE A.[Kategorie-Nr] = K.[Kategorie-Nr]"
```

**HINWEIS** Weitere Beispiele finden Sie in den Abschnitten Microsoft Desktop Datenbank-ODBC-Verbindungen und Verbindungen zu Datenquellen über OLE DB (Word 2002 und 2003).

**ACHTUNG** Word 2003 scheint mehr Probleme zu haben, Abfragen zu benutzen, die auf externen Tabellen basieren.

### Microsoft Access-Sicherheit bei DDE-Verbindungen

Vom Sicherheitsstandpunkt aus gesehen, ist der Ablauf mit einer DDE-Verbindung genauso, als würden Sie die Datenbank direkt mit Access öffnen.

Falls die Datenbank mit einem Passwort gesichert ist, erscheint eine Aufforderung. Es ist **nicht** möglich, dieses in einem Parameter der OpenDataSource-Methode an Access zu übermitteln.

Wenn die Sicherheit über eine Arbeitsgruppendatei (Systemdatenbank) verwaltet wird, öffnet Access die Datenbank nur, wenn diese als die standardmäßige Arbeitsgruppendatenbank bestimmt wurde. In diesem Fall erscheint eine Aufforderung zur Eingabe des Benutzernamens und des Passworts. Auch dann kann die Verbindung nur zu Stande kommen, wenn der Benutzername die notwendigen Berechtigungen für die Tabellen und Abfragen hat, die in den Parametern Connection und SQLStatement vorkommen.

### Microsoft Excel mit DDE

#### Eine Arbeitsmappe einbinden

Wenn der Name-Parameter eine Arbeitsmappe (*.xls*-Datei) enthält, blendet Word eine Aufforderung ein, worin die Arbeitsmappe, das Arbeitsblatt oder ein Excel-Bereich festgelegt werden muss:

```
ActiveDocument.MailMerge.OpenDataSource Name:="C:\WBDB\Bestellungen.xls"
```

**HINWEIS** Für Word 2002/2003 ist die Methode mit Subtype:= wdMergeSubTypeWord2000 zu ergänzen.

Das Dialogfeld bietet nur *Gesamtes Tabellenblatt*, den Namen des ersten Blatts und benannte Bereiche auf diesem Blatt an. Es ist nicht möglich, ein anderes Tabellenblatt oder einen Bereich auf einem anderen Tabellenblatt zu bestimmen. Dies liegt wohl daran, dass zu der Zeit, als diese Verbindungsmethode entwickelt wurde, eine Excel-Datei nur ein Tabellenblatt enthalten konnte.

Die einzige andere Methode, einen Bereich – auf dem ersten Tabellenblatt wohlgemerkt – festzulegen, ist die Angabe einer RnCn-Referenzadresse, wie beispielsweise R1C1:R6C6. Schwierigkeiten können auftreten, wenn Sie einen Bereich angeben, der größer ist als die vorhandenen Datenbereiche im Tabellenblatt.

Wenn der Bereichsname unbekannt ist, nimmt die Verbindung den standardmäßigen Eintrag *Gesamtes Tabellenblatt* in der jeweiligen lokalen Sprache. Dies bedeutet, die Methode ist sprachenunabhängig, auch wenn Sie die Bezeichnung für ein anderes Land nicht wissen oder berücksichtigt haben.

### Eine Verbindung ohne Aufforderung herstellen

Legen Sie einfach, wie oben beschrieben, den Namen des ersten Tabellenblatts oder eine Bereichsadresse auf dem ersten Tabellenblatt für den Connection-Parameter fest:

```
ActiveDocument.MailMerge.OpenDataSource Name:="C:\wdbd\Bestellungen.xls", _
    Connection:="R1C1:R6C6"
ActiveDocument.MailMerge.OpenDataSource Name:="C:\wdbd\Bestellungen.xls", _
    Connection:="Best1R5"
ActiveDocument.MailMerge.OpenDataSource Name:="C:\wdbd\Bestellungen.xls", _
    Connection:="Gesamtes Tabellenblatt"
```

## Datenquellen mit ODBC

Bei ODBC-Verbindungen erwartet Word für den Connection-Parameter einen *ODBC-Connection-String*.

Jeder, der schon mal versucht hat, eine ODBC-Verbindung über VBA (entweder für den Seriendruck oder eine Datenbank-Feldfunktion) herzustellen, hat wahrscheinlich gewisse Schwierigkeiten gehabt, unter anderem diese:

- Bei mit dem Makrorekorder aufgezeichneten Code gelingt es nicht immer, die Verbindung herzustellen.
- Es ist nicht einfach, Dokumentationen über ODBC-Connection-Strings, und wie sie in Word einzusetzen sind, zu finden.

### HINWEIS

Mehr über ODBC und DSNs erfahren Sie in der Datei *ODBC.doc* auf der CD-ROM zum Buch im Ordner *\Beilagen\Zusatzmaterial Seriendruck*.

## Anmerkungen zur Sicherheit von Logon-Angaben

Diese Bemerkungen gelten auch für andere Verbindungsarten.

Verbindungen zu Datenbanken wie Access, SQL Server u.Ä. erfordern allgemein Logon-Angaben, meistens als UserID mit Kennwort. Unter gewissen Umständen (beispielsweise Access über eine DDE-Verbindung), wenn diese im Connection String fehlen, blendet die Datenbank-Anwendung eine Eingabeaufforderung ein, um die benötigten Angaben abzufragen. In solchen Fällen ist es nicht notwendig, diese im Dokument oder im Code festzuschreiben. ►

Dies ist jedoch nicht immer der Fall, und dann ist es nicht möglich, zu kontrollieren, was passiert, wenn diese Angaben in der `OpenDataSource`-Methode fehlen. Die Datenbank könnte beispielsweise ein Datalink-Dialogfeld einblenden, das nicht nur die Eingabe dieser Angaben, sondern auch die Änderung der Verbindung erlaubt.

Es ist offensichtlich keine gute Idee, UserID und Kennwort in einem Dokument, DSN oder anderen Datei festzuschreiben. Falls der SQL Server mit »Integrated Security« unter Windows benutzt wird, kann das Problem gelöst werden, indem dies im Connection-String festgelegt wird.

Sonst bleibt nur die Möglichkeit, eine eigene Eingabeaufforderung einzusetzen, und den Connection-String dynamisch aufzubauen. Sie müssen sich jedoch bewusst sein, dass Word diese Angaben vermutlich im Dokument festschreiben wird, und sie über die `ConnectionString`-Eigenschaft oder aus einem im XML-Format gespeicherten Dokument gelesen werden können. Nur durch die Umwandlung in ein normales Word-Dokument können diese Informationen daraus entfernt werden. Sie sollten auch dafür sorgen, dass der Anwender das Dokument nicht zwischendurch speichern darf.

## Microsoft Desktop Datenbank-ODBC-Verbindungen

Die Microsoft ODBC Desktop-Datenbanktreiber sind für das Lesen von und Schreiben in einigen weit verbreiteten Datenbankformaten, inklusive Access, Paradox, dBase und Textdateien geeignet. Sie basieren alle auf der Microsoft Jet-Datenbank-Engine und dessen IISAM-Technologie. Diese gemeinsame Grundlage bedeutet, dass einige Verhaltensmuster und Schlüsselwörter für alle Datenbanken gelten. Von besonderem Interesse ist, dass Sie für alle den Jet SQL-Dialekt benutzen können.

Ein anderer Treiber wird für Microsoft Visual FoxPro installiert, der nicht auf der Jet IISAM-Technologie basiert und der den FoxPro SQL-Dialekt implementiert.

Weitere Einzelheiten zu diesen Treibern finden Sie in der MDAC SDK-Dokumentation auf MSDN (<http://www.microsoft.com/downloads/results.aspx?productID=&freetext=mdac&DisplayLang=en>).

### Access-ODBC-Connection-Strings

**Tabelle 6.30** Schlüsselwörter für Access-ODBC-Verbindungen

Schlüsselwort	Beschreibung	Beispielwerte
DBQ=	Legt die Datenbank fest. Wird es weggelassen, wird der Benutzer aufgefordert, eine Datenbank auszuwählen.	Northwind pubs
DefaultDir=	Ordner, worin sich die Datenbank befindet.	
Driver=	ODBC-Treibername	Üblicherweise: "MS Access-Treiber (*.mdb)"
DriverID=	Legt den »untergeordneten« IISAM-Teiler fest.	25

**Tabelle 6.30** Schlüsselwörter für Access-ODBC-Verbindungen (Fortsetzung)

Schlüsselwort	Beschreibung	Beispielwerte
ExtendedAnsiSQL=	<p>Jet 4.0. Ermöglicht den Einsatz des ANSI SQL-92 Standards, was unter anderem die erlaubten Platzhalterzeichen in SQL-Anweisungen bestimmt.</p> <p>Ist <b>ExtendedAnsiSQL=0</b>, steht das »?« für ein beliebiges Zeichen und »*« für keine oder mehrere Zeichen.</p> <p><b>ExtendedAnsiSQL=1</b> bedeutet, dass »_« für ein beliebiges Zeichen und »%« für keine oder mehrere Zeichen steht.</p>	<p>0 (default – disable)</p> <p>1</p>
FIL=	Dateityp	MS Access
PWD=	Kennwort, sofern erforderlich	
SystemDB	Pfadname der Systemdatenbank, wenn vorhanden	
UID=	BenutzerID, sofern erforderlich	Admin (default)

### Benutzer- oder System-DSN mit Datenbankaufforderung

Wenn keine Datenbank im DSN oder im Connection-Parameter steht, blendet Word eine Aufforderung ein:

```
ActiveDocument.MailMerge.OpenDataSource Name="", _
    Connection:="DSN=wbdb-access-us-kdb;", _
    SQLStatement:"SELECT * FROM 'Bestellungen'"
```

Einen Startpfad festlegen:

```
ActiveDocument.MailMerge.OpenDataSource Name="", _
    Connection:="DSN= wbdb-access-us-kdb;DefaultDir=c:\wbdb", _
    SQLStatement:"SELECT * FROM 'Bestellungen'"
```

Ob die Daten in einer Tabelle oder in einer Abfrage zu finden sind, wird mit **TABLE Tabellename** bzw. **TABLE Abfragename** bestimmt. Der Ausdruck **QUERY Abfragename** wird nicht erkannt.

```
ActiveDocument.MailMerge.OpenDataSource Name="", _
    Connection:="DSN= wbdb-access-us-nwdb;", _
    SQLStatement:"TABLE [Liste der aktuellen Artikel]"
```

### Benutzer- oder System-DSN: Ohne Datenbankaufforderung

Alle Verbindungsparameter können in **OpenDataSource** festgelegt werden. Das folgende Beispiel setzt den von Office installierten standardmäßigen Access-DSN für deutschsprachige Umgebungen ein: *Microsoft Access-Datenbank*.



```
ActiveDocument.MailMerge.OpenDataSource Name:="C:\wdbd\Nordwind.mdb", _
Connection:="DSN=Microsoft Access-Datenbank;" & _
"DBQ=C:\wdbd\Nordwind.mdb;" & _
"DriverId=25;FIL=MS Access;MaxBufferSize=2048;PageTimeout=5;", _
SQLStatement:="SELECT * FROM 'Bestellungen' WHERE 'Kunden-Code'='CHOPS'",
SQLStatement1:=""
```

### Access-Sicherheitsüberlegungen

Access/Jet stellt zwei Möglichkeiten für die Sicherung einer Datenbank zur Verfügung. Wurde die Datenbankdatei mit einem Kennwort versehen, erscheint eine Aufforderung, wenn Word versucht, die Verbindung herzustellen (das Kennwort für die Beispieldatenbank ist *mkw*).

```
ActiveDocument.MailMerge.OpenDataSource Name:="", _
Connection:="DSN=wdbd-access-us-kdb;DBQ=c:\wdbd\Artikelkw.mdb;" & _
SQLStatement:="SELECT * FROM 'Artikel'"
```

Das PWD-Schlüsselwort mit dem Kennwort hinzufügen, um die Aufforderung zu unterbinden:

```
ActiveDocument.MailMerge.OpenDataSource Name:="", _
Connection:="DSN=wdbd-access-us-kdb;DBQ=c:\wdbd\Artikelkw.mdb;PWD=mkw", _
SQLStatement:="SELECT * FROM 'Artikel'"
```

Falls die Sicherheit durch eine System- (Arbeitsgruppen-) Datenbank verwaltet wird, muss ein Benutzername mit Kennwort im Connection-Parameter stehen:

```
ActiveDocument.MailMerge.OpenDataSource Name:="", _
Connection:="DSN=wdbd-access-us-kdb;DBQ=c:\wdbd\Artikelsd.mdb;" & _
& "SystemDB=c:\wdbd\Secured.mdw;UID=wb;PWD=sdkw", _
SQLStatement:="SELECT * FROM 'Artikel'"
```

Im Gegensatz zum SQL Server wird das Kennwort im DSN gespeichert, sofern bei der Konfiguration ein Kennwort eines Benutzer- oder System-DSNs definiert ist:

```
ActiveDocument.MailMerge.OpenDataSource Name:="", _
Connection:="DSN=wdbd-access-us-sicher;" & _
SQLStatement:="SELECT * FROM 'Artikel'"
```

### Datei-DSN: Mit Datenbankaufforderung

Ein Datei-DSN, der mit dem Dialogfeld zur ODBC-DSN-Konfiguration erstellt wurde, speichert kein Kennwort; Sie müssen es im Connection-String angeben (alternativ kann die DSN-Datei bearbeitet und damit ergänzt werden):

```
ActiveDocument.MailMerge.OpenDataSource Name:="c:\wdbd\wdbd-access-f-sicher.dsn", _
Connection:="FILEDSN=c:\wdbd\wdbd-access-f-sicher.dsn;PWD=sdkw;", _
SQLStatement:="SELECT * FROM 'Artikel'"
```

## Excel-ODBC-Verbindungen

**Tabelle 6.31** Schlüsselwörter für Excel-ODBC-Verbindungen

Schlüsselwort	Beschreibung	Beispielwerte
DBQ=	Legt das Verzeichnis (Excel 3 und Excel 4) bzw. die Arbeitsmappe (Excel 5 und später) fest.	
DefaultDir=	Verzeichnis, worin sich die Arbeitsmappe befindet	
Driver=	ODBC-Treibername	Normalerweise: "MS Excel-Treiber (*.xls)"
DriverID=	Legt den IISAM-»Untertreiber« für Dateien einer spezifischen Version von Excel fest.	534 (Excel 3) 278 (Excel 4) 22 (Excel 5, 7) 790 (Excel 97 und später)
FIL=	Dateityp	Excel 3.0 Excel 4.0 Excel 5.0 Excel 7.0 Excel 97
FirstRowHasNames=	Auf »1« festlegen, wenn die erste Reihe der Tabelle die Feldnamen enthält.	
MaxScanRows=	Legt die Anzahl an Reihen fest, die der Treiber testen soll, um den Datentyp einer Spalte zu bestimmen.	Standardwert ist »8«
PWD=	Kennwort, sofern benötigt	
UID=	BenutzerID, sofern benötigt	Admin (Standardwert)

Geben Sie im Name-Parameter einen Dateinamen an, lassen aber SQLStatement leer, erscheint eine Aufforderung für den Datei- sowie Tabellennamen:

```
ActiveDocument.MailMerge.OpenDataSource Name:="C:\wdbd\Bestellungen.xls",
Connection:="DSN=Excel Files;DBQ=C:\wdbd\Bestellungen.xls;DriverId=790;" & _
"MaxBufferSize=2048;PageTimeout=5;", SQLStatement:=""
```

Steht im Connection-Parameter ein DSN, und SQLStatement ist nicht leer, aber ein Dateiname wurde weder in der Connection noch im Name-Parameter festgelegt, erscheint eine Aufforderung für die Arbeitsmappendatei. Versuchen Sie aber, auch das Schlüsselwort Defaultdir in der Connection anzugeben, schlägt die Verbindung fehl:

```
ActiveDocument.MailMerge.OpenDataSource Name:="", _
Connection:="DSN=Excel Files;DriverId=790;" & _
"MaxBufferSize=2048;PageTimeout=5;", _
SQLStatement:="SELECT * FROM 'Bestellungen'"
```

Sind `DefaultDir` in der Connection und der Pfadname zur Arbeitsmappe im SQLStatement vorhanden, müsste die Verbindung klappen:

```
ActiveDocument.MailMerge.OpenDataSource Name="",
Connection:="DSN=Excel Files;DriverId=790;DefaultDir=c:\wdbd" & _
"MaxBufferSize=2048;PageTimeout=5;", _
SQLStatement:="SELECT * FROM 'Bestellungen.xls'.'Bestellungen'"
```

Es ist auch möglich, sämtliche Informationen im DSN zu speichern:

```
ActiveDocument.MailMerge.OpenDataSource Name="", _
Connection:="DSN=wdbd-excel-us;", _
SQLStatement:="SELECT * FROM 'Bestellungen'"
```

Sie können Tabellen aus mehreren Dateien verknüpfen. Es ist auch möglich, den Namen eines Tabellenblatts oder eines Bereichs als »Tabellenname« anzugeben. In diesem Beispiel befindet sich der nicht näher qualifizierte Bereich *'Best1R5'* in der im DSN enthaltenen Tabelle:

```
ActiveDocument.MailMerge.OpenDataSource Name="", _
Connection:="DSN=wdbd-excel-us;", _
SQLStatement:=" SELECT * FROM 'c:\wdbd\Bestellungen2.xls'.'Bestellungen','Best1R5'"
```

Auch die Referenzadressenart *A1:Xn* wird für Bereiche auf dem **ersten** Tabellenblatt erkannt, *RIC1:RnCn*-Referenzadressen hingegen nicht:

```
ActiveDocument.MailMerge.OpenDataSource Name="", _
Connection:="DSN=wdbd-excel-us;", _
SQLStatement:=" SELECT * FROM 'A1:E10'"
```

### ODBC-Verbindungen für Textdateien

Für eine ODBC-Verbindung zu einer Textdatei wird ein Teil der benötigten Information im DSN und ein anderer in der *Schema.ini*-Datei gespeichert, die sich im gleichen Ordner wie die Textdatei befindet. Wenn es um das Feldtrennzeichen geht, ist ODBC viel strikter als Word, und muss es immer der *Schema.ini* entnehmen. Es ist nicht möglich, ein anderes Datensatztrennzeichen als den Wagenrücklauf (CR) festzulegen.

Dies ist der Grund, warum Word seit Word 97 solche Mühe mit ODBC-Verbindungen zu zeichenge-trennten Textdateien hat. Der von Office installierte DSN für Textdateien legt ein Semikolon fest.

**Tabelle 6.32** ODBC-Schlüsselwörter für den Textdatei-ODBC-Treiber

Schlüsselwort	Beschreibung	Beispielswerte
DBQ= oder DefaultDir=	Verzeichnis, worin sich die Textdatei befindet. Üblicherweise werden für eine ODBC-Verbindung die Datenquellendatei(en) in der SQL-Anweisung aufgeführt; sie dürfen aber auch im <b>Name</b> -Parameter erscheinen. Bei dem Versuch, eine Datenquelle mit <b>DBQ</b> festzulegen, wird ODBC die Angabe wie einen Pfadnamen behandeln und die Verbindung wird fehlschlagen.	

**Tabelle 6.32** ODBC-Schlüsselwörter für den Textdatei-ODBC-Treiber (Fortsetzung)

Schlüsselwort	Beschreibung	Beispielswerte
Driver=	ODBC-Treibername	Normalerweise "MS Text-Treiber (*.txt, *.csv)"
DriverID=	Interne Desktop-Treiber-ID	27
FIL=	Dateityp	Text
MaxScanRows=	Legt die Anzahl der Datensätze fest, die ODBC für das Ermitteln des Datentyps testet.	

### Schema.ini

Auch Microsoft Jet bedient sich *Schema.ini*-Dateien, um sich die Eigenschaften von importierten oder verknüpften Textdatenquellen zu merken. Es könnte abweichende Schlüsselwörter für die Datentypen einsetzen.

**Tabelle 6.33** Einträge einer *Schema.ini*-Datei für Textdateien

Schlüsselwort	Beschreibung	Beispielswerte
[filename]	Hält den Namen des Abschnitts in <i>Schema.ini</i> fest, worin die Informationen für die Textdatei aufgelistet sind.	[a.txt]
CharacterSet=	OEM (DOS) oder ANSI (Windows). Unter Umständen wird auch eine Codepage akzeptiert, wie CharacterSet=1252 Solche Angaben können jedoch verloren gehen, wenn der DSN mit den Werkzeugen des ODBC-Administrators bearbeitet wird.	OEM ANSI
Coln=	Legt einen Namen, einen Datentyp und die Anzahl an Zeichen einer Spalte (Feld) fest. <b>Col&lt;number&gt;=&lt;Feldname&gt; &lt;Datentyp&gt; &lt;Optional AnzZeichen&gt;</b> Erlaubte Werte für <Datentyp> in ODBC sind: <b>Char</b> (für Jet: <b>Text</b> ) <b>Float</b> (für Jet: <b>Double</b> ) <b>Integer</b> (für Jet: <b>Short</b> ) <b>LongChar</b> (für Jet: <b>Memo</b> ) <b>Date</b>	Col1=K Integer Col2=T Char Width 255
ColNameHeader=	Geben Sie »Yes« ein, wenn die erste Reihe die Spaltenüberschriften (Feldnamen) enthält.	Yes No
Format=	Die Art der Textdatei. Für zeichengetrennte können Sie ein einziges Zeichen im Character-, Dezimal- oder Hex-Format festlegen. Beispiel: <b>DELIMITED(,)</b>	FixedLength TabDelimited CSVDelimited Delimited()
MaxScanRows=	Legt die Anzahl der Datensätze fest, die ODBC für das Ermitteln des Datentyps testet.	

Tabelle 6.33 Einträge einer *Schema.ini*-Datei für Textdateien (Fortsetzung)

Schlüsselwort	Beschreibung	Beispielswerte
TextDelimiter=	Nicht dokumentiertes Schlüsselwort, das das Festlegen eines anderen Texttrennzeichens statt des Anführungszeichens " ermöglicht. Dieser Eintrag könnte verloren gehen, wenn der DSN mit den Werkzeugen des ODBC-Administrator bearbeitet wird.	

### Verbindungen mit Benutzer-/System-DSN

Der DSN des folgenden Beispiels legt Schlüsselwörter wie DefaultDir, DriverID usw. fest:

```
ActiveDocument.MailMerge.OpenDataSource Name="", _
Connection:="DSN=wbdb-txt-us;"
SQLStatement:=" SELECT t.* FROM 'Bestellungen-tab-1-fn.txt' t"
```

Sie können auch mehrere Textdateien verbinden, die jedoch alle im durch DefaultDir spezifizierten Ordner liegen müssen:

```
ActiveDocument.MailMerge.OpenDataSource Name="", _
Connection:="DSN=wbdb-txt-us-1;"
SQLStatement:=" SELECT "*" & _
" FROM 'Bestellungen-tab-1-fn.txt' B," & _
" 'Kunden-semikolon-1-fn.txt' K" & _
" WHERE B.'Kunden-Code' = K.'Kunden-Code'"
```

### Verbindung mit Datei-DSN

Wie oben, aber mit einem Datei-DSN:

```
ActiveDocument.MailMerge.OpenDataSource Name:="c:\wbdb\wbdb-txt-f.dsn", _
Connection:="FILEDSN=c:\wbdb\wbdb-txt-f.dsn;"
SQLStatement:=" SELECT "*" & _
" FROM 'Bestellungen-tab-1-fn.txt' B," & _
" 'Kunden-semikolon-1-fn.txt' K" & _
" WHERE B.'Kunden-Code' = K.'Kunden-Code'"
```

## Andere Datenbankarten

### SQL Server-Benutzer- und System-DSN

#### Vertrauenswürdige Verbindung – minimale Version

Der folgende Beispielcode müsste funktionieren, egal ob im DSN eine vertrauenswürdige oder nicht vertrauenswürdige Verbindung bestimmt ist, weil der Standardwert für Trusted\_Connection in einem Connection-String Yes ist und dieser Wert über die Angabe im DSN Vorrang hat:

```
ActiveDocument.MailMerge.OpenDataSource Name="", _
Connection:="DSN=wbdb-sqlserver-us;"
SQLStatement:="SELECT a.* FROM pubs.dbo.authors a"
```

### Vertrauenswürdige Verbindung – vervollständigte Version

Dieses Beispiel setzt voraus, dass die TCP/IP-Netzwerkbibliothek in Gebrauch ist

```
ActiveDocument.MailMerge.OpenDataSource Name="", _
    ConfirmConversions:= False, Format:=wdOpenFormatAuto, _
    Connection:= "DSN=wbdb-sqlserver-us;" & _
        "Network=DBMSSOCN;Address=mserver,1433;Trusted_Connection=Yes", _
    SQLStatement:= "SELECT a.* FROM pubs.dbo.authors a", SQLStatement1:=""
```

### Keine vertrauenswürdige Verbindung – minimale Version

```
ActiveDocument.MailMerge.OpenDataSource _
    Name="", _
    Connection:= "DSN= wbdb-sqlserver-us;" & _
        "UID=mname;PWD=mkennwort;Trusted_Connection=No", _
    SQLStatement:= "SELECT a.* FROM pubs.dbo.authors a"
```

### Keine vertrauenswürdige Verbindung – vervollständigte Version

```
ActiveDocument.MailMerge.OpenDataSource Name="", _
    ConfirmConversions:= False, Format:=wdOpenFormatAuto, _
    Connection:= "DSN=wbdb-sqlserver-us;UID=mname;PWD=mkennwort;" & _
        "Network=DBMSSOCN;Address=mserver,1433;Trusted_Connection=No", _
    SQLStatement:= "SELECT a.* FROM pubs.dbo.authors a", SQLStatement1:=""
```

### SQL Server-Datei-DSN

Wenn Sie in Word einen Datei-DSN benutzen, muss auf folgende Punkte geachtet werden:

- Ein Dateiname muss im Name-Parameter der OpenDataSource-Methode stehen. Logisch, aber nicht zwingend wäre der Pfadname des DSN.
- Anstelle von DSN muss im Connection-String das Schlüsselwort `FILEDSN` vorhanden sein.
- Obwohl der Pfadname der Datei-DSN im Name-Parameter steht, muss er auch im Connection-String vorkommen.

### Vertrauenswürdige Verbindung – minimale Version

```
ActiveDocument.MailMerge.OpenDataSource Name="c:\wbdb\wbdb-sqlserver-f.dsn", _
    Connection:= "FILEDSN=c:\wbdb\wbdb-sqlserver-f.dsn;", _
    SQLStatement:= "SELECT a.* FROM pubs.dbo.authors a"
```

### Vertrauenswürdige Verbindung – vervollständigte Version

```
ActiveDocument.MailMerge.OpenDataSource Name="c:\wbdb\wbdb-sqlserver-us.dsn", _
    ConfirmConversions:= False, Format:=wdOpenFormatAuto, _
    Connection:= "FILEDSN=c:\wbdb\wbdb-sqlserver-f.dsn;" & _
        "Network=DBMSSOCN;Address=mname,1433;Trusted_Connection=Yes", _
    SQLStatement:= "SELECT a.* FROM pubs.dbo.authors a", SQLStatement1:=""
```

## Nicht vertrauenswürdige Verbindung – minimale Version

```
ActiveDocument.MailMerge.OpenDataSource Name:="c:\wdbd\wdbd-sqlserver-f.dsn", _
    Connection:= "FILEDSN= c:\wdbd\wdbd-sqlserver-f.dsn;" & _
        "UID=mname;PWD=mkenwort;Trusted_Connection=No", _
    SQLStatement:= "SELECT a.* FROM pubs.dbo.authors a"
```

## Nicht vertrauenswürdige Verbindung – vervollständigte Version

```
ActiveDocument.MailMerge.OpenDataSource Name:="c:\wdbd\wdbd-sqlserver-f.dsn", _
    ConfirmConversions:= False, Format:=wdOpenFormatAuto, _
    Connection:= "FILEDSN= c:\wdbd\wdbd-sqlserver-f.dsn;UID=mname;" & _
        "PWD=mkenwort;Network=DBMSSOCN;Address=msserver,1433;" & _
        & Trusted_Connection=No", _
    SQLStatement:= "SELECT a.* FROM pubs.dbo.authors a", SQLStatement1:=""
```

## dBase-ODBC-Verbindungen

Probleme mit Verbindungen zu dBase-*.dbf*-Dateien könnten von der Namensfestlegung relevanter *.inf*-Dateien stammen. Manchmal kann ein Umbenennen helfen.

Tabelle 6.34

## dBase-ODBC-Schlüsselwörter

Schlüsselwort	Beschreibung	Beispielswerte
CollatingSequence	Bestimmt die Sortierreihenfolge.	ASCII International
DBQ=		
DefaultDir=	Verzeichnis, worin sich die <i>.dbf</i> -Datei befindet.	
Driver=	ODBC-Treibername	Normalerweise "Microsoft dBase-Treiber (*.dbf)"
DriverID=	Legt den IISAM-»Untertreiber« fest	21 – dBase III 277 – dBase IV 533 – dBase 5.0
Exclusive=	Datenbank exklusiv sperren oder nicht	0 – not exclusive 1 – exclusive
FIL=	Dateityp	dBase III dBase IV dBase 5.0
MaxScanRows=	Legt die Anzahl der Datensätze fest, die ODBC für das Ermitteln des Datentyps testet	

## Verbindungen mit Benutzer-/System-DSN

Der DSN des folgenden Beispiels legt Schlüsselwörter wie DefaultDir, DriverID usw. fest:

```
ActiveDocument.MailMerge.OpenDataSource Name="", _
    Connection:="DSN=wdb-dbase3-us;", _
    SQLStatement:=" SELECT * FROM 'Bestldb3.dbf'"
```

Sie können auch mehrere Dateien verbinden, die jedoch alle im durch DefaultDir spezifizierten Ordner liegen müssen. **Warnung:** dBase III hat Begrenzungen bei der Namenlänge und dem Zeichensatz.

```
ActiveDocument.MailMerge.OpenDataSource Name="", _
    Connection:="DSN=wdb-dbase3-us;", _
    SQLStatement:=" SELECT * &
        " FROM 'Bestldb3.dbf' B," & _
        " 'Kundndb3.dbf' K" & _
        " WHERE B.'Kunden_Cod' = K.'Kunden_Cod'"
```

## Verbindungen mit Datei-DSN

Wie oben, aber mit einem Datei-DSN:

```
ActiveDocument.MailMerge.OpenDataSource Name="c:\wdb\wdb-dbase3-f.dsn", _
    Connection:="FILEDSN=c:\wdb\wdb-dbase3-f.dsn;", _
    SQLStatement:=" SELECT * &
        " FROM 'Bestldb3.db' B," & _
        " 'Kundndb3.db' K" & _
        " WHERE B.'Kunden_Cod' = K.'Kunden_Cod'"
```

## FoxPro-ODBC-Verbindungen

Angaben zur Erstellung eines FoxPro-DSNs stehen in der Datei *ODBC.doc* auf der CD-ROM zum Buch im Ordner \Beilagen\Zusatzmaterial Seriendruck beschrieben.

Tabelle 6.35 FoxPro-ODBC-Schlüsselwörter

Schlüsselwort	Beschreibung	Beispielswerte
BackgroundFetch=	Fortfahren, ohne zu warten, bis alle Datensätze geholt wurden. Für den Seriendruck wahrscheinlich irrelevant.	No Yes
Collate=	Bestimmt die Sortierreihenfolge.	Machine General German – phone book
Deleted=	Datensätze mit einbeziehen, die als »gelöscht« markiert sind.	Yes No
Driver=	ODBC-Treibername	Für gewöhnlich: "Microsoft Visual FoxPro-Treiber"
Exclusive=	Exklusiv sperren oder nicht – gilt nur für .dbc-Datenbanken	Yes No



Tabelle 6.35 FoxPro-ODBC-Schlüsselwörter (Fortsetzung)

Schlüsselwort	Beschreibung	Beispielswerte
Null=	(Irrelevant für Leseoperationen)	
SourceDB=	Für .dbf-Dateien, der Pfadname des Ordners, worin sie sich befinden. Für .dbc-Datenbanken, der Pfadname der .dbc-Datei.	
SourceType=	DBF für einzelne DBF-Dateien DBC für FoxPro-Datenquellen, die auf mehr als eine DBF verweisen.	DBF DBC

## Verbindung zu einer Datenbankdatei (DBC) mit Benutzer-/System-DSN

```
ActiveDocument.MailMerge.OpenDataSource Name="", _
Connection:="DSN=Visual FoxPro Database;SourceType=DBC;" & _
"SourceDB=c:\wdbd\fp\nw.dbc;Deleted=No;Exclusive=No;", _
SQLStatement:=" SELECT * FROM 'Bestellungen' B"
```

## Verbindung zu einem Verzeichnis von DBF mit Benutzer-/System-DSN

```
ActiveDocument.MailMerge.OpenDataSource Name="", _
Connection:="DSN=Visual FoxPro Tables;SourceType=DBF;" & _
"SourceDB=c:\wdbd\fp;Deleted=No;Exclusive=No;", _
SQLStatement:=" SELECT *" & _
" FROM 'Bestellungen' B," & _
" 'Kunden' K" & _
" WHERE B.'Kunden-Code' = K.'Kunden-Code'"
```

## Verbindung zu einer Datenbankdatei (DBC) mit Datei-DSN

```
ActiveDocument.MailMerge.OpenDataSource Name:="c:\wdbd\wdbd-foxpro-dbc-f.dsn", _
Connection:="FILEDSN=c:\wdbd\wdbd-foxpro-dbc-f.dsn;", _
SQLStatement:=" SELECT *" & _
" FROM 'Bestellungen' B," & _
" 'Kunden' K" & _
" WHERE B.'Kunden-Code' = K.'Kunden-Code'"
```

## Verbindung zu einem Verzeichnis von DBF mit Datei-DSN

```
ActiveDocument.MailMerge.OpenDataSource Name:="c:\wdbd\wdbd-foxpro-dbf-f.dsn", _
Connection:="FILEDSN=c:\wdbd\wdbd-foxpro-dbf-f.dsn;", _
SQLStatement:=" SELECT *" & _
" FROM 'Bestellungen' B," & _
" 'Kunden' K" & _
" WHERE B.'Kunden-Code' = K.'Kunden-Code'"
```

## Oracle-ODBC-Verbindungen

Nur die unabdingbaren Schlüsselwörter werden in Tabelle 6.36 aufgeführt. Weitere Informationen finden Sie in der Hilfe zum Treiber im ODBC-Manager. Die Schlüsselwörter für Oracle 7 können, mit Vorbehalten, auch eine Verbindung zu Oracle 8i herstellen.

**Tabelle 6.36** Oracle 7-ODBC-Schlüsselwörter für den Microsoft-Treiber

Schlüsselwort	Beschreibung	Beispielwerte
Driver=	ODBC-Treibername	Für gewöhnlich: "Microsoft ODBC für Oracle"
Server=	SQL*Net-Name. Das, was hier funktioniert, kommt wahrscheinlich auf der Netzwerkumgebung an. Mit Oracle 8i und später scheint ein Oracle Net8-Servicename wirksam zu sein, wie <i>oracle.mserver</i>	
PWD=	(Wird bei der Konfiguration des DSN nicht aufgezichnet)	tiger
UID=	Benutzer-LoginID	Peter

Die Lage für den Oracle 8i-ODBC-Treiber ist weniger klar, da die in der Registry gespeicherten Namen nicht mit denen der Connection-Strings übereinstimmen. Sie können die Schlüsselwörter aus Tabelle 6.37 ausprobieren.

**Tabelle 6.37** Oracle 8i-ODBC-Schlüsselwörter

Schlüsselwort	Beschreibung	Beispielwerte
Driver=	ODBC-Treibername	Für gewöhnlich: "ODBC für Oracle"
DBQ=	Der Name des Servers	<i>oracle.mserver</i>
PWD=	(Wird bei der Konfiguration des DSN nicht aufgezichnet)	tiger
UID=	Benutzer-LoginID	Peter

## Verbindung zu Oracle mit dem Microsoft-Treiber und Benutzer-/System-DSN

```
ActiveDocument.MailMerge.OpenDataSource Name:="", _
Connection:="DSN=wbdb-oracle7-us;PWD=tiger;", _
SQLStatement:="SELECT * FROM DEPT", _
subtype:=wdMergeSubTypeWord2000
```

## Verbindung zu Oracle mit dem Oracle-Treiber und Benutzer-/System-DSN

```
ActiveDocument.MailMerge.OpenDataSource Name:="", _
Connection:="DSN=wbdb-oracle8-us;PWD=tiger;", _
SQLStatement:="SELECT * FROM DEPT", _
subtype:=wdMergeSubTypeWord2000
```

## Verbindung zu Oracle mit dem Microsoft-Treiber und Datei-DSN

```
ActiveDocument.MailMerge.OpenDataSource Name:="c:\wdbd\wdbd-oracle7-f.dsn", _
Connection:="FILEDSN=c:\wdbd\wdbd-oracle7-f.dsn;PWD=tiger;", _
SQLStatement:="SELECT * FROM DEPT", _
subtype:=wdMergeSubTypeWord2000
```

## Verbindung zu Oracle mit dem Oracle-Treiber und Datei-DSN

```
ActiveDocument.MailMerge.OpenDataSource _
Name:="c:\wdbd\wdbd-oracle8-f.dsn", _
Connection:="FILEDSN=c:\wdbd\wdbd-oracle8-f.dsn;PWD=tiger;", _
SQLStatement:="SELECT * FROM DEPT", _
subtype:=wdMergeSubTypeWord2000
```

## MySQL-ODBC-Verbindungen

Dieser Abschnitt beschreibt den MySQL-ODBC-Treiber Version 3.51.11.

**Tabelle 6.38** MySQL-ODBC-Schlüsselwörter

Schlüsselwort	Beschreibung	Beispielwerte
DATABASE=	Name der Datenbank auf dem angegebenen Server	Nordwind
OPTION=	Ein Bit-Muster, das eine oder mehrere Optionen definiert. Die gewünschten Komponenten werden addiert, um den Wert für die Optionen zu erhalten.	Eine Liste der Werte befindet sich in der MySQL-Dokumentation.
PORT=	MySQL-Port, wenn eine andere Einstellung als die standardmäßige gebraucht wird (3306).	0 = Standardwert (3306)
PWD=	MySQL-Kennwort – MySQL speichert das Kennwort im DSN; es muss also nicht im Connection-String erscheinen.	<i>meinSchlüsselwort</i>
SERVER=	Name des Server-Rechners	<i>localhost</i> oder eine IP-Adresse wie 192.168.16.15
STMT=	Eine SQL-Anweisung für eine Initialisierung vor Ausführung der <b>SQLStatement</b> und <b>SQLStatement1</b> -Parameter.	
UID=	MySQL-BenutzerID – MySQL speichert diese Information im DSN; sie muss also nicht im Connection-String erscheinen.	<i>meinBenutzerID</i>

## Verbindung zu MySQL mit Benutzer-/System-DSN

```
ActiveDocument.MailMerge.OpenDataSource Name:="", _
Connection:="DSN=wdbd-mysql-us;PWD=mkennwort;", _
SQLStatement:= "SELECT left(BestellNr,3) FROM bestellungen WHERE KundenCode = 'ISLAT'"
```

## Verbindung zu MySQL mit Datei-DSN

```
ActiveDocument.MailMerge.OpenDataSource Name:="c:\wdbd\wdbd-mysql-f.dsn", _
    Connection:="FILEDSN=c:\wdbd\wdbd-mysql-f.dsn;PWD=mkennwort;", _
    SQLStatement:= "SELECT left(BestellNr,3) FROM bestellungen WHERE KundenCode = 'ISLAT'"
```

## Verbindungen zu Datenquellen über OLE DB (Word 2002 und 2003)

Word 2002 war die erste Version, durch die OLE DB-Verbindungen zu Datenquellen unterstützt wurden. Da diese Methode jetzt die standardmäßige geworden ist, müssen Sie, wie schon erläutert, unter Umständen andere Verbindungsmethoden mit dem Parameter SubType spezifizieren.

**HINWEIS** Mehr über OLE DB erfahren Sie in der Datei *OLEDB.doc* auf der CD-ROM zum Buch im Ordner *\Beilagen\Zusatzmaterial Seriendruck*.

### Das Format eines OLE DB-Connection-Strings

Das Format eines OLE DB-Connection-Strings ähnelt dem für ODBC: Er besteht aus aneinander gereihten Schlüsselwort=Wert-Paaren. Meistens fängt ein OLE DB-Connection-String mit dem Paar *Provider=Providernamen* an.

Einer der Standard-OLE DB-Provider heißt »Microsoft OLE DB Provider for ODBC«. Theoretisch sollen Sie dadurch eine Verbindung zu jeder vorhandenen ODBC-Datenquelle herstellen können, indem Sie dem Connection-String diesen Providernamen voranstellen: *Provider=MSDASQL.1*.

Geben Sie keinen Providernamen an, nimmt OLE DB den standardmäßigen, was auch der OLE DB-Provider für ODBC ist. Ein gültiger ODBC-Connection-String ist also auch ein gültiger OLE DB-Connection-String, außer er enthält Zeichen, die nicht OLE DB-konform sind. Ein Beispiel:

Wenn Sie mit VBA programmieren und Anführungszeichen in einer Stringvariablen weitergeben müssen, geben Sie zwei Anführungszeichen ein:

```
Jet OLEDB:System Database="c:\wdbd\secured.mdw";
```

Um Anführungszeichen in einem Connection-String einer *.odc*-Datei zu bestimmen, müssen Sie dafür die HTML-Syntax `&quot;` benutzen:

```
Jet OLEDB:System Database=&quot;c:\wdbd\secured.mdw&quot;;
```

### Die OLE DB-Beispiele

Alle vorangegangenen Beispiele in diesem Dokument sollten unter Word 2002/2003 funktionieren und geben Ihnen zumindest eine Methode an die Hand, um eine der Datenquellen für den Seriendruck zu verknüpfen. Es müsste auch möglich sein, die OLE DB-Verbindungen zu einfachen, nicht gesicherten Datenquellen aufzuzeichnen und wiederzuverwenden. Deshalb stellt dieser Abschnitt für verschiedene Datenquellen Beispiele für drei Verbindungsarten vor:

- Im ersten stehen Connection-String und Abfrage im *OpenDataSource*-Befehl; die *.odc*-Datei ist leer (*c:\wdbd\leer.odc*).
- Wo es funktionierte, wurden im zweiten Beispiel Connection-String und Abfrage in eine *.odc*-Datei passenden Namens verlegt.

- Im dritten wird durch Verwendung der *ODC.Database* in der Art der *.odc*-Datei eine Liste der Tabellen und Abfragen der Datenquelle angezeigt. Auch hier scheint es, dass dies nicht für alle Arten von Datenquellen möglich ist.

#### Verbindung zu SQL Server über eine leere *.odc*-Datei

```
ActiveDocument.MailMerge.OpenDataSource Name="c:\wbdb\leer.odc", _
    Connection:="Provider=SQLOLEDB.1;Integrated Security=SSPI;Persist Security " & _
        "Info=False;Initial Catalog=pubs;Data Source=LSL2KS01;", _
    SQLStatement:="SELECT * FROM ""authors""", _
    SubType:=wdMergeSubTypeOther
```

#### Verbindung zu SQL Server über eine *.odc*-Datei

```
ActiveDocument.MailMerge.OpenDataSource Name="c:\wbdb\sqlserver.odc", _
    Connection:="", SQLStatement:="", _
    SubType:=wdMergeSubTypeOther
```

#### SQL Server-Datenbankinhalt vorstellen

```
ActiveDocument.MailMerge.OpenDataSource Name="c:\wbdb\sqlserver_t.odc", _
    Connection:="", SQLStatement:="", _
    SubType:=wdMergeSubTypeOther
```

#### Verbindung zu einer geschützten *.mdb*-Datei (Access) über eine leere *.odc*-Datei

```
ActiveDocument.MailMerge.OpenDataSource Name="c:\wbdb\leer.odc", _
    Connection:="Provider=Microsoft.Jet.OLEDB.4.0;Password=sdkw;" & _
        "User ID=wb;Data Source=c:\wbdb\artikel.sd.mdb;" & _
        "Jet OLEDB:System Database=c:\wbdb\secured.mdw;", _
    SQLStatement:="SELECT * FROM 'Artikel'", _
    SubType:=wdMergeSubTypeOther
```

#### Verbindung zu einer geschützten *.mdb*-Datei (Access) über eine *.odc*-Datei

```
ActiveDocument.MailMerge.OpenDataSource Name="c:\wbdb\access.odc", _
    Connection:="", SQLStatement:="", _
    SubType:=wdMergeSubTypeOther
```

#### Anzeigen des Inhalts einer *.mdb*-Datei (Access)

```
ActiveDocument.MailMerge.OpenDataSource Name="c:\wbdb\access_t.odc", _
    Connection:="", SQLStatement:="", _
    SubType:=wdMergeSubTypeOther
```

### Verbindung zu einer komplexeren Jet-Abfrage über eine leere .odc-Datei

Beachten Sie, dass man bei OLE DB die Syntax »SELECT <tablealiasname>.\*« nutzen muss:

```
ActiveDocument.MailMerge.OpenDataSource Name="c:\wdbd\leer.odc",
    Connection:="Provider=Microsoft.Jet.OLEDB.4.0;Data Source=c:\wdbd\leer.mdb;", _
    SQLStatement:=" SELECT B.*, A.* " &
        " FROM [c:\wdbd\Nordwind.mdb].[Bestelldetails] B," &
        " [Excel 8.0;HDR=YES;Database=c:\wdbd\artikel.xls;].[Artikel] A", _
    SQLStatement1:=" WHERE B.[Artikel-Nr] = A.[Artikel-Nr]", _
    SubType:=wdMergeSubTypeOther
```

### Verbindung mit einer Excel-Arbeitsmappe über Jet und eine leere .odc-Datei

```
ActiveDocument.MailMerge.OpenDataSource Name="c:\wdbd\leer.odc", _
    Connection:="Provider=Microsoft.Jet.OLEDB.4.0;" &
        "Data Source=c:\wdbd\Bestellungen.xls;" &
        "Extended Properties="""Excel 8.0;HDR=Yes""";", _
    SQLStatement:="SELECT * FROM 'Bestellungen'", _
    SubType:=wdMergeSubTypeOther
```

### Verbindung mit einer Excel-Arbeitsmappe über Jet und eine .odc-Datei

```
ActiveDocument.MailMerge.OpenDataSource Name="c:\wdbd\excel.odc", _
    Connection:="", SQLStatement:="", _
    SubType:=wdMergeSubTypeOther
```

### Anzeigen des Inhalts der Excel-Arbeitsmappe

```
ActiveDocument.MailMerge.OpenDataSource Name="c:\wdbd\excel_t.odc", _
    Connection:="", SQLStatement:="", _
    SubType:=wdMergeSubTypeOther
```

### Verbindung mit einer Textdatei über Jet und eine leere .odc-Datei

```
ActiveDocument.MailMerge.OpenDataSource Name="c:\wdbd\leer.odc",
    Connection:="Provider=Microsoft.Jet.OLEDB.4.0;Data Source=c:\wdbd\;" &
        "Extended properties="""Text;HDR=YES;""";", _
    SQLStatement:=" SELECT * " &
        " FROM 'bestellungen-tab-1-fn.txt'", _
    SubType:=wdMergeSubTypeOther
```

### Verbindung mit einer Textdatei über Jet und eine .odc-Datei

```
ActiveDocument.MailMerge.OpenDataSource Name="c:\wdbd\txt.odc", _
    Connection:="", SQLStatement:="", _
    SubType:=wdMergeSubTypeOther
```

### Anzeigen des Inhalts eines Textdateien-Ordners

Zur Erinnerung: Nur bestimmte Dateitypen werden in der Liste angezeigt.

```
ActiveDocument.MailMerge.OpenDataSource Name="c:\wdbd\txt_t.odc", _
Connection:="", SQLStatement:="", _
SubType:=wdMergeSubTypeOther
```

### Verbindung zu einer Paradox-Tabelle über Jet und eine leere .odc-Datei

```
ActiveDocument.MailMerge.OpenDataSource Name="c:\wdbd\leer.odc", _
Connection:="Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\wdbd;" & _
"Extended Properties=""Paradox 5.X;HDR=YES;"";", _
SQLStatement:=" SELECT * FROM 'Best1pd5'", _
SubType:=wdMergeSubTypeOther
```

### Verbindung zu einer Paradox-Tabelle über Jet und eine .odc-Datei

```
ActiveDocument.MailMerge.OpenDataSource Name="c:\wdbd\paradox.odc", _
Connection:="", SQLStatement:="", _
SubType:=wdMergeSubTypeOther
```

### Anzeigen des Inhalts eines Paradox-Dateien-Ordners

```
ActiveDocument.MailMerge.OpenDataSource Name="c:\wdbd\paradox_t.odc", _
Connection:="", SQLStatement:="", _
SubType:=wdMergeSubTypeOther
```

### Verbindung zu einer dBase-Tabelle über Jet und eine leere .odc-Datei

```
ActiveDocument.MailMerge.OpenDataSource Name="c:\wdbd\leer.odc", _
Connection:="Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\wdbd\db3;" & _
"Extended Properties=""dBASE III;HDR=YES;"";", _
SQLStatement:=" SELECT * FROM 'Best1db3'", _
SubType:=wdMergeSubTypeOther
```

### Verbindung zu einer dBase-Tabelle über Jet und eine .odc-Datei

```
ActiveDocument.MailMerge.OpenDataSource Name="c:\wdbd\dbase.odc", _
Connection:="", SQLStatement:="", _
SubType:=wdMergeSubTypeOther
```

### Anzeigen des Inhalts eines dBase-Dateien-Ordners

```
ActiveDocument.MailMerge.OpenDataSource Name:="c:\wdbd\ibase_t.odc", _
    Connection:="", SQLStatement:="", _
    SubType:=wdMergeSubTypeOther
```

### Verbindung zu einem FoxPro-Ordner mit .dbf-Dateien über eine leere .odc-Datei

```
ActiveDocument.MailMerge.OpenDataSource Name:="c:\wdbd\leer.odc", _
    Connection:="Provider=VFPOLEDB.1;Data Source=C:\WBDB\FP;" & _
        "Collating Sequence=MACHINE", _
    SQLStatement:="SELECT * FROM 'Kategorien'", _
    SubType:=wdMergeSubTypeOther
```

### Verbindung zu einem FoxPro-Ordner mit .dbf-Dateien über eine .odc-Datei

```
ActiveDocument.MailMerge.OpenDataSource Name:="c:\wdbd\foxpro_dbf.odc", _
    Connection:="", SQLStatement:="", _
    SubType:=wdMergeSubTypeOther
```

### Anzeigen des Inhalts eines FoxPro-Ordners mit .dbf-Dateien

```
ActiveDocument.MailMerge.OpenDataSource Name:="c:\wdbd\foxpro_dbf_t.odc", _
    Connection:="", SQLStatement:="", _
    SubType:=wdMergeSubTypeOther
```

### Verbindung zu einer FoxPro-Datenbank (DBC) über eine leere .odc-Datei

```
ActiveDocument.MailMerge.OpenDataSource Name:="c:\wdbd\leer.odc", _
    Connection:="Provider=VFPOLEDB.1;Data Source=C:\WBDB\FP\NW.DBC;" & _
        "Collating Sequence=MACHINE", _
    SQLStatement:="SELECT * FROM 'Kategorien'", _
    SubType:=wdMergeSubTypeOther
```

### Verbindung zu einer FoxPro-Datenbank (DBC) über eine .odc-Datei

```
ActiveDocument.MailMerge.OpenDataSource Name:="c:\wdbd\foxpro_dbc.odc", _
    Connection:="", SQLStatement:="", _
    SubType:=wdMergeSubTypeOther
```

### Anzeigen des Inhalts einer FoxPro-Datenbank (DBC)

```
ActiveDocument.MailMerge.OpenDataSource Name:="c:\wdbd\foxpro_dbc_t.odc", _
    Connection:="", SQLStatement:="", _
    SubType:=wdMergeSubTypeOther
```



### Verbindung zu einer Oracle-Datenbank über eine leere .odc-Datei

```
ActiveDocument.MailMerge.OpenDataSource Name="c:\wbdb\leer.odc", _
    Connection:="Provider=MSDAORA.1;Password=tiger;User ID=scott;" & _
        "Data Source=wbdb.lingmoor.local;", _
    SQLStatement:="SELECT * FROM ""DEPT""", _
    SubType:=wdMergeSubTypeOther
```

### Verbindung zu einer Oracle-Datenbank über eine .odc-Datei

```
ActiveDocument.MailMerge.OpenDataSource Name="c:\wbdb\oracle.odc", _
    Connection:="", SQLStatement:="", _
    SubType:=wdMergeSubTypeOther
```

### Anzeigen des Inhalts einer Oracle-Datenbank

```
ActiveDocument.MailMerge.OpenDataSource Name="c:\wbdb\oracle_t.odc", _
    Connection:="", SQLStatement:="", _
    SubType:=wdMergeSubTypeOther
```

## Zusammenfassung

In diesem Kapitel wurde ein Überblick über das Objektmodell von Word vermittelt. Es wurden Grundlagen des programmtechnischen Umgangs mit den Hauptobjekten vermittelt und deren wichtigste Eigenschaften und Methoden näher vorstellt.

- Als erstes wurde das *System*-Objekt (Seite 189 ff.) und das *Application*-Objekt (Seite 191 ff.) vorgestellt.
- Anschließend wurden die wichtigsten Hauptobjekte wie das *Selection*- (Seite 200 ff.), *Document*- (Seite 202 ff.), *Template*- (Seite 214 ff.), *Range*- (Seite 219 ff.), *Find/Raplace*- (Seite 233 ff.), *Style*- (Seite 254 ff.) und *Bookmark*-Objekt (Seite 271 ff.) erläutert.
- In den weiteren Abschnitten standen die Tabellen (Seite 280 ff.), die Nummerierungen (Seite 309 ff.), die Feldfunktionen (Seite 320 ff.) und die Grafik-Objekte (Seite 334 ff.) im Vordergrund des Interesses.
- Das Wissen zum Einrichten eines Dokuments basiert auf dem *Section*- (Seite 357 ff.), *Story-Range*- (Seite 359 ff.), *PageSetup*- (Seite 363 ff.) und *HeaderFooter*-Objekt (Seite 371 ff.).
- Vertiefte Informationen zu den Themen wie Lange Dokumente (Seite 377 ff.), Formularfelder (Seite 381 ff.) und den Seriendruck (Seite 388 ff.) runden dieses Kapitel ab.

Basierend auf der Erfahrung der Autoren in den Newsgruppen wurden Aspekte der Benutzerschnittstelle aufgezeigt, die bekanntlich für Missverständnisse und Probleme sorgen. Entsprechende Vorschläge zur Umgehung dieser Probleme und zur Automatisierung derselben sind auch vorhanden.



## Kapitel 7

# Ereignisse in Word

### In diesem Kapitel:

Auto-Makros	432
Ereignisse auf Dokumentebene	434
Ereignisse auf Applikationsebene	436
Klassenmodul einrichten und initialisieren	438
Übersicht über die verfügbaren Ereignisse	441
Seriendruckereignisse	454
Zusammenfassung	461

Wenn Sie mit Word arbeiten und Dokumente erstellen, bearbeiten und schließen, laufen im Hintergrund die verschiedensten Ereignisse ab. Auch wenn Sie die Einfügemarke verschieben oder zwischen Dokumenten wechseln, werden Ereignisse ausgelöst.

Einige dieser Ereignisse sind für Sie als Anwender sichtbar und behandeln die Formatierung und Darstellung der eingegebenen Texte, andere laufen im Hintergrund und erledigen bestimmte Aufgaben.

Mit Hilfe von VBA können Sie auf eine ganze Reihe dieser Word-internen Ereignisse reagieren, sie abfangen und mit ihnen interagieren sowie um eigene Abläufe und Aktionen erweitern. Sie können auch einzelne Ereignisse auslösen und so weitere Aktionen anstoßen. Dadurch können Sie gezielt bestimmte Abläufe verbessern und an Ihre Anforderungen anpassen.

Diese Ereignisse lassen sich grob in folgende Klassen einteilen:

- **Befehlsergebnisse**

Diese Ereignisse werden ausgelöst, wenn Sie einen Word-Befehl oder eine Word-Funktion ausführen, indem Sie z.B. im Menü *Datei* den Befehl *Öffnen* aufrufen oder in der Symbolleiste *Format* das Symbol für Fett anklicken. Gleichzeitig gibt es Befehlsergebnisse, die beim Klick auf einen Symbolleisteneintrag (Klassenmodul-Ereignis `CommandBarButton_Click`) oder bei der Auswahl eines Eintrags in einem Symbolleistenauswahlfeld (Klassenmodul-Ereignis `CommandBarComboBox_Change`) ausgelöst werden. Weitere Informationen dazu finden Sie in Kapitel 18 sowie in Kapitel 9.

- **Ereignisse auf Dokumentebene**

Diese Ereignisse werden beim Erstellen, Öffnen oder Schließen von Dokumenten ausgeführt (siehe den Abschnitt »Ereignisse auf Dokumentebene« in diesem Kapitel).

- **Ereignisse auf Programmebene (Applikationsebene)**

Die Ereignisse auf Programmebene werden unabhängig von bestimmten Dokumenten oder Dokumentvorlagen ausgelöst. Zu diesen Ereignissen gehören ebenfalls jene, die beim Erstellen oder Schließen von Dokumenten ausgelöst werden. Zusätzliche Ereignisse betreffen die Änderung an Dokumenten und Fenstern, die Serienbriefferstellung und den Umgang mit XML-Dokumenten (siehe den Abschnitt »Ereignisse auf Applikationsebene« in diesem Kapitel).

- **Automatisch ausgeführte Makros**

Neben diesen Ereignissen gibt es zusätzlich noch fünf spezielle Auto-Makros, die automatisch ausgeführt werden, wenn Sie eine der damit verbundenen Aktionen ausführen (siehe den Abschnitt »Auto-Makros« in diesem Kapitel).

In diesem Kapitel erfahren Sie, welche Ereignisse Word 2003 auf Dokument- und Anwendungsebene besitzt, und wie Sie diese nutzen können.

## Auto-Makros

Auto-Makros werden automatisch ausgeführt, wenn ein Makro mit diesem Namen erstellt und die mit dem Auto-Makro verknüpfte Aktion ausgeführt wird.

In Word hat sich in den Versionen Word 2000 bis Word 2003 nichts an diesen Makros geändert. Die Tabelle 7.1 listet die existierenden Auto-Makros auf.

Tabelle 7.1 Auto-Makros in Word

Name des Makros	Ausführung
AutoExec	Beim Starten von Word
AutoNew	Beim Erstellen eines neuen Dokuments
AutoOpen	Beim Öffnen eines vorhandenen Dokuments
AutoClose	Beim Schließen eines Dokuments
AutoExit	Beim Beenden von Word

Die Auto-Makros werden unabhängig von ihrem Speicherort ausgeführt, wenn eine der beiden folgenden Bedingungen erfüllt ist:

- Das Makro besitzt einen der genannten Namen und ist in einem Modul oder im Bereich *This Document* eines Dokuments, einer Dokumentvorlage oder eines geladenen Add-Ins gespeichert.
- Das Modul ist nach dem Namen des Auto-Makros benannt (z.B. »AutoOpen«) und enthält eine Prozedur mit der Bezeichnung *Main*.

**ACHTUNG** Nicht alle Auto-Makros werden in Add-Ins ausgeführt. Dies gilt für die Makros:

- AutoOpen
- AutoNew

Diese werden nicht ausgeführt, da Add-Ins von Word beim Start mitgeladen und keine Dokumente auf ihrer Basis erstellt werden.

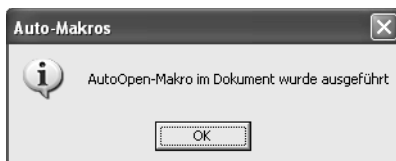
Hingegen wird das Auto-Makro AutoExec in jedem Add-In ausgeführt, so dass mit diesem Makro z.B. sichergestellt werden kann, dass die Schnittstelle zu einem Klassenmodul auf jeden Fall initialisiert wird (siehe den Abschnitt »Klassenmodul einrichten und initialisieren« in diesem Kapitel).

Eine Besonderheit dieser Bedingungen stellt das Makro AutoExec dar, das nur dann automatisch ausgeführt wird, wenn es an einem der folgenden Orte gespeichert wurde:

- in der Vorlage *Normal.dot*,
- in einer Vorlage, die global über das Dialogfeld *Dokumentvorlagen und Add-Ins* geladen wird, oder
- in einer globalen Dokumentvorlage, die im *Startup*-Ordner von Word abgelegt ist.

Ist ein Makro mehrfach (mit Ausnahme des Auto-Makros AutoExec) vorhanden, z.B. in einem Dokument und der verbundenen Dokumentvorlage, wird nur das Auto-Makro aus dem Dokument geladen.


Abbildg. 7.1 Bevorzugtes Ausführen von Makros in Dokumenten



Die Reihenfolge bei der Suche nach einem Makro ist dabei folgende:

1. Aktuelles/zu öffnendes Dokument.
2. Die dem aktuellen Dokument zu Grunde liegende Dokumentvorlage.
3. Die Standarddokumentvorlage *Normal.dot*.
4. Die Add-Ins in der Reihenfolge, wie sie geladen werden.

**HINWEIS** Sofern die Sicherheitseinstellungen das Ausführen von Makros (ggf. auf Nachfrage) erlauben, werden die Makros ausgeführt.

Wenn Sie das Laden von Auto-Makros in einzelnen Fällen unterbinden möchten, halten Sie beim Öffnen der Datei die -Taste gedrückt.

Alternativ können Sie in einem Makro, das ein Auto-Makro auslöst, indem es z.B. ein neues Dokument erstellt, das Ausführen aller Auto-Makros mit folgendem Befehl unterbinden:

```
Application.AutomationSecurity
```

bzw.

```
WordBasic.DisableAutoMacros
```

Weitere Informationen mit Beispielen finden Sie in Kapitel 6.

## Ereignisse auf Dokumentebene

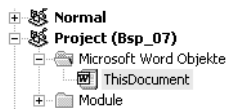
Neben den Auto-Makros gibt es auf Dokumentebene vergleichbare Ereignisse, mit denen auf das Öffnen, Erstellen oder Schließen von Dokumenten reagiert werden kann.

Tabelle 7.2 Ereignisse auf Dokumentebene

Name des Ereignisses	Aurufende Aktion
Document_New()	Erstellen eines neuen Dokuments
Document_Open()	Öffnen eines vorhandenen Dokuments
Document_Close()	Schließen eines Dokuments

Im Gegensatz zu den Auto-Makros, die an verschiedenen Stellen im Dokument oder der Dokumentvorlage gespeichert sein können, erfordern die Dokument-Ereignisse als Speicherort den Bereich *ThisDocument* in der Dokumentvorlage, die dem Dokument zu Grunde liegt. Dieses Modul finden Sie im Visual Basic-Editor, wenn Sie das Projekt der Dokumentvorlage aufrufen und den Eintrag *Microsoft Word Objekte* öffnen.

Abbildg. 7.2 Speicherort für die Ereignisse auf Dokumentebene



**WICHTIG**

Bei diesem Speicherort handelt es sich nicht um ein Standardmodul, sondern um ein spezielles Klassenmodul, das für jedes Dokument und jede Dokumentvorlage bereits vorhanden ist und immer den Namen *ThisDocument* besitzt.

Weitere Informationen zu diesem Klassenmodul finden Sie in Kapitel 2 sowie in Kapitel 19.

Da sowohl die Dokumentvorlage wie auch das Dokument dieses Klassenmodul *ThisDocument* besitzen, können beide Klassenmodule die Ereignisse auf Dokumentebene auslösen.

Wenn Sie also in beiden Klassenmodulen (dem eines Dokuments und dem der zu Grunde liegenden Dokumentvorlage) das Ereignis `Document_Close()` definieren, werden auch beide Ereignisse ausgeführt. Dabei wird zuerst das Ereignis in der Dokumentvorlage ausgelöst, bevor das gleichnamige Ereignis im Dokument selbst ausgelöst wird.

Wenn Sie zusätzlich, wie bei den Auto-Makros, auch in der Standarddokumentvorlage *Normal.dot* Dokument-Ereignisse definieren, werden diese nur dann ausgelöst, wenn das jeweilige Dokument direkt auf dieser Vorlage basiert. Bei Dokumenten mit eigenen Dokumentvorlagen werden die Dokument-Ereignisse in der *Normal.dot* nicht ausgelöst.

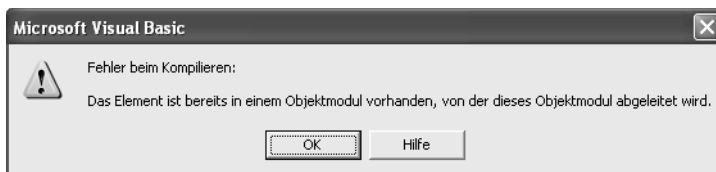
Definieren Sie zusätzlich zu den Dokument-Ereignissen entsprechende Auto-Makros, werden diese in folgender Reihenfolge ausgelöst (z.B. beim Öffnen eines Dokuments):

- Auto-Makro: `AutoOpen()`
- Dokument-Ereignis in der Dokumentvorlage: `Document_Open()`
- Dokument-Ereignis im Dokument: `Document_Open()`

**WICHTIG**

Wenn Sie die gleichen Ereignisse in einem Dokument und der Dokumentvorlage definieren, müssen Sie diese unbedingt vom Typ `Private` deklarieren, da Sie andernfalls eine Fehlermeldung wegen eines Namenskonfliktes erhalten. Dieser Fehler tritt auf, da Prozeduren und Makros, die nicht explizit als `Private` deklariert werden, automatisch vom Typ `Public` sind und auch außerhalb des Moduls oder Klassenmoduls gültig sind. Da aber eine Ereignis-Prozedur in einem Dokument nicht den gleichen Namen besitzen darf wie eine gleichnamige **öffentliche** Ereignis-Prozedur der Dokumentvorlage, erhalten Sie die in Abbildung 7.3 gezeigte Fehlermeldung.

**Abbildg. 7.3** Fehlermeldung beim Verwenden einer privaten und einer öffentlichen gleichnamigen Ereignis-Prozedur



Als Beispiel für ein `Document_New()`-Ereignis auf Dokumentebene kopieren Sie das Makro aus Listing 7.1 in das Modul *ThisDocument* einer Dokumentvorlage oder erstellen Sie ein neues Dokument auf der Basis der Beispiel-Dokumentvorlage *Bsp\_07.dot*.



Die Beispiel-Dokumentvorlage *Bsp\_07.dot* finden Sie auf der CD-ROM im Verzeichnis *\Beispiele\Kap07*.

Kopieren Sie die Dokumentvorlage in ein beliebiges Verzeichnis auf Ihrer Festplatte (z.B. in den Ordner »Eigene Dateien« auf dem Desktop) und führen einen Doppelklick auf diese Dokumentvorlage aus. Es wird daraufhin Word gestartet. Wenn die Makro-Sicherheitseinstellung in Word für Makros auf *Mittel* steht (siehe in Kapitel 1 den Abschnitt »Makrosicherheit«), erscheint eine Abfrage, ob die in der Dokumentvorlage enthaltenen Makros ausgeführt werden sollen. Bestätigen Sie diese Abfrage, indem Sie auf *Makros aktivieren* klicken, und es wird ein neues leeres Dokument mit dem Dialogfeld aus Abbildung 7.4 angezeigt.

**Listing 7.1** Ein *Document\_New*-Ereignis im Klassenmodul *ThisDocument* einer Dokumentvorlage

```
Private Sub Document_New()  
' Beispiel zum Document_New-Ereignis im Klassenmodul ThisDocument der Vorlage  
Const c_Title As String = "Dokument-Ereignisse"  
MsgBox "Document_New-Ereignis in Dokumentvorlage wurde ausgelöst" & vbCrLf & _  
"Dokumentvorlage: " & ThisDocument.Name, vbInformation, c_Title & ActiveDocument.Name  
End Sub
```

Wenn Sie den Code im Visual Basic-Editor eingegeben sowie die Dokumentvorlage gespeichert und geschlossen haben und dann ein neues Dokument erstellen, sollte ein ähnliches Dialogfeld wie in Abbildung 7.4 angezeigt werden.

**Abbildg. 7.4** Ausgabe beim Erstellen eines neuen Dokuments basierend auf einer Vorlage mit *Document\_New*-Ereignis



## Ereignisse auf Applikationsebene

Zusätzlich zu den bereits genannten Ereignissen auf Dokumentebene gibt es in Word noch eine Reihe von Ereignissen, die auf Programmebene (Applikationsebene) ausgeführt werden; also Ereignisse, die direkt von Word ausgelöst werden.

Diese Ereignisse auf Applikationsebene lassen sich in folgende Bereiche unterteilen:

- Ereignisse, die sich auf Dokumente beziehen
- Ereignisse, die sich auf Briefe und Sendungen (Serienbriefe) beziehen
- Ereignisse, die sich auf Anwenderaktionen in Word beziehen
- Ereignisse, die sich auf XML-Dateien beziehen
- Ereignisse, die sich auf das E-Porto-Add-In beziehen



**HINWEIS** Die E-Porto-Ereignisse stehen nur zur Verfügung, wenn das Add-In »Elektronisches Porto« installiert ist. Allerdings bieten weder Microsoft noch ein Servicepartner diesen Dienst für den deutschsprachigen Raum an. Aus diesem Grund wird auf diese Ereignisse nicht näher eingegangen.

Wie auch die Dokument-Ereignisse müssen sich die Applikations-Ereignisse in einem Klassenmodul befinden und initialisiert werden. Dabei wird unterschieden, ob Sie das von den Dokumentereignissen bekannte Klassenmodul *ThisDocument* (siehe den Abschnitt »Ereignisse auf Dokumentebene« in diesem Kapitel) oder im Projekt-Explorer des Visual Basic-Editors ein neues Klassenmodul im Bereich *Klassenmodule* (siehe Kapitel 2) anlegen.

Diese beiden Speicherorte für die Applikations-Ereignisse bestimmen den Gültigkeitsbereich der Ereignisse. So werden Ereignisse im Klassenmodul *ThisDocument* nur bei Dokumenten und Dokumentvorlagen ausgeführt, die nicht von Word als Add-In mitgestartet werden (und somit nicht global zur Verfügung stehen). Sollen die Ereignisse global für alle Dokumente und Dokumentvorlagen zur Verfügung stehen, müssen Sie die Ereignisse in einem eigenen Klassenmodul im Bereich *Klassenmodule* definieren.

Hierbei gilt, dass das Klassenmodul *ThisDocument* bereits initialisiert ist, so dass Sie dort nur die Ereignisse definieren müssen, während Sie die Klassenmodule im Bereich *Klassenmodule* erst einrichten und initialisieren müssen (siehe den Abschnitt »Klassenmodul einrichten und initialisieren« in diesem Kapitel).

Ereignisse im Klassenmodul *ThisDocument* werden bei Add-Ins (z.B. Dokumentvorlagen, die im *Startup*-Ordner von Word gespeichert sind) nicht ausgeführt.



Als Test für das unterschiedliche Verhalten können Sie die Dokumentvorlage *Bsp\_07-1.dot* auf der CD-ROM im Verzeichnis *\Beispiele\Kap07* einmal direkt mit einem Doppelklick aufrufen und einmal in den *Startup*-Ordner von Word kopieren und dann Word starten.

Wenn Sie mit einem Doppelklick auf die Dokumentvorlage eine neue Datei erstellen und anschließend über das Symbol *Neues leeres Dokument* oder über den entsprechenden Menübefehl *Datei/Neu* und dann im Aufgabenbereich über den Befehl *Leeres Dokument* weitere Dateien erstellen, werden Ihnen in Dialogfeldern die Aufrufreihenfolgen der Ereignisse mit ihrem Speicherort angezeigt.

Abbildg. 7.5 Auslösereihenfolge der Ereignisse bei nicht global geladener Dokumentvorlage



Wenn Sie die Dokumentvorlage in den *Startup*-Ordner kopieren, Word starten und dann weitere Dateien erstellen, können Sie an den angezeigten Dialogfeldern (Abbildung 7.6) erkennen, dass keine Ereignisse im *ThisDocument*-Klassenmodul ausgelöst werden.

Abbildg. 7.6 Auslösereihenfolge der Ereignisse bei global geladener Dokumentvorlage im *Startup*-Ordner



#### ACHTUNG

Wie Sie an diesem Beispiel sehen, reagieren auf diese Ereignisse auf Applikations-ebene **alle** Dokumente, die geöffnet oder erstellt werden, solange die Dokumentvorlage mit den Ereignissen geladen ist. Dieses betrifft auch die Dokumente, die auf Basis anderer Dokumentvorlagen erstellt werden; unabhängig davon, ob die Dokumentvorlage mit den Ereignissen global oder nicht global geladen wurde.

Aus diesem Grund ist es beim Umgang mit Dokumenten mitunter notwendig, dass Sie in den Applikations-Ereignissen zuerst die zu Grunde liegende Dokumentvorlage des jeweiligen Dokuments abfragen, bevor das Ereignis eine bestimmte Aktion ausführt.

## Klassenmodul einrichten und initialisieren

Im Gegensatz zum Klassenmodul *ThisDocument*, das ja in jedem Dokument und jeder Dokumentvorlage automatisch existiert, müssen Klassenmodule in einem Dokument oder einer Dokumentvorlage erst noch angelegt werden. Anschließend muss das Klassenmodul mit seinen Prozeduren (dazu zählen auch die Ereignisse) über eine Schnittstelle initialisiert werden, bevor die Ereignisse ausgelöst werden können. Das Einrichten eines Klassenmoduls und die Initialisierung wird in diesem Abschnitt beschrieben.

### Klassenmodul einrichten

Klassenmodule erstellen Sie im Visual Basic-Editor ganz normal über den Menüpunkt *Einfügen/Klassenmodul* oder über das gleichnamige Kontextmenü. Geben Sie dem neuen Klassenmodul einen aussagekräftigeren Namen; für die Beispiele verwenden Sie den Namen *EventClassModule*. Wechseln Sie anschließend in das Code-Fenster des Visual Basic-Editors.

Damit Word auf das Klassenmodul und die Programmereignisse später zugreifen kann, muss eine Variable, die das Programm (*Application*) repräsentiert, öffentlich deklariert werden. Damit gleichzeitig die Ereignisse dieses durch die Variable dargestellten Objekts auch öffentlich gemacht werden, müssen Sie das Schlüsselwort *WithEvents* mit angeben. Verwenden Sie für die Variable einen Namen, der das Objekt aussagekräftig widerspiegelt, z.B. »App«. Die Deklarationszeile lautet dann:

**Listing 7.2** Deklaration einer öffentlichen Ereignis-Variablen im Klassenmodul

```
Public WithEvents App As Application
```

Damit haben Sie die Variable *App* vom Typ *Application* mit seinen Ereignissen öffentlich dem System bekannt gemacht.

Wenn Sie in der Objekt-Auswahlliste dann dieses Objekt *App* auswählen, stehen Ihnen in der Prozedur-Auswahlliste alle verfügbaren Ereignisse dieses Objekts zur Verfügung. Wenn Sie einen Prozedur-Eintrag aus der Liste anklicken, wird automatisch eine syntaktisch korrekte Prozedur im Code-Fenster erstellt. So erstellt ein Klick auf den Eintrag *DocumentOpen* die in Listing 7.3 aufgeführte Prozedur.

**Listing 7.3** Automatisch angelegtes Ereignisgerüst zum Ereignis *DocumentOpen*

```
Private Sub App_DocumentOpen(ByVal Doc As Document)
End Sub
```

Somit können Sie über die Einträge in der Prozedur-Auswahlliste bequem alle Ereignisse mit korrekter Syntax erstellen.

Insgesamt stehen für Word 2003 folgende Ereignisse zur Verfügung und werden in der Prozedur-Auswahlliste aufgeführt:

- Applikations-Ereignisse
  - *WindowActivate*
  - *WindowDeactivate*
  - *WindowSize*
  - *WindowBeforeDoubleClick*
  - *WindowBeforeRightClick*
  - *WindowSelectionChange*
  - *Sync*
- Dokument-spezifische Ereignisse
  - *NewDocument*
  - *DocumentOpen*
  - *DocumentChange*
  - *DocumentBeforeClose*
  - *DocumentBeforePrint*
  - *DocumentBeforeSave*
  - *DocumentSync*
  - Serienbrief-spezifische Ereignisse
    - *MailMergeAfterMerge*

- MailMergeAfterRecordMerge
- MailMergeBeforeMerge
- MailMergeBeforeRecordMerge
- MailMergeDataSourceLoad
- MailMergeDataSourceValidate
- MailMergeWizardSendToCustom
- MailMergeWizardStateChange
- XML-spezifische Ereignisse
  - XMLSelectionChange
  - XMLValidationError
- E-Porto-Ereignisse
  - EPostageInsert
  - EPostageInsertEx
  - EPostagePropertyDialog

Diese Ereignisse werden im Abschnitt »Übersicht über die verfügbaren Ereignisse« in diesem Kapitel im Detail behandelt.

## Klassenmodul initialisieren

Damit das System auf die Ereignisse im Klassenmodul reagieren und die entsprechenden Ereignis-Prozeduren ausführen kann, muss das Klassenmodul über die deklarierte Objekt-Variable initialisiert werden.

Dazu müssen Sie in einem normalen Modul, im Beispiel wird `modCLSInit` verwendet, eine globale Objektvariable auf Modulebene deklarieren, die ein neues Instanz-Objekt des Klassenmoduls darstellt. Neben dem Variablennamen müssen Sie dabei das Schlüsselwort `New` vor dem Variablentypen angeben, da nur darüber eine neue Instanz erstellt wird (Listing 7.4). Mit dieser Zuweisung entfällt gleichzeitig der ansonsten notwendige Objektverweis mittels `Set`-Anweisung (siehe das Kapitel 6).

**Listing 7.4**

Globale Deklaration einer neuen Klasseninstanz vom Klassenmodul *EventClassModule*

```
Dim objWord As New EventClassModule
```

Mit dieser Deklaration ist eine neue Objektvariable `objWord` vom Typ des eingerichteten Klassenmoduls `EventClassModule` erstellt worden. Gleichzeitig stehen alle öffentlichen Objektvariablen dieses Klassenmoduls als Eigenschaften der Variablen zur Verfügung. Diese werden z.B. in der Auswahlliste angezeigt, wenn Sie nach dem Variablennamen einen Punkt eingeben. Die bisher einzige öffentliche Objektvariable im Klassenmodul `EventClassModule` ist die Variable `App` (siehe Listing 7.2), so dass in der Auswahlliste nur diese Variable `App` angeboten wird (siehe Abbildung 7.7).

Als nächstes muss die Schnittstelle zum Klassenmodul mit Hilfe dieser Variablen `App` initialisiert werden. Dazu muss über einen Objektverweis auf ein passendes Objekt die Klassenmodulvariable `App` veröffentlicht werden. Da es sich bei den möglichen Ereignissen um Ereignisse auf Programmebene handelt, muss der Objektverweis ebenfalls auf ein Programmobjekt gesetzt werden. Dazu steht in Word nur das Programm selbst zur Verfügung, das im Visual Basic-Editor durch das `Application`-Objekt (siehe auch Kapitel 19) repräsentiert wird.

**Abbildg. 7.7** Auswahlliste mit allen öffentlichen Variablen des Klassenmoduls

```
Set objWord. = Word.Application
```



Damit das Klassenmodul einer Add-In-Vorlage automatisch beim Start von Word initialisiert wird, erfolgt die Objektzuweisung in der Prozedur *AutoExec()*.

**Listing 7.5** Initialisierung eines Klassenmoduls

```
Sub AutoExec()  
    Set objWord.App = Word.Application  
End Sub
```

Nach dem nächsten Start von Word stehen alle definierten Ereignis-Prozeduren aus dem so initialisierten Klassenmodul zur Verfügung und werden ausgelöst.

**ACHTUNG** Wenn Sie Änderungen an den Prozeduren in einem Klassenmodul vornehmen, müssen Sie anschließend die Initialisierungsprozedur erneut ausführen. Denn durch die Änderung am Klassenmodul wird der Objektverweis gelöscht und somit die Ereignisverarbeitung unterbrochen. Gleichzeitig werden durch die erneute Initialisierung die Änderungen im System bekannt gemacht.

## Übersicht über die verfügbaren Ereignisse

In den folgenden Abschnitten werden die einzelnen in Word 2003 verfügbaren Ereignisse auf Applikationsebene vorgestellt.

### WindowActivate

Das Ereignis *WindowActivate* wird ausgeführt, wenn ein Dokumentfenster aktiviert wird, also den Fokus erhält. Dieses ist sowohl beim Wechsel zwischen mehreren geöffneten Dokumenten der Fall als auch beim Erstellen eines neuen Dokuments oder wenn Sie von einem anderen Programm zu dem Dokument wechseln. Beim Wechsel vom Dokument zum Überarbeitungsfenster löst dieses Ereignis ebenfalls aus. Die Parameter sind in Tabelle 7.3 aufgelistet. Die Syntax lautet:

```
Private Sub App_WindowActivate(ByVal Doc As Document, ByVal Wn As Window)
```

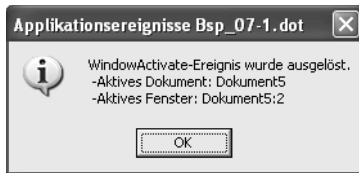
Ein Beispiel für seine Verwendung befindet sich in Listing 7.6; das Ergebnis ist in Abbildung 7.8 ersichtlich.

**Tabelle 7.3** Beschreibung der Parameter der Ereignis-Prozedur *WindowActivate*

Parameter	Beschreibung
Doc	Referenz auf das aktivierte Dokument mit allen Eigenschaften und Methoden eines <b>Document</b> -Objekts.
Wn	Referenz auf das aktivierte Fenster mit allen Eigenschaften und Methoden eines <b>Window</b> -Objekts.

**Listing 7.6** Beispiel zur Ereignis-Prozedur *WindowActivate*, das die jeweiligen Parameter anzeigt

```
Private Sub App_WindowActivate(ByVal Doc As Document, ByVal Wn As Window)
    Dim strMSG As String
    Const c_Title As String = "Applikations-Ereignisse "
    strMSG = "WindowActivate-Ereignis wurde ausgelöst." & vbCrLf & _
        "-Aktives Dokument: " & Doc.Name & vbCrLf & _
        "-Aktives Fenster: " & Wn.Caption & vbCrLf
    MsgBox strMSG, vbInformation, c_Title & ThisDocument.AttachedTemplate.Name
End Sub
```

**Abbildg. 7.8** Ausgabe der Parameter beim Auslösen der Ereignis-Prozedur *WindowActivate*


## WindowDeactivate

Das Ereignis *WindowDeactivate* wird immer dann ausgeführt, wenn ein Dokumentfenster deaktiviert wird, also den Fokus verliert. Dieses ist nicht nur der Fall, wenn Sie zu einem anderen geöffneten Dokument, sondern auch, wenn Sie zu einem anderen Programm wechseln. Die Parameter und das Verhalten sind die gleichen, wie für *WindowActivate*, nur dass die Angaben des deaktivierten Fensters angezeigt werden. Die Syntax lautet:

```
Private Sub App_WindowDeactivate(ByVal Doc As Document, ByVal Wn As Window)
```

## WindowSize

Das Ereignis *WindowSize* wird bei jeder Änderung der Fenstergröße ausgeführt. Dazu gehören das Minimieren, Maximieren und manuelle Ändern der Fenstergröße; aber auch das Verschieben eines Fensters löst dieses Ereignis aus. Die Parameter sind die gleichen wie für *WindowActivate*. Die Syntax lautet:

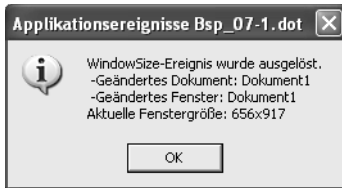
```
Private Sub App_WindowSize(ByVal Doc As Document, ByVal Wn As Window)
```

Ein Beispiel für seine Verwendung befindet sich in Listing 7.7, das Ergebnis ist in Abbildung 7.9 ersichtlich.

**Listing 7.7** Die Ereignis-Prozedur *WindowSize* gibt die Größe des aktuellen Fensters aus, wenn es verkleinert oder vergrößert wird

```
Private Sub App_WindowSize(ByVal Doc As Document, ByVal Wn As Window)
    Dim strMSG As String
    Const c_Title As String = "Applikationsereignisse "
    strMSG = "WindowSize-Ereignis wurde ausgelöst." & vbCrLf & _
        "-Geändertes Dokument: " & Doc.Name & vbCrLf & _
        "-Geändertes Fenster: " & Wn.Caption & vbCrLf
    Select Case Wn.WindowState
    Case wdWindowStateNormal, wdWindowStateMinimize
        strMSG = strMSG & "Aktuelle Fenstergröße (HxW): " & _
            Wn.Height & "x" & Wn.Width & vbCrLf
    End Select
    MsgBox strMSG, vbInformation, c_Title & ThisDocument.AttachedTemplate.Name
End Sub
```

**Abbildg. 7.9** Ausgabe der aktuellen Fenstergröße durch die Ereignis-Prozedur *WindowSize*



## WindowBeforeDoubleClick

Wenn Sie mit der Maus auf einen Text doppelt klicken, wird das Ereignis *WindowBeforeDoubleClick* ausgelöst, bevor eine evtl. mit dem Doppelklick verbundene Aktion ausgeführt wird. Die Parameter sind in Tabelle 7.4 aufgelistet. Die Syntax lautet:

```
Private Sub App_WindowBeforeDoubleClick(ByVal Sel As Selection, Cancel As Boolean)
```

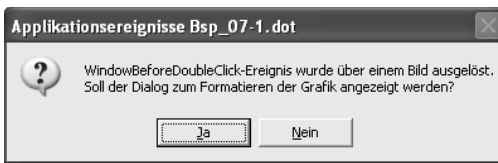
**Tabelle 7.4** Die Parameter der Ereignis-Prozedur *WindowBeforeDoubleClick*

Parameter	Beschreibung
<b>Sel</b>	Referenz auf die Markierung, auf die der Doppelklick erfolgte, mit allen Eigenschaften und Methoden eines <b>Selection</b> -Objekts.
<b>Cancel</b>	Parameter zum Steuern der Ereignisausführung. Standardmäßig wird die Aktion mit <b>Cancel=False</b> ausgeführt; durch Setzen von <b>Cancel=True</b> kann die Aktion unterbunden werden.

Sie können mit diesem Ereignis z.B. den Typ der Markierung abfragen und nur gezielt auf einen Doppelklick auf eine Grafik (z.B. *InLineShape*) reagieren. In dem Beispiel in Listing 7.8 wird geprüft, ob das markierte Objekt ein *InLineShape* ist. Nur für diesen Fall wird gefragt (Abbildung 7.10), ob das standardmäßig angezeigte Dialogfenster zum Formatieren der Grafik angezeigt werden soll oder nicht.

**Listing 7.8** Beispiel zur Ereignis-Prozedur *WindowBeforeDoubleClick*

```
Private Sub App_WindowBeforeDoubleClick(ByVal Sel As Selection, Cancel As Boolean)
    Dim strMSG As String
    Dim ret As Integer
    Const c_Title As String = "Applikations-Ereignisse "
    If Sel.Type = wdSelectionInlineShape Then
        strMSG = "WindowBeforeDoubleClick-Ereignis wurde über einem Bild ausgelöst." _
            & vbCrLf & "Soll der Dialog zum Formatieren der Grafik angezeigt werden?" & vbCrLf
        ret = MsgBox(strMSG, vbQuestion + vbYesNo, c_Title & _
            ThisDocument.AttachedTemplate.Name)
        If ret = vbNo Then
            MsgBox "Dialog wird nicht angezeigt.", vbInformation, "Aktion ausführen"
            Cancel = True
        End If
    End If
End Sub
```

**Abbildg. 7.10** Ausgabe der Aktion der Ereignis-Prozedur *WindowBeforeDoubleClick*

**HINWEIS**

Ein weiteres Beispiel zur Verwendung dieses Ereignisses finden Sie in Kapitel 29.

## WindowBeforeRightClick

Dieses Ereignis *WindowBeforeRightClick* wird ausgelöst, wenn Sie mit der rechten Maustaste auf einen Text oder ein Element im Dokument klicken, bevor die damit verbundene Aktion ausgeführt wird. In den meisten Fällen wird bei einem Rechtsklick ein Kontextmenü angezeigt, über das Sie weitere Aktionen ausführen können. Die Parameter sind die gleichen wie für *WindowBeforeDoubleClick*. Die Syntax lautet:

```
Private Sub App_WindowBeforeRightClick(ByVal Sel As Selection, Cancel As Boolean)
```

Sie können damit aber auch prüfen, ob z.B. der Rechtsklick auf ein Formularfeld ausgeführt wurde und in diesem Fall zusätzliche Menüeinträge in das Kontextmenü einblenden. In Listing 7.9 wird dann ein geändertes Kontextmenü angezeigt, über das Sie den Formularschutz ein- bzw. ausschalten können (Abbildung 7.11).

**Listing 7.9** Die Ereignis-Prozedur *WindowBeforeRightClick* blendet bei Formularfeldern ein eigenes Kontextmenü ein

```
Private Sub App_WindowBeforeRightClick(ByVal Sel As Selection, Cancel As Boolean)
    If ActiveDocument.ProtectionType = wdAllowOnlyFormFields Then
        CreateContextMenu.ShowPopup
        Cancel = True
    End If
End Sub
```



**Listing 7.9** Die Ereignis-Prozedur *WindowBeforeRightClick* blendet bei Formularfeldern ein eigenes Kontextmenü ein (Fortsetzung)

```

ElseIf ActiveDocument.ProtectionType = wdNoProtection Then
    Dim fldFF As FormField
    For Each fldFF In ActiveDocument.FormFields
        If Sel.Range.InRange(fldFF.Range) Then
            CreateContextMenu.ShowPopup
            Cancel = True
        End If
    Next fldFF
End If
End Sub

Function CreateContextMenu() As CommandBar
    Dim cbar As CommandBar
    Dim ctl1 As CommandBarControl
    Dim ctl2 As CommandBarControl
    CustomizationContext = ActiveDocument.AttachedTemplate
    Set cbar = CommandBars("Form Fields")
    cbar.Reset
    With cbar
        Set ctl1 = .FindControl(msoControlButton, 1, "Dokumentschutz aufheben", , True)
        If ctl1 Is Nothing Then
            Set ctl1 = .Controls.Add(msoControlButton, 1, , , True)
        End If
        ctl1.Caption = "Dokumentschutz aufheben"
        ctl1.Tag = "Dokumentschutz aufheben"
        ctl1.OnAction = "UnprotectDoc"
        Set ctl2 = .FindControl(msoControlButton, 1, "Dokumentschutz setzen", , True)
        If ctl2 Is Nothing Then
            Set ctl2 = .Controls.Add(msoControlButton, 1, , , True)
        End If
        ctl2.Caption = "Dokumentschutz setzen"
        ctl2.Tag = "Dokumentschutz setzen"
        ctl2.OnAction = "ProtectDoc"
        If ActiveDocument.ProtectionType = wdNoProtection Then
            ctl1.Enabled = False
            ctl2.Enabled = True
        ElseIf ActiveDocument.ProtectionType = wdAllowOnlyFormFields Then
            ctl1.Enabled = True
            ctl2.Enabled = False
        End If
    End With
    Set CreateContextMenu = cbar
    Set ctl2 = Nothing
    Set ctl1 = Nothing
    Set cbar = Nothing
End Function

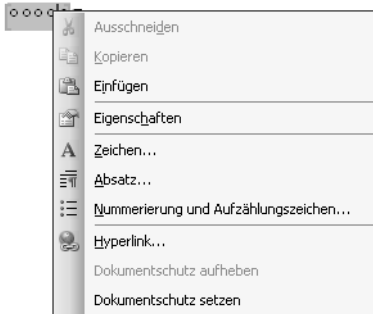
```

Damit das Beispiel aus Listing 7.9 funktioniert, müssen Sie in ein Modul die Prozeduren »ProtectDoc« und »UnprotectDoc« zum Setzen und Aufheben des Formularschutzes kopieren, wie in Listing 7.10.

Listing 7.10 Listing 7.10:Prozeduren zum Setzen und Aufheben des Formularschutzes

```
Public Sub ProtectDoc()
    If ActiveDocument.ProtectionType = wdNoProtection Then
        ActiveDocument.Protect Type:=wdAllowOnlyFormFields, NoReset:=True
    End If
End Sub
Public Sub UnprotectDoc()
    If ActiveDocument.ProtectionType = wdAllowOnlyFormFields Then
        ActiveDocument.Unprotect
    End If
End Sub
```

Abbildg. 7.11 Einblenden eines Kontextmenüs für Formularfelder mittels *WindowBeforeRightClick*



**HINWEIS** Mehr über die Erstellung von Symbolleisten und -Schaltflächen sowie ein weiteres Beispiel zu diesem Ereignis finden Sie in Kapitel 15.

## WindowSelectionChange

Wenn Sie die Einfügemarke mit der Maus an eine andere Stelle im Dokument verschieben oder mit der Maus einen Text bzw. ein Objekt im Dokument markieren, wird das Ereignis `WindowSelectionChange` ausgelöst. Das Ereignis wird auch dann ausgelöst, wenn Sie die Einfügemarke mittels der Pfeil-Tasten bewegen, in einer Tabelle einen Tabulatorsprung (-Taste) einfügen oder in einem geschützten Formular mittels der -Taste bzw. der Tastenkombination + zwischen den einzelnen Feldern wechseln.

Der Parameter ist in Tabelle 7.5 aufgelistet, die Syntax lautet:

```
Private Sub App_WindowSelectionChange(ByVal Sel As Selection)
```

Tabelle 7.5 Der Parameter der Ereignis-Prozedur *WindowSelectionChange*

Parameter	Beschreibung
Sel	Referenz auf die Markierung, auf die die Einfügemarke verschoben wurde, mit allen Eigenschaften und Methoden eines <code>Selection</code> -Objekts.

Sie können z.B. den Typ der Markierung abfragen und nur gezielt auf das Markieren eines Feldes reagieren.

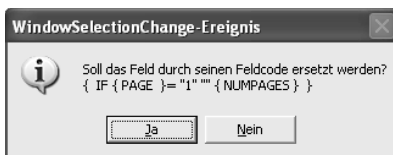
**ACHTUNG** Das Ereignis *WindowSelectionChange* wird nicht ausgelöst, wenn Sie mit der Tastatur die Einfügemarke verschieben, z.B. bei der Texteingabe.

Im Beispiel in Listing 7.11 wird geprüft, ob die Einfügemarke in ein Feld verschoben wurde. Nur für diesen Fall wird gefragt (Abbildung 7.12), ob das entsprechende Feld durch seinen Feldcode ersetzt werden soll oder nicht.

**Listing 7.11** Mit der Ereignis-Prozedur *WindowSelectionChange* werden Felder durch ihren Feldcode ersetzt

```
Private Sub App_WindowSelectionChange(ByVal Sel As Selection)
    Dim strField As String
    Dim ret As Integer
    Dim fldField As Field
    For Each fldField In ActiveDocument.Fields
        On Error Resume Next
        If Not TypeOf fldField Is FormField Then
            ' Feldfunktion ausschalten
            fldField.ShowCodes = False
            strField = fldField.Result
            If Err.Number = 0 Then
                If Sel.Range.InRange(fldField.Result) Then
                    strField = Replace(fldField.Code, Chr(19), "{")
                    strField = Replace(strField, Chr(21), "}")
                    ret = MsgBox("Soll das Feld durch seinen Feldcode ersetzt werden?" & vbCrLf & _
                        "{ " & strField & " }", vbInformation + vbYesNo, _
                        "WindowSelectionChange-Ereignis")
                    If ret = vbYes Then
                        fldField.ShowCodes = True
                        fldField.Select
                        strField = Selection.Range
                        strField = Replace(strField, Chr(19), "{")
                        strField = Replace(strField, Chr(21), "}")
                        Selection.Text = strField
                    End If
                End If
            End If
        End If
    Next fldField
End Sub
```

**Abbildg. 7.12** Ein Feld wird mittels *WindowSelectionChange* durch seinen Feldcode ersetzt



**HINWEIS** Ein weiteres Beispiel für dieses Ereignis befindet sich in Kapitel 9 im Abschnitt über VSTO.

## NewDocument-Ereignis

Jedes Mal, wenn Sie ein neues Dokument erstellen, wird das Ereignis `NewDocument` ausgelöst, das die folgende Syntax hat:

```
Private Sub App_NewDocument(ByVal Doc As Document)
```

Der Parameter ist in Tabelle 7.6 aufgelistet.

**Tabelle 7.6** Der Parameter der Ereignis-Prozedur *NewDocument*

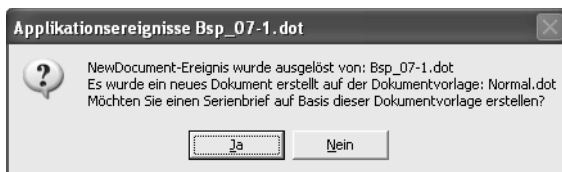
Parameter	Beschreibung
Doc	Referenz auf das neu erstellte Dokument mit allen Eigenschaften und Methoden eines Document-Objekts.

Sie können mit diesem Ereignis z.B. direkt eine Aktion mit dem erstellten Dokument verknüpfen. Wenn es auf einer bestimmten Dokumentvorlage basiert (in Listing 7.12 auf der *Normal.dot*), können Sie z.B. direkt den Assistenten zum Erstellen eines Serienbriefs aufrufen.

**Listing 7.12** Verwenden der Ereignis-Prozedur *NewDocument*, um direkt den Serienbrief-Assistenten aufzurufen

```
Private Sub App_NewDocument(ByVal Doc As Document)
    Dim strMSG As String
    Dim ret As Integer
    Const c_Title As String = "Applikations-Ereignisse "
    strMSG = "NewDocument-Ereignis wurde ausgelöst von: " & _
        ThisDocument.AttachedTemplate.Name & vbCrLf & _
        "Es wurde ein neues Dokument erstellt auf der Dokumentvorlage: " & _
        Doc.AttachedTemplate If Doc.AttachedTemplate = NormalTemplate Then
        strMSG = strMSG & vbCrLf & _
            "Möchten Sie einen Serienbrief auf Basis dieser Dokumentvorlage erstellen?"
        ret = MsgBox(strMSG, vbQuestion + vbYesNo, c_Title & _
            ThisDocument.AttachedTemplate.Name)
        If ret = vbYes And Doc.MailMerge.MainDocumentType = wdNotAMergeDocument Then
            Doc.MailMerge.ShowWizard 1
        End If
    End If
End Sub
```

**Abbildg. 7.13** Erstellen eines Serienbriefdokuments als Beispiel zur Ereignis-Prozedur *NewDocument*



## DocumentOpen

Wenn Sie ein Dokument öffnen, wird das Ereignis DocumentOpen ausgeführt, das die gleichen Parameter wie DocumentNew besitzt und folgende Syntax aufweist:

```
Private Sub App_DocumentOpen(ByVal Doc As Document)
```

Mit diesem Ereignis können Sie z.B. das geöffnete Dokument dahingehend prüfen, ob die Dokumenteigenschaften ausgefüllt sind. In Listing 7.13 wird geprüft, ob der Titel in den Eigenschaften ausgefüllt ist und ggf. das Dialogfeld mit den Dokumenteigenschaften angezeigt ist. Dieses Dialogfeld besitzt keine eigene benannte Konstante, so dass es über die interne Dialognummer »750« geöffnet wird. Zusätzlich wird sichergestellt, dass in den Word-Einstellungen die Optionen *Sicherungskopie immer erstellen* gesetzt und *Speichern im Hintergrund* nicht gesetzt ist.

**Listing 7.13** Prüfen, ob im geöffneten Dokument ein Titel eingetragen ist, und ggf. Anzeige der Dokumenteigenschaften

```
Private Sub App_DocumentOpen(ByVal Doc As Document)
    Dim strMSG As String
    Dim ret As Integer
    Application.Options.BackgroundSave = False
    Application.Options.CreateBackup = True
    If Doc.BuiltInDocumentProperties("Title").Value = "" Then
        On Error Resume Next
        ret = Dialogs(750).Show
        On Error GoTo 0
    End If
End Sub
```

## DocumentChange

Bei jedem Wechsel zwischen geöffneten Dokumenten, aber auch beim Öffnen oder Schließen eines Dokuments, wenn der Fokus zu einem anderen Dokument wechselt, wird das Ereignis DocumentChange ausgeführt mit der folgenden Syntax:

```
Private Sub App_DocumentChange()
```

In Listing 7.14 wird bei jedem Wechsel zwischen geöffneten Dokumenten und wenn ein Dokument geöffnet bzw. geschlossen wird, geprüft, ob die Symbolleiste »Web« sichtbar ist oder sich automatisch eingeblendet hat. Ist dies der Fall, wird diese Symbolleiste wieder ausgeblendet.

**Listing 7.14** Überprüft, ob die Symbolleiste Web angezeigt wird, und schließt diese gegebenenfalls

```
Private Sub App_DocumentChange()
    Dim cbarWeb As CommandBar
    Set cbarWeb = App.CommandBars("Web")
    If cbarWeb.Visible = True Then
        cbarWeb.Visible = False
    End If
End Sub
```

## DocumentBeforeClose

Jedes Mal, bevor das aktuelle Dokument geschlossen werden soll, wird das Ereignis `DocumentBeforeClose` mit der folgenden Syntax ausgeführt. Die Parameter sind in Tabelle 7.7 aufgelistet.

```
Private Sub App_DocumentBeforeClose(ByVal Doc As Document, Cancel As Boolean)
```

Tabelle 7.7 Die Parameter der Ereignis-Prozedur *DocumentBeforeClose*

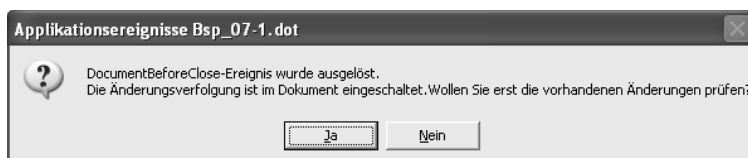
Parameter	Beschreibung
Doc	Referenz auf das zu schließende Dokument mit allen Eigenschaften und Methoden eines <b>Document</b> -Objekts.
Cancel	Parameter zum Steuern der Ereignisausführung. Standardmäßig wird die Aktion mit <b>Cancel=False</b> ausgeführt. Durch Setzen von <b>Cancel=True</b> kann das Ausführen der Aktion verhindert werden.

Mit diesem Ereignis können Sie auf das absichtliche oder unabsichtliche Schließen des aktuellen Dokuments reagieren. In Listing 7.15 wird vor dem Schließen geprüft, ob die Änderungsverfolgung für das Dokument eingeschaltet ist und Überarbeitungsänderungen im Dokument vorhanden sind. Ist dies der Fall, wird der Anwender gefragt (Abbildung 7.14), ob er die Änderungen sehen und prüfen möchte.

Listing 7.15 Überprüft das zu schließende Dokument, ob Überarbeitungsänderungen vorhanden sind

```
Private Sub App_DocumentBeforeClose(ByVal Doc As Document, Cancel As Boolean)
    Dim strMSG As String
    Dim ret As Integer
    Const c_Title As String = "Applikations-Ereignisse "
    strMSG = "DocumentBeforeClose-Ereignis wurde ausgelöst." & vbCrLf
    If Doc.TrackRevisions = True Then
        strMSG = strMSG & "Die Änderungsverfolgung ist im Dokument eingeschaltet."
        If Doc.Revisions.Count > 0 Then
            strMSG = strMSG & "Wollen Sie erst die vorhandene Änderungen prüfen?"
            ret = MsgBox(strMSG, vbQuestion + vbYesNo, c_Title & _
                ThisDocument.AttachedTemplate.Name)
            If ret = vbYes Then
                Doc.ShowRevisions = True
                Cancel = True
                Doc.Revisions(1).Range.Select
            End If
        End If
    End If
End Sub
```

Abbildg. 7.14 Die Nachprüfung beim Schließen des Dokuments hat ergeben, dass darin Revisionen noch vorhanden sind



## DocumentBeforeSave

Das Ereignis DocumentBeforeSave wird jedes Mal, bevor Sie das aktuelle Dokument speichern, ausgeführt. Die Parameter sind in Tabelle 7.8 aufgelistet, die Syntax lautet:

```
Private Sub App_DocumentBeforeSave(ByVal Doc As Document, SaveAsUI As Boolean, _
    Cancel As Boolean)
```

**Tabelle 7.8** Die Parameter der Ereignis-Prozedur *DocumentBeforeSave*

Parameter	Beschreibung
Doc	Referenz auf das aktuelle zu speichernde Dokument mit allen Eigenschaften und Methoden eines <b>Document</b> -Objekts.
SaveAsUI	Parameter zur Anzeige des <i>Speichern unter</i> -Dialogfeldes. Leider wird dieser Parameter entgegen der Onlinehilfe von Word 2003 nicht unterstützt.
Cancel	Parameter zum Steuern der Ereignisausführung. Standardmäßig wird die Aktion mit <b>Cancel=False</b> ausgeführt. Durch Setzen von <b>Cancel=True</b> kann das Ausführen der Aktion verhindert werden.

Mit diesem Ereignis können Sie gezielt vor jedem Speichervorgang eine zusätzliche Aktion ausführen.

### HINWEIS

Dieses Ereignis wird vor jedem Speichervorgang ausgelöst; auch bei der automatischen Speicherung oder wenn Sie in einem Makro ein Dokument über den entsprechenden VBA-Befehl speichern.

Bei der Automatisierung durch das .NET Framework wird der Cancel-Parameter nicht korrekt erkannt. Dieses Problem ist im Knowledge Base-Artikel »BUG: Cancel parameter for Office events is ignored in Visual Studio .NET 2003« (<http://support.microsoft.com/default.aspx?scid=kb;en-us;830519>) beschrieben und wurde in Office 2003 behoben.

In Listing 7.16 wird vor dem Speichern gefragt (Abbildung 7.15), ob eine Sicherungskopie von dem Dokument erstellt werden soll, die sich aus dem Dateinamen und dem Datum mit Uhrzeit zusammensetzt. Wird die Frage verneint, wird das Dokument ganz normal unter seinem Dateinamen gespeichert, andernfalls wird die Sicherungskopie im selben Ordner gespeichert.

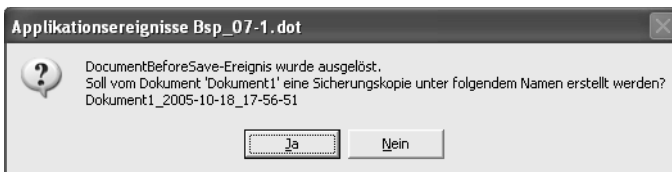
**Listing 7.16** Die Ereignis-Prozedur *DocumentBeforeSave* fragt nach, ob eine Sicherungskopie erstellt werden soll

```
Private Sub App_DocumentBeforeSave(ByVal Doc As Document, SaveAsUI As Boolean, _
    Cancel As Boolean)
    Dim strMSG As String, strName As String, strNameOrig As String
    Dim ret As Integer
    Const c_Title As String = "Applikations-Ereignisse "
    strMSG = "DocumentBeforeSave-Ereignis wurde ausgelöst." & vbCrLf
    If Doc.Type = wdTypeDocument Then
        strName = If(Right(Doc.Name, 4) = ".doc", Replace(Doc.Name, ".doc", _
            Format(Date, "YYYY-mm-dd")) & ".doc", Doc.Name & Format(Date, " _YYYY-mm-dd"))
        strMSG = strMSG & "Soll vom Dokument '" & Doc.Name & _
            "' eine Sicherungskopie unter folgendem Namen erstellt werden?" & vbCrLf & strName
```

**Listing 7.16** Die Ereignis-Prozedur *DocumentBeforeSave* fragt nach, ob eine Sicherungskopie erstellt werden soll (*Fortsetzung*)

```
ret = MsgBox(strMSG, vbQuestion + vbYesNo, c_Title & _
    ThisDocument.AttachedTemplate.Name)
If ret = vbYes Then
    strNameOrig = Doc.Name
    Doc.SaveAs Doc.Path & "\" & strName
    Doc.SaveAs Doc.Path & "\" & strNameOrig
    Cancel = True
End If
End If
End Sub
```

**Abbildung. 7.15** Erstellung einer Sicherungskopie mit dem Ereignis *DocumentBeforeSave*



## DocumentBeforePrint

Das Ereignis *DocumentBeforePrint* wird ausgeführt, bevor ein Dokument ausgedruckt bzw. zum Drucker geschickt wird. Es hat folgende Syntax (die Parameter sind die gleichen wie für *DocumentBeforeClose*):

```
Private Sub App_DocumentBeforePrint(ByVal Doc As Document, Cancel As Boolean)
```

Mit diesem Ereignis können Sie z.B. das Dokument vor dem Ausdruck verändern, indem Sie bestimmte Bereiche aus- oder einblenden. In Listing 7.17 wird das Ereignis dazu verwendet, um vor dem Ausdruck ein Wasserzeichen in Form des Schriftzuges »Kopie« in das Dokument einzufügen (Abbildung 7.16).

**Listing 7.17** Einfügen eines Wasserzeichens mit der Ereignis-Prozedur *DocumentBeforePrint*

```
Private Sub App_DocumentBeforePrint(ByVal Doc As Document, Cancel As Boolean)
    Dim strMSG As String
    Dim ret As Integer
    Dim oHead As HeaderFooter
    Dim shpHead As Shape
    Const c_Title As String = "Applikations-Ereignisse "
    strMSG = "DocumentBeforeClose-Ereignis wurde ausgelöst." & vbCrLf
    strMSG = strMSG & _
        "Soll das Dokument mit einem Wasserzeichen ('Kopie') ausgedruckt werden?"
    ret = MsgBox(strMSG, vbQuestion + vbYesNo, c_Title & _
        ThisDocument.AttachedTemplate.Name)
    If ret = vbYes Then
        Set oHead = ActiveDocument.Sections(1).Headers(wdHeaderFooterFirstPage)
        Set shpHead = oHead.Shapes.AddTextEffect( PresetTextEffect:=msoTextEffect1, _
```



Listing 7.17 Einfügen eines Wasserzeichens mit der Ereignis-Prozedur *DocumentBeforePrint* (Fortsetzung)

```

Text:="Kopie", FontName:="Arial Black", FontSize:=36, FontBold:=msoTrue, _
FontItalic:=msoFalse, Left:=100, Top:=100, Anchor:=oHead.Range)
With shpHead
.Fill.Visible = msoTrue
.Fill.Solid
.Fill.ForeColor.RGB = RGB(255, 255, 255)
.Fill.Transparency = 0#
.Line.Weight = 0.75
.Line.DashStyle = msoLineSolid
.Line.Style = msoLineSingle
.Line.Transparency = 0#
.Line.Visible = msoTrue
.Line.ForeColor.RGB = RGB(0, 0, 0)
.Line.BackColor.RGB = RGB(255, 255, 255)
.LockAspectRatio = msoFalse
.Height = 280
.Width = 320
.Rotation = 330#
.RelativeVerticalPosition = wdRelativeVerticalPositionPage
.Left = CentimetersToPoints(6)
.Top = CentimetersToPoints(7.86)
.LockAnchor = False
.WrapFormat.Type = wdWrapNone
.WrapFormat.Side = wdWrapBoth
End With
End If
End Sub

```

Abbildg. 7.16 Optionales Erstellen eines Wasserzeichens vor einem Dateiausdruck

**HINWEIS**

Mehr über die Automatisierung von Grafiken und WordArt steht in Kapitel 6 beschrieben.

## DocumentSync

Das Ereignis DocumentSync wird ausgeführt, wenn eine lokale Kopie eines Dokuments mit der Version auf einem SharePoint-Server synchronisiert wird. Die Parameter sind in Tabelle 7.9 aufgelistet, die Syntax lautet:

```
Private Sub app_DocumentSync(ByVal Doc As Document, _
    ByVal SyncEventType As Office.MsoSyncEventType)
```

Tabelle 7.9 Die Parameter der Ereignis-Prozedur *DocumentSync*

Parameter	Beschreibung
Doc	Referenz auf das lokale zu synchronisierende Dokument mit allen Eigenschaften und Methoden eines <b>Document</b> -Objektes.
SyncEventType	Parameter, der den Status der Synchronisation wiedergibt. Folgende <b>msoSyncEventType</b> -Werte sind möglich: <b>msoSyncEventDownloadFailed</b> <b>msoSyncEventDownloadInitiated</b> <b>msoSyncEventDownloadNoChange</b> <b>msoSyncEventDownloadSucceeded</b> <b>msoSyncEventOffline</b> <b>msoSyncEventUploadFailed</b> <b>msoSyncEventUploadInitiated</b> <b>msoSyncEventUploadSucceeded</b>

Über den Parameter SyncEventType können Sie den Status der Synchronisation ablesen.

Listing 7.18 Ausgabe der Ereignis-Prozedur *DocumentSync* bei einem Synchronisationsfehler

```
Private Sub App_DocumentSync(ByVal Doc As Document, _
    ByVal SyncEventType As Office.MsoSyncEventType) _
    Dim strMSG As String
    Const c_Title As String = "Applikationsereignisse "
    If SyncEventType = msoSyncEventDownloadFailed Or _
        SyncEventType = msoSyncEventUploadFailed Then _
        strMSG = "Fehler beim Synchronisieren des Dokumentes '" & Doc & "'"
        MsgBox strMSG, vbCritical, c_Title & ThisDocument.AttachedTemplate.Name
    End If
End Sub
```

## Seriendruckereignisse

Neben den Ereignissen des Seriendruck-Aufgabenbereichs, die in Kapitel 6 vorgestellt wurden, sind vier Ereignisse von Interesse, die während des Zusammenführens stattfinden. Sie werden zuerst einzeln vorgestellt, dann anhand eines Beispiels veranschaulicht.

## MailMergeBeforeMerge

Das Ereignis `MailMergeBeforeMerge` tritt ein, wenn Word den Befehl für die Zusammenführung bekommt, aber bevor ein Datensatz zusammengeführt wird. Die Parameter werden in Tabelle 7.10 erläutert. Die Syntax lautet:

```
MailMergeBeforeMerge(ByVal Doc As Document, ByVal StartRecord As Long, _  
ByVal EndRecord As Long, Cancel As Boolean)
```

Tabelle 7.10

Die Parameter der Ereignis-Prozedur *MailMergeBeforeMerge*

Parameter	Beschreibung
<code>Doc</code>	Gibt ein <b>Document</b> -Objekt zurück, das sich auf das Seriendruckhauptdokument bezieht.
<code>StartRecord</code>	Der Indexwert des ersten Datensatzes in der Datenquelle, der im Seriendruck enthalten sein soll.
<code>EndRecord</code>	Der Indexwert des letzten Datensatzes in der Datenquelle, der im Seriendruck enthalten sein soll.
<code>Cancel</code>	Ermöglicht den Abbruch der Zusammenführung, bevor der erste Datensatz zusammengeführt wird. Standardmäßiger Wert ist <b>False</b> . Wird ausdrücklich der Wert <b>True</b> zugewiesen, wird die Ausführung abgebrochen.

`MailMergeBeforeMerge` wird ausschließlich durch Betätigung einer Befehlsoption im letzten Schritt des Assistenten (Seriendruck-Aufgabenbereiches) ausgelöst. Wird der Seriendruck über die Symbolleiste oder die Automatisierungsschnittstelle zusammengeführt, ist das erste Ereignis `MailMergeBeforeRecordMerge`.

## MailMergeBeforeRecordMerge

Das Ereignis `MailMergeBeforeRecordMerge` tritt für jeden einzelnen Datensatz während einer Zusammenführung ein. Der Zeitpunkt liegt nach dem Zugriff auf diesen Datensatz, jedoch vor seiner Zusammenführung mit dem Hauptdokument. Die Syntax lautet:

```
MailMergeBeforeRecordMerge(ByVal Doc As Document, Cancel As Boolean)
```

Die Beschreibungen der Parameter `Doc` und `Cancel` sind die gleichen wie für `MailMergeBeforeMerge`. Wird `Cancel` auf `True` festgelegt, wird lediglich die Zusammenführung dieses einen Datensatzes abgebrochen.

### WICHTIG

Stellen Sie sicher, dass die Methode `MailMerge.Execute` mit dem Argument `Pause:=False` aufgerufen wird. Sonst kann es vorkommen, dass der Seriendruck nach dem ersten zusammengeführten Datensatz ohne Fehlermeldung abbricht.

## MailMergeAfterRecordMerge

Das Ereignis MailMergeAfterRecordMerge tritt nach der Zusammenführung eines einzelnen Datensatzes ein. Die Syntax lautet:

```
MailMergeAfterRecordMerge(ByVal Doc As Document)
```

Der Parameter Doc stellt das Seriendruckhauptdokument dar.

## MailMergeAfterMerge

MailMergeAfterMerge ist das abschließende Ereignis des Seriendrucks und tritt ein, nachdem alle Datensätze erfolgreich zusammengeführt wurden. Die Syntax lautet:

```
MailMergeAfterMerge(ByVal Doc As Document, ByVal DocResult As Document)
```

Der Parameter Doc stellt weiterhin das Seriendruck-Hauptdokument dar. Falls der Seriendruck in ein neues Dokument zusammengeführt wurde (statt direkt auf den Drucker oder als E-Mail), gibt der Parameter DocResult dieses Dokument zurück.

## Beispiel: 1:n-Liste im Seriendruckresultat

Dieses Beispiel veranschaulicht das Zusammenspiel der Ereignisse MailMergeBeforeRecordMerge, MailMergeAfterRecordMerge sowie MailMergeAfterMerge. Es zeigt auf, wie eine Liste 1:n Daten aus einer Datenquelle zusammenstellt und als Tabelle im Seriendruckresultat darstellt.

Das Seriendruck-Hauptdokument ist in Abbildung 7.17 ersichtlich. Alle für die Aufgabe relevanten Seriendruckbefehle befinden sich in der Symbolleiste, die Word-internen Seriendruckschnittstellen sind ausgeblendet.

Als Datenquelle für das Beispiel dient eine zeichengetrennte Textdatei, wie sie beispielsweise ein Großrechner zur Verfügung stellt. Diese Datei enthält die Artikelliste der Firma »Nordwind« mit Informationen über den Lagerbestand. Die Lieferantendaten kommen darin mehrmals vor, einmal je Produkt, das Nordwind von ihnen bezieht.

**Abbildg. 7.17** Das Seriendruck-Hauptdokument ist in englischer Sprache, weil die meisten Lieferanten sich außerhalb Deutschlands befinden

Nordwind GmbH München, den 31. Oktober 2005

Bsp07\_03 Seriendruckereignisse  
 Seriendruck ausführen

«Firma»  
 «Kontaktperson»  
 «Straße»  
 «PLZ» «Ort»  
 «Land»

**MERCHANDISE ORDER**

Dear «Kontaktperson»

Please send us the following articles:

I

For transportation and payment conditions, the agreements in our contract apply.

Die Aufgabe der Automatisierungslösung besteht darin, die Einträge für jeden Lieferanten zu bündeln und nur diejenigen aufzulisten, die bestellt werden müssen. Ein Teil des Seriendruckereignisses ist in Abbildung 7.18 abgebildet. Ein Vergleich der beiden Abbildungen zeigt auf, dass die Tabelle in Abbildung 7.18 an die Stelle der Textmarke in Abbildung 7.17 eingefügt wird.

#### HINWEIS

Eine Diskussion des Seriendruck-Objektmodells befindet sich in Kapitel 6.

**Abbildg. 7.18** Das Ergebnis des Seriendrucks – eine Tabelle als Teil der Prozedur *MailMergeBeforeRecordMerge*

Formaggi Fortini s.r.l.  
 Elio Rossi  
 Viale Dante, 75  
 48100 Ravenna  
 Italien

**MERCHANDISE ORDER**

Dear Elio Rossi

Please send us the following articles:

Produkt	Anzahl
Gorgonzola Telino 12 x 100-g-Packungen	22
Mozzarella di Giovanni 24 x 200 g-Packungen	22

Diese Voraussetzungen bedingen, dass die Datensätze nach Lieferant sortiert sind, wofür die Prozedur, die den Seriendruck ausführt, sorgt, indem der SQL-Anweisung (QueryString-Eigenschaft) eine Order By-Klausel hinzugefügt wird. Jetzt kann durch die Datensätze geschleift werden, um die Anzahl der Einträge jedes Lieferanten zu ermitteln. Da während der Zusammenführung der Seriendruck nur vorwärts durch die Datensätze schreiten kann, muss dies vor der Zusammenführung getan werden. Der Name des Lieferanten sowie die Anzahl seiner vorhandenen Datensätze werden in einem globalen Array festgehalten.

Die Bildung des Arrays findet in der Prozedur *DatenVorbereiten* aus Listing 7.19 statt. Bitte beachten Sie, wie durch die Datensätze geschleift wird. Diese Handlung dauert relativ lange, da Word buchstäblich vom ersten bis zum gewählten Datensatz blättert. Schneller wäre es, eine Datenbankverbindung zur Datenquelle aufzubauen, um die Informationen zusammenzustellen.

**Listing 7.19** Eine Liste der Lieferanten mit der Anzahl ihrer Datensätze zusammenstellen

```
Public Function DatenVorbereiten(doc As Word.Document) As Boolean
    Dim ds As Word.MailMergeDataSource
    Dim lZaehler As Long
    Dim lAnzDS As Long
    Dim lDSZaehler As Long
    Dim lFirmenzaehler As Long
    Dim sFeldInhalt As String

    Set ds = doc.MailMerge.DataSource
    ds.ActiveRecord = wdLastRecord
    lAnzDS = ds.ActiveRecord
    ds.ActiveRecord = wdFirstDataSourceRecord
    lDSZaehler = 0
    lFirmenzaehler = -1
    'Ein Array der Firmen mit der Anzahl der Datensätze bilden
    For lZaehler = 1 To lAnzDS
        If sFeldInhalt <> ds.DataFields("Firma").Value Then
            'Ein anderer Lieferant
            lDSZaehler = 1
            lFirmenzaehler = lFirmenzaehler + 1
            sFeldInhalt = ds.DataFields("Firma").Value
            ReDim Preserve aFIRMEN_DS(1, lFirmenzaehler)
            aFIRMEN_DS(0, lFirmenzaehler) = sFeldInhalt
            aFIRMEN_DS(1, lFirmenzaehler) = lDSZaehler
        Else
            lDSZaehler = lDSZaehler + 1
            aFIRMEN_DS(1, lFirmenzaehler) = lDSZaehler
        End If
        ds.ActiveRecord = wdNextDataSourceRecord
    Next
    ds.ActiveRecord = wdFirstRecord
    If lAnzDS < 1 Then
        DatenVorbereiten = False
    Else
        DatenVorbereiten = True
    End If
End Function
```

Nachdem die Anwendungsereignisse initialisiert wurden, wird der Seriendruck zusammengeführt. Als erstes Ereignis löst dieser MailMergeBeforeRecordMerge aus, dessen Inhalt in Listing 7.20 erscheint. Sie ruft die Funktion *DatenSatzPrüfen* in Listing 7.21 auf. Gibt diese »Wahr« zurück, wird die Zusammenführung dieses Datensatzes unterbunden und Word geht sofort zum nächsten (MailMergeAfterRecordMerge wird nicht ausgelöst).

Listing 7.20 Die Ereignis-Prozedur *MailMergeBeforeRecordMerge*

```
Private Sub wdApp_MailMergeBeforeRecordMerge(ByVal doc As Document, Cancel As Boolean)
    If DatenSatzPrüfen(doc) Then
        Cancel = True
    End If
End Sub
```

Die Prozedur *DatenSatzPrüfen* arbeitet mit dem aktuellen Datensatz. Beim Wechsel der Firma werden die globalen Variablen, worin die Array-Elemente und Artikelliste aufgeführt werden, zurückgestellt. Es wird durch das Array durchgeschleift, bis der Firmenname gefunden wird. Die Anzahl der Datensätze dafür wird festgehalten.

Danach wird kontrolliert, ob der Artikel des aktuellen Datensatzes die Kriterien für eine Bestellung erfüllt:

- Es handelt sich um keinen Auslaufartikel
- Es ist keine Bestellung offen
- Der Lagerbestand ist geringer oder gleich dem Mindestbestand

Ist dies der Fall, werden die Artikelbezeichnung und die Anzahl der Einheiten (110% des Mindestbestands) in der Artikelliste als eine zeichengetrennte Zeichenkette aufgenommen.

Außer es handelt sich um den letzten Datensatz einer Firma *und* die Artikelliste ist nicht leer, wird MailMergeBeforeRecordMerge »Wahr« zurückgegeben. Liegt doch eine Bestellung vor, wird die Liste ins Dokument in den Textmarkenbereich eingefügt und in eine Tabelle umgewandelt.

Listing 7.21 Die 1:n-Listen jedes Lieferanten aufbauen

```
Public Function DatenSatzPrüfen(doc As Word.Document) As Boolean
    Dim sFirma As String
    Dim lZaehler As Long
    Dim ds As Word.MailMergeDataSource

    sFirma = doc.MailMerge.DataSource.DataFields("Firma").Value
    'Die Firmenangabe hat gewechselt; alles zurücksetzen
    If lANZ_FIRMA_DS = 0 Then
        sDatenliste = ""
        lFIRMA_DS_ZAEHLER = 1
        For lZaehler = LBound(aFIRMEN_DS, 2) To UBound(aFIRMEN_DS, 2)
            If aFIRMEN_DS(0, lZaehler) = sFirma Then
                lANZ_FIRMA_DS = aFIRMEN_DS(1, lZaehler)
                Exit For
            End If
        Next
    Else
        lFIRMA_DS_ZAEHLER = lFIRMA_DS_ZAEHLER + 1
    End If
```

**Listing 7.21** Die 1:n-Listen jedes Lieferanten aufbauen (Fortsetzung)

```

Dim bIstAuslauf As Boolean
Dim lBestellt As Long
Dim lLagerBestand As Long
Dim lMindestBestand As Long
Set ds = doc.MailMerge.DataSource
bIstAuslauf = CBool(ds.DataFields("Auslaufartikel").Value)
lBestellt = ds.DataFields("BestellteEinheiten").Value
lLagerBestand = ds.DataFields("Lagerbestand").Value
lMindestBestand = ds.DataFields("Mindestbestand").Value

'Datensatz nur zur Liste hinzufügen, wenn folgende Kriterien wahr sind
If bIstAuslauf = False And lBestellt = 0 And _
    lLagerBestand <= lMindestBestand Then

    sDatenliste = sDatenliste & vbCr & ds.DataFields("Artikelname").Value & _
        "|" & ds.DataFields("Liefereinheit").Value & vbTab & _
        CStr(Int(ds.DataFields("Mindestbestand") * 1.1))
End If

'Wenn es nicht der letzte Datensatz der Firma ist,
'darf der Datensatz nicht zusammengeführt werden
If lFIRMA_DS_ZAEHLER < lANZ_FIRMA_DS Then
    DatenSatzPrüfen = True
Else
    'Beim letzten Datensatz einer Firma wird der Zähler zurückgesetzt.
    lANZ_FIRMA_DS = 0
    'Falls sich Produkte in der Liste befinden, Spaltenüberschriften
    'hinzufügen, Tabelle in dem Hauptdokument erstellen
    'und den aktuellen Datensatz zusammenfügen
    If Len(sDatenliste) > 0 Then
        sDatenliste = "Produkt" & vbTab & "Anzahl" & sDatenliste
        TabelleErstellen doc, sDatenliste
        DatenSatzPrüfen = False
    Else
        'Sind für die Firma keine Produkte aufgelistet, wird
        'der Datensatz nicht zusammengeführt
        DatenSatzPrüfen = True
    End If
End If
End Function

```

Da in diesem Fall der Seriendruck den Datensatz zusammenführen darf, wird das Ereignis *MailMergeAfterRecordMerge* ausgelöst und die Prozedur aus Listing 7.22 ausgeführt. Ihre Aufgabe ist es, das Seriendruck-Hauptdokument wieder in den ursprünglichen Zustand zu versetzen. Die Tabelle mit der Bestellung wird gelöscht und die Textmarke wieder erstellt.

**Listing 7.22** Im Ereignis *MailMergeAfterRecordMerge* wird das Hauptdokument zurückgesetzt

```

Private Sub wdApp_MailMergeAfterRecordMerge(ByVal doc As Document)
    Dim rng As Word.Range
    Dim sBkmName As String

    sBkmName = "Artikelliste"
    Set rng = doc.Bookmarks(sBkmName).Range

```



**Listing 7.22** Im Ereignis *MailMergeAfterRecordMerge* wird das Hauptdokument zurückgesetzt (Fortsetzung)

```
'Tabelle mit den Produkten entfernen und Textmarke
'wieder erstellen, so dass das Dokument wieder in seinem
'ursprünglichen Zustand für den nächsten Datensatz bereitsteht.
rng.Tables(1).Delete
doc.Bookmarks.Add Name:=sBkmName, Range:=rng
End Sub
```

Nachdem alle Datensätze abgearbeitet wurden, kommt das Ereignis *MailMergeAfterMerge* (Listing 7.23) zum Zug. Es teilt dem Anwender mit, dass der Seriendruck abgeschlossen ist, zeigt das Ergebnisdokument an und sorgt dafür, dass der Anwender nicht aufgefordert wird, durch den Seriendruck entstandene Änderungen im Hauptdokument zu speichern.

**Listing 7.23** Abschluss-Handlungen werden durch *MailMergeAfterMerge* durchgeführt

```
Private Sub wdApp_MailMergeAfterMerge(ByVal doc As Document, ByVal DocResult As Document)
    MsgBox "Der Seriendruck wurde zusammengeführt"
    If Not DocResult Is Nothing Then
        DocResult.Activate
    End If
    doc.Saved = True
End Sub
```



Die Beispieldateien *Bsp07\_03.doc* mit dem Beispielcode sowie *Bsp07\_Beiispiel.doc* mit dem Seriendruckresultat finden Sie auf der CD-ROM im Ordner *\Beispiele\Kap07*. Die Seriendruck-Datenquelle *Artikelliste.txt* befindet sich im Ordner *\Datenbank*.

## Zusammenfassung

Ereignisse bieten dem Entwickler die Möglichkeit, auf bestimmte Benutzer-Aktionen zu reagieren. Im Gegensatz zur alten *WordBasic*-Methode, die auf Prozeduren mit internen Befehlsnamen basiert (siehe Kapitel 18), stehen diese Automatisierungscodes auch außerhalb eines Word-VBA-Projekts zur Verfügung. Sie können beispielsweise in COM-Add-Ins sowie VISTO-Projekten benutzt werden.

- Dieses Kapitel stellte auf Seite 434 sowie auf Seite 436 eine Übersicht der Dokument- bzw. Anwendungsereignisse vor.
- Wie das Klassenmodul, das die Ereignisprozeduren enthält, erstellt und initialisiert wird, steht auf Seite 438 ff. beschrieben.
- Die Anwendungs- sowie Dokumentereignisse wurden ab Seite 441 näher erläutert.
- Abschließend, beginnend auf Seite 454, wurden die Seriendruckereignisse diskutiert. Ein praxisnahes Beispiel folgte ab Seite 456.

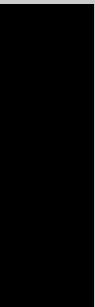


# Teil C

## Steuern und gesteuert werden

### In diesem Teil:

<b>Kapitel 8</b>	Grundlagen der Fernsteuerung	465
<b>Kapitel 9</b>	Word von anderen Umgebungen aus steuern	481
<b>Kapitel 10</b>	Andere Programme aus Word heraus steuern	509
<b>Kapitel 11</b>	Eingebettete OLE-Objekte	539



## Kapitel 8

# Grundlagen der Fernsteuerung

### In diesem Kapitel:

Verweise auf externe Bibliotheken	466
Early versus Late Binding	470
Über das .NET Framework	477
Zusammenfassung	479

Im zweiten Teil dieses Buches wurden das Objektmodell von Word sowie die Dokument- und Programmereignisse näher vorgestellt. Im dritten Teil erläutern wir nun das Zusammenspiel von verschiedenen Applikationen:

- Im aktuellen Kapitel wird aufgezeigt, wie externe Bibliotheken in das Projekt eingebunden und deren Ressourcen genutzt werden.
- In Kapitel 9 wird zunächst aufgezeigt, wie sich Word von außerhalb durch ein anderes Programm steuern lässt. Dabei kann es sich sowohl um ein Modul aus Microsoft Office als auch um eine Eigenentwicklung oder sogar um eine andere Word-Instanz handeln.
- Das Kapitel 10 ist dem Fernsteuern wichtiger Microsoft Office-Programme aus Word heraus gewidmet. Es werden Lösungsbeispiele für den Zugriff auf Excel, Access, PowerPoint, Visio und Outlook präsentiert.
- In Kapitel 11 schließlich geht es um das Steuern von eingebetteten Objekten innerhalb eines Word-Dokuments.

Das Zusammenspiel mit anderen Programmen funktioniert jedoch nur, wenn die Programmbibliotheken der entsprechenden Applikationen über definierte Schnittstellen »angesprochen« werden können. Hierfür müssen der Programmierumgebung (VB-Editor) die Schnittstellen bekannt gemacht werden, indem ein so genannter Verweis auf die jeweils benötigten Bibliotheken gesetzt wird.

Durch den Verweis stehen dann drei wichtige Funktionalitäten zur Verfügung: Erstens kann der Programmierer im VB-Editor IntelliSense nutzen, zweitens können die einzelnen Programmzeilen bereits während des Programmierens vom Compiler geprüft werden, und drittens ist ein kontextsensitiver Zugriff auf Hilfedateien möglich.

Vor- und Nachteile, die durch das Setzen eines Verweises auf eine Bibliothek entstehen, werden im ersten Abschnitt dieses Kapitels erläutert. Im zweiten Abschnitt werden Möglichkeiten aufgezeigt, wie eine andere Applikation auch ohne entsprechenden Verweis gesteuert werden kann.

## Verweise auf externe Bibliotheken

Durch den Verweis auf eine externe Programmbibliothek erhält der VB-Editor den Zugriff auf die darin enthaltenen Objekte. Eigenschaften und Methoden aller Objekte sowie die definierten Konstanten und Aufzählungen kann er direkt ansprechen und innerhalb des Programms verwenden. Zudem erfolgt beim Erfassen der Programmzeilen die Unterstützung mittels IntelliSense.

Ein Verweis wird für jedes VB-Projekt separat gesetzt. Dies bedeutet, dass für jede Dokumentvorlage oder für jedes einzelne Add-In die zusätzlichen Verweise gesetzt werden müssen. So kann die benötigte Funktionalität zusammengestellt werden.

Innerhalb der Word-Entwicklungsumgebung sind standardmäßig fünf Verweise aktiv und werden somit automatisch in jede Datei eingebunden:

- Visual Basic für Applikationen
- Microsoft Word 11.0 Object Library
- Die zugehörige Dokumentvorlage
- OLE-Automation
- Microsoft Office 11.0 Object Library

Die beiden letzten Einträge sind optional und können bei Nichtgebrauch deaktiviert werden.

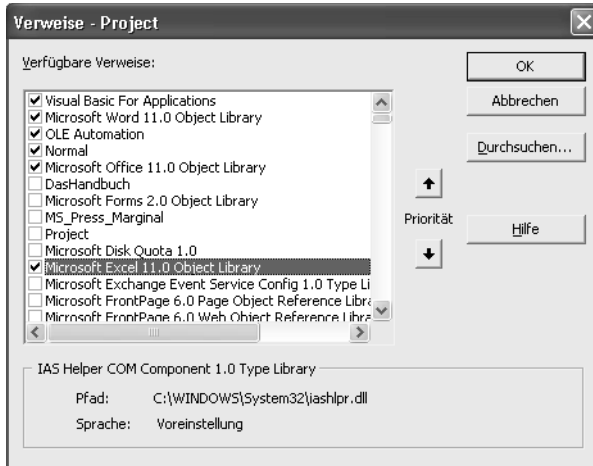


Verweis  
einfügen

Normalerweise werden die Verweise während der Entwicklung eines Programms manuell innerhalb des Editors gesetzt. Rufen Sie hierzu in der Visual Basic-Entwicklungsumgebung (VB-Editor) den Menübefehl *Extras/Verweise* auf und aktivieren Sie, wie in Abbildung 8.1 ersichtlich, im Listenfeld *Verfügbare Verweise* die benötigte Bibliothek.

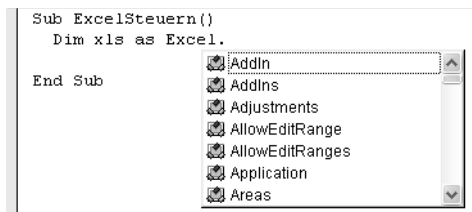
Wie Verweise auch dynamisch während der Laufzeit des Programms gesetzt werden können, ist in Kapitel 19 detailliert beschrieben.

**Abbildg. 8.1** Fügen Sie einen Verweis auf die *Microsoft Excel 11.0 Object Library* ein.



Nachdem der Verweis auf die Bibliothek von Excel gesetzt wurde, kann auf die Eigenschaften und Methoden von Microsoft Excel zugegriffen werden. Das Listing 8.1 zeigt ein einfaches Beispiel, wie das Programm gesteuert werden kann.

**Abbildg. 8.2** Unterstützung durch IntelliSense nach dem Einbinden der *Microsoft Excel 11.0 Object Library*



#### HINWEIS

Das Dialogfeld *Verweise* listet alle registrierten Programmbibliotheken auf. Hierzu gehören ActiveX-Dateien mit den Erweiterungen *.exe*, *.dll*, *.olb*, *.tlb* und *.ocx* sowie alle in Word geöffneten Dokumente, Dokumentvorlagen und geladenen Add-Ins.

**Listing 8.1** Erstellen einer Excel-Instanz und diese steuern

```

Sub ExcelSteuern()
'Das Beispiel verwendet einen Verweis auf die
'Microsoft Excel 11.0 Object Library
    Dim xls As Excel.Application
    Dim wkb As Excel.Workbook
    Dim wks As Excel.Worksheet

'Objekte erstellen
    Set xls = New Excel.Application
    Set wkb = xls.Workbooks.Add
    Set wks = wkb.Worksheets.Add

'Applikation bearbeiten
    With xls
        .WindowState = xlNormal
        .Visible = True
    End With

'Arbeitsblatt bearbeiten
    With wks
        .Range("A1").Formula = "Hallo Welt"
        .PrintOut
    End With

'Arbeitsmappe schließen
    With wkb
        .Saved = True
        .Close
    End With

'Applikation schließen
    xls.Quit

'Objekte freigeben (umgekehrte Reihenfolge)
    Set wks = Nothing
    Set wkb = Nothing
    Set xls = Nothing
End Sub

```

In Abbildung 8.1 erkennen Sie unter *Verfügbare Verweise* den Eintrag »Das Handbuch«. Es handelt sich dabei um ein geladenes Add-In (.dot-Datei), das Makros und Symbolleisten enthält. Auch auf dieses Add-In kann ein Verweis gesetzt werden, der es ermöglicht, die Prozeduren und Funktionen dieser Datei anzusprechen und im neuen Makro zu verwenden. Ein Beispiel für die Verwendung eines Verweises auf ein Add-In und die damit entstehenden Möglichkeiten ist in Kapitel 9 aufgeführt.

---

**WICHTIG** Damit die Makros mit den eingebundenen Bibliotheken auf jeder Arbeitsstation lauffähig sind, muss sichergestellt werden, dass die benötigten Dateien auf der Arbeitsstation installiert und registriert sind.

Werden die entsprechenden Dateien zusammen mit dem eigenen VBA-Projekt ausgeliefert, sind zusätzlich die Urheberrechte des Herstellers zu beachten.

---



Wird ein Verweis auf ein Add-In (.dot-Datei) gesetzt, sollte das Add-In im *StartUp*-Ordner von Word abgelegt werden. Nur so kann Word diesen Verweis dynamisch nachführen, falls die Verzeichnisstruktur auf der Entwicklungsstation mit jener auf dem Zielrechner nicht übereinstimmt.

Verweise auf korrekt installierte ActiveX-Dateien werden automatisch nachgeführt, da die benötigten Informationen in der Windows-Registrierung hinterlegt sind.

### Einbinden von zusätzlichen Steuerelementen

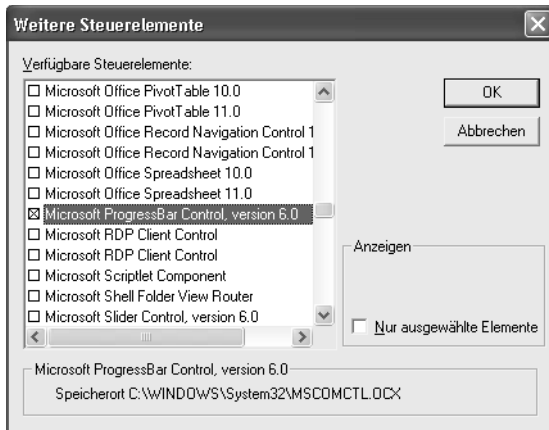
Grundsätzlich können neben den bestehenden Steuerelementen, die durch die *Microsoft Forms 2.0 Object Library* zur Verfügung gestellt werden, weitere Komponenten eingebunden werden.



Steuerelement  
einbinden

Dazu öffnen Sie innerhalb der VB-Entwicklungsumgebung ein Formular und wählen anschließend den Menübefehl *Extras/Zusätzliche Steuerelemente*. Im Listenfeld *Verfügbare Steuerelemente* aktivieren Sie nun, wie in Abbildung 8.3 ersichtlich, das Kontrollkästchen des gewünschten Steuerelements. Die Abbildung 8.4 zeigt das zusätzliche Steuerelement in der *Werkzeugsammlung*.

Abbildg. 8.3 Wählen Sie das benötigte Steuerelement aus und aktivieren Sie den Eintrag.



Der Verweis auf die zugehörige Bibliothek des Steuerelements wird automatisch gesetzt, sobald die Komponente zum ersten Mal in ein Formular bzw. Dialogfeld eingefügt wird.

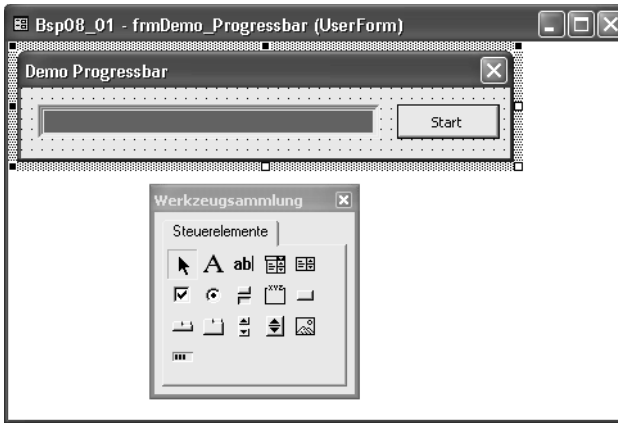
#### WICHTIG

Damit die Makros mit den eingebundenen Steuerelementen auf jeder Arbeitsstation lauffähig sind, muss sichergestellt werden, dass die benötigten Dateien auf der Arbeitsstation installiert und registriert sind.



Das zusätzliche Steuerelement ist in der Datei *Mscomctl.ocx* enthalten. Diese Datei befindet sich normalerweise bereits auf der Arbeitsstation und zwar im Verzeichnis *\Windows\System32*. Ansonsten befindet sich die Datei im Ordner *\Beispiele\Kap08* auf der beigelegten CD-ROM.

Abbildg. 8.4 Verwenden Sie das zusätzliche Steuerelement *Progressbar* wie alle anderen Steuerelemente.



### Zugriff auf unsichere ActiveX-Steuerelemente

Ab Office XP kann beim Öffnen von Dokumenten bzw. beim Laden von Add-Ins eine Meldung am Bildschirm erscheinen, die vor einer Aktivierung eines unsicheren ActiveX-Steuerelements warnt.

Diese Warnung erscheint, wenn im aktuellen Dokument ein ActiveX-Steuerelement eingebunden ist, dessen Quelle noch nicht als vertrauenswürdig eingestuft wurde.

Die detaillierten Hintergründe zu dieser Problematik und eventuelle Lösungen werden in verschiedenen Beiträgen von Microsoft aufgezeigt:

- »You are prompted to grant permission for ActiveX Controls when you open an Office XP or Office 2003 document« (<http://support.microsoft.com/default.aspx?scid=kb;en-us;827742>)
- »Problems occur when you use the Rich TextBox Control 6.0 in Office XP and in Office 2003« (<http://support.microsoft.com/default.aspx?scid=kb;en-us;838010>)
- »FIX Failed to Mark Safe for Scripting Using Visual Basic PDW« (<http://support.microsoft.com/default.aspx?scid=kb;en-us;221541>)



Das dargestellte Beispiel mit dem zusätzlichen Steuerelement finden Sie in der Beispieldatei *Bsp08\_01.doc* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap08*.

## Early versus Late Binding

Grundsätzlich gibt es zwei verschiedene Arten, eine andere Applikation aus einem Programm bzw. einem Makro heraus zu steuern. Für beide Arten wird in diesem Buch nur die englische Bezeichnung verwendet, da kein äquivalenter deutscher Ausdruck bekannt ist.

Early  
Binding

Beim *Early Binding* muss unbedingt ein Verweis auf die benötigte Bibliothek gesetzt werden. Die entsprechende Vorgehensweise wurde bereits im vorangegangenen Abschnitt »Verweise auf externe Bibliotheken« besprochen.

```
Dim xls As Excel.Application
Set xls = New Excel.Application
```

Late Binding

Beim *Late Binding* dagegen wird mittels `CreateObject` eine neue Instanz auf das benötigte Applikationsobjekt gesetzt. Ein Verweis auf die Bibliothek ist nicht nötig, die Objektvariable kann in diesem Fall nur als `Object` definiert werden.

```
Dim xls As Object
Set xls = CreateObject("Excel.Application")
```

Grundsätzlich besteht die Möglichkeit, die `CreateObject`-Funktion auch im Zusammenhang mit *Early Binding* einzusetzen. Das Schlüsselwort `New` ist jedoch vorzuziehen, sofern dieses von der Objektbibliothek bereits unterstützt wird. Zusätzliche interessante Informationen zur `CreateObject`-Funktion können in der Online-Hilfe nachgeschlagen werden.

#### HINWEIS

In beiden Fällen kann mittels `GetObject` eine Objektvariable auf eine bereits laufende Instanz gesetzt werden.

```
Set xls = GetObject(, "Excel.Application")
```

## Vorteile von Early Binding

Für den Einsatz von *Early Binding* – im Vergleich zu *Late Binding* – sprechen folgende Punkte, die das Programmieren bedeutend vereinfachen:

- Das Programm kann schneller ausgeführt werden, da der Programmcode bereits zur Entwicklungszeit kompiliert werden kann. Beim *Late Binding* hingegen kann das Programm erst zur Laufzeit mit der Applikation verknüpft und kompiliert werden.
- Das Auffinden von fehlerhaften Programmzeilen während der Entwicklungszeit gestaltet sich viel einfacher. Durch Aufruf des Menübefehls *Debuggen/Kompilieren* werden die Syntaxfehler sofort markiert.
- Zum Erfassen von Code stehen *IntelliSense* und eine kontextsensitive Hilfe zur Verfügung. Die Bibliothek kann im Objektkatalog durchforstet werden.
- Nebst dem Zugriff auf alle Eigenschaften und Methoden stehen auch alle definierten Konstanten und Aufzählungen zur Verfügung.

```
xls.WindowState = xlNormal
```

Ohne den Verweis auf die Bibliothek (in diesem Beispiel auf die Excel-Bibliothek) müsste der effektive Wert an die Objekteigenschaft zugewiesen werden. Dieser müsste zuerst ermittelt oder im Objektkatalog aufgespürt werden.

```
xls.WindowState = -4143
```

Das folgende Beispiel (Listing 8.2) zeigt, wie ein Makro mittels Early Binding auf die Kontakte von Microsoft Outlook zugreift und die gefundenen Einträge auflistet.

**Listing 8.2** Auflisten aller Kontakte aus Outlook unter Verwendung von Early Binding

```
Sub Earlybinding_Outlook()
'Das Beispiel verwendet einen Verweis auf
'Microsoft Outlook 11.0 Object Library
Dim doc As Word.Document
Dim olAnw As Outlook.Application
Dim olOrdner As Outlook.MAPIFolder
Dim olKontakt As Object '**
Dim olItem As Items

'** = Der Kontakt kann nicht als "Outlook.ContactItem" definiert werden,
da auch Verteilerlisten, Ordner usw. im Kontakte-Ordner vorhanden sein
können. Dies muss speziell behandelt werden.

'Neues Dokument erzeugen
Set doc = Application.Documents.Add

'Outlook-Instanz setzen
Set olAnw = New Outlook.Application

'Kontakte-Ordner setzen
Set olOrdner = olAnw.Session.GetDefaultFolder(olFolderContacts)

'Einträge setzen
Set olItem = olOrdner.Items

'Alle Kontakte auflisten
For Each olKontakt In olItem
If olKontakt.Class = olContact Then
doc.Range.InsertAfter olKontakt.FileAs & vbVerticalTab
Else
'Bei allen anderen nichts machen
End If
Next olKontakt

olAnw.Quit
Set olItem = Nothing
Set olKontakt = Nothing
Set olOrdner = Nothing
Set olAnw = Nothing
End Sub
```



Das dargestellte Beispiel finden Sie in der Beispieldatei *Bsp08\_02.doc* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap08*.

## Vorteile von Late Binding

Für den Einsatz von Late Binding stehen die folgenden Vorteile im Vordergrund:

- Der Programmcode von Late Binding ist unabhängiger von der installierten Programmversion. In den meisten Fällen wird zwar ein gesetzter Verweis automatisch an die aktuelle Programmversion angepasst, doch leider nicht in jedem Fall.  
Wird ein VBA-Projekt beispielsweise mit Word 2002 entwickelt und zusätzlich ein Verweis auf die Microsoft Excel 10.0 Object Library gesetzt, so kann dieser Verweis dynamisch auf die Version 11.0 angepasst werden, wenn das Makro auf einer Arbeitsstation mit installiertem Word 2003 ausgeführt wird.  
Da diese dynamische Anpassung nicht in allen Fällen bzw. mit allen Bibliotheken einwandfrei funktioniert, wird gerne auf Late Binding zurückgegriffen, wenn ein Programm unabhängig von der installierten Version von Microsoft Office entwickelt wird.
- Da weniger Verweise innerhalb eines Projektes gesetzt werden, benötigt der Compiler weniger Zeit für seine Arbeit.

### WICHTIG

Wird ein Makro auf verschiedenen Versionen von Microsoft Office eingesetzt, so sollte die Entwicklung des Projekts grundsätzlich mit der niedrigsten aller benötigten Office-Versionen umgesetzt werden.

Bei diesem Vorgehen ist sichergestellt, dass keine Objekte und Funktionen in das Projekt einfließen, die nicht in allen Versionen zur Verfügung stehen.

Das Makro aus Listing 8.3 greift – wie Listing 8.2 – ebenfalls auf die Outlook-Kontakte zu und erstellt eine Liste der gefundenen Einträge. Diesmal jedoch wird Late Binding für den Zugriff auf Outlook verwendet.

Listing 8.3

Auflisten aller Kontakte aus Outlook unter Verwendung von Late Binding

```
Sub Latebinding_Outlook()
    Dim doc As Word.Document
    Dim olAnw As Object
    Dim olOrdner As Object
    Dim olKontakt As Object
    Dim olItem As Object

    'Neues Dokument erzeugen
    Set doc = Application.Documents.Add

    'Outlook-Instanz setzen
    Set olAnw = CreateObject("Outlook.Application")

    'Kontakte-Ordner setzen
    Set olOrdner = olAnw.Session.GetDefaultFolder(10)

    'Einträge setzen
    Set olItem = olOrdner.Items

    'Alle Kontakte auflisten
    For Each olKontakt In olItem
        If olKontakt.Class = 40 Then
```

**Listing 8.3** Auflisten aller Kontakte aus Outlook unter Verwendung von Late Binding (Fortsetzung)

```

        doc.Range.InsertAfter olKontakt.FileAs & vbVerticalTab
    Else
        'Bei allen anderen nichts machen
    End If
Next olKontakt

olAnw.Quit
Set olItem = Nothing
Set olKontakt = Nothing
Set olOrdner = Nothing
Set olAnw = Nothing
End Sub

```

Durch das Weglassen eines Verweises besteht auch kein Zugriff auf definierte Konstanten. Aus diesem Grund müssen die effektiven Werte ermittelt und direkt in das Programm eingetragen werden. So steht beispielsweise der Wert 10 anstelle der Konstante `olFolderContacts`.

```
Set olOrdner = olAnw.Session.GetDefaultFolder(10)
```

**PROFITIPP**

Im Abschnitt »Ganz clever, wer beide Arten kombiniert« zeigen wir auf, wie die Vorteile von beiden Varianten – Early und Late Binding – im gleichen Projekt genutzt werden können. Wir Autoren möchten Ihnen dieses Vorgehen nahe legen, da Sie Ihr Programm schneller entwickeln und dennoch unabhängig von der Programmversion der zu steuernden Applikation sind.

## Ganz clever, wer beide Arten kombiniert

Die Entscheidung, welche Art Sie in Ihrem Projekt einsetzen, können wir Ihnen nicht abnehmen. Für den Einsteiger wird wohl eher Early Binding in Frage kommen, weil damit die direkte Unterstützung durch den VB-Editor verbunden ist.

Im professionellen Umfeld ist eher Late Binding empfehlenswert, da eine Unabhängigkeit der eingesetzten Programmversionen im Vordergrund steht.

Doch was spricht dagegen, die Vorteile beider Varianten zu nutzen? Auch das Beispiel-Makro aus Listing 8.4 greift auf die Outlook-Kontakte zu. Innerhalb der gleichen Prozedur kommen nun aber beide Arten der Programmierung zum Einsatz, wobei zu beachten ist, dass entweder Early oder Late Binding aktiv ist. Dies kann mit einer Compiler-Konstante gesteuert werden.

```
#Const EARLYBINDING = False 'oder True
```

Wird der Wert der Compiler-Konstante geändert, muss zusätzlich der Verweis auf die *Microsoft Outlook 11.0 Object Library* manuell hinzugefügt oder eben wieder entfernt werden.

Listing 8.4 Die Vorteile von Early und Late Binding, kombiniert im selben Programm

```

Sub CleverKombiniert Outlook()
    #Const EARLYBINDING = False 'oder True

    #If EARLYBINDING Then
        'Wird mit Early Binding gearbeitet, muss der Verweis auf die
        'Microsoft Outlook 11.0 Object Library aktiviert werden.
        Dim olAnw As Outlook.Application
        Dim olOrdner As Outlook.MAPIFolder
        Dim olKontakt As Object
        Dim olItem As Items
    'Outlook-Instanz setzen
        Set olAnw = New Outlook.Application
    #Else
        Const olFolderContacts As Integer = 10
        Const olContact As Integer = 40
        Dim olAnw As Object
        Dim olOrdner As Object
        Dim olKontakt As Object
        Dim olItem As Object
    'Outlook-Instanz setzen
        Set olAnw = CreateObject("Outlook.Application")
    #End If

    Dim doc As Word.Document

    'Neues Dokument erzeugen
    Set doc = Application.Documents.Add

    'Kontakte-Ordner setzen
    Set olOrdner = olAnw.Session.GetDefaultFolder(olFolderContacts)

    'Einträge setzen
    Set olItem = olOrdner.Items

    'Alle Kontakte auflisten
    For Each olKontakt In olItem
        If olKontakt.Class = olContact Then
            doc.Range.InsertAfter olKontakt.FileAs & vbVerticalTab
        Else
            'Bei allen anderen nichts machen
        End If
    Next olKontakt

    olAnw.Quit
    Set olItem = Nothing
    Set olKontakt = Nothing
    Set olOrdner = Nothing
    Set olAnw = Nothing
End Sub

```

Um die Vorteile der beiden Arten innerhalb Ihres Projekts zu nutzen, gehen Sie folgendermaßen vor:

1. Setzen Sie einen Verweis auf die benötigte Bibliothek. So erhalten Sie Zugriff auf die Objekte und Unterstützung durch IntelliSense.

2. Deklarieren Sie einen ersten Block mit Programmzeilen, der die Compiler-Anweisungen enthält:

```
#Const EARLYBINDING = False 'oder True
#If EARLYBINDING Then
#Else
#End If
```

3. Erfassen Sie alle Codezeilen für Ihr Projekt. Die Deklarationszeilen aller Objektvariablen und das Erstellen der Instanz auf die eingebundene Applikation wird innerhalb der #If...Then-Anweisung gesetzt:

```
#If EARLYBINDING Then
    Dim olAnw As Outlook.Application
    Set olAnw = New Outlook.Application
```

4. Kopieren Sie die Zeilen vom ersten Block (True) der #If...Then-Anweisung in den zweiten Block (False). Definieren Sie alle Objektvariablen auf den Datentyp *Object* um und passen Sie die Zeile zum Erstellen der Instanz auf die eingebundene Applikation an:

```
#Else
    Dim olAnw As Object
    Set olAnw = CreateObject("Outlook.Application")
```

5. Stellen Sie sicher, dass das Programm ohne Fehler ausgeführt werden kann.
6. Entfernen Sie den Verweis auf die eingebundene Bibliothek und ändern Sie den Wert der Compiler-Konstanten auf False. Rufen Sie anschließend den Menübefehl *Debuggen/Kompilieren* auf. Der Versuch, die Applikation erneut zu kompilieren, wird fehlschlagen, da alle verwendeten Konstanten und Aufzählungen aus der nicht mehr eingebundenen Bibliothek unbekannt sind. Definieren Sie manuell jede vermisste Konstante im zweiten Block (False) der #If...Then-Anweisung. Weisen Sie der Konstanten in einem ersten Schritt einen Dummy-Wert zu, denn dies reicht aus, damit der Compiler den nächsten Fehler im Programm anzeigt:

```
#Else
    Const olFolderContacts As Integer = 0
    Const olContact As Integer = 0
    Dim olAnw As Object
```

7. Alle fehlenden Konstanten und Aufzählungswerte werden ermittelt, sobald das Programm fehlerfrei kompiliert werden kann. Anschließend wird der Verweis erneut gesetzt. Im Objektkatalog werden die effektiven Werte für alle Konstanten und Aufzählungswerte ermittelt und in die Deklarationszeile übertragen:

```
#Else
    Const olFolderContacts As Integer = 10
    Const olContact As Integer = 40
    Dim olAnw As Object
```



#...

### Compiler-Anweisungen

Steht am Anfang einer Programmzeile eine Raute (#), so bedeutet dies, dass diese Anweisung für den Compiler bestimmt ist. Diese Zeile hat somit keinen Einfluss auf das Verhalten des Programms, sondern auf die Interpretation des Compilers.

Mittels einer Compiler-Anweisung können Konstanten definiert werden, die innerhalb des Programms eine `#If...Then`-Anweisung steuern.

Auf diese Art ist es möglich, den gleichen Programmcode für zwei oder mehrere unterschiedliche Zwecke zu verwenden. Es muss lediglich vor dem Kompilieren der Applikation der Wert der steuernden Konstanten gesetzt werden.

Compiler-Konstanten stehen nur einer Compiler-Anweisung zur Verfügung und können nicht innerhalb einer »normalen« Programmzeile verwendet werden. Das Gleiche gilt umgekehrt natürlich auch. Eine »normale« Konstante kann innerhalb einer Compiler-Anweisung nicht ausgewertet werden.

Jetzt stehen innerhalb des gleichen Programms beide Möglichkeiten, also Early und Late Binding, zur Verfügung. Welche der beiden Arten aktiv und kompiliert wird, kann durch Zuweisen des gewünschten Werts an die entsprechende Compiler-Konstante definiert werden.

Wird das Makro zu einem späteren Zeitpunkt weiterentwickelt, wird wiederum Early Binding aktiviert. Wurde die Änderung umgesetzt und das Programm getestet, wird für den produktiven Einsatz auf Late Binding umgeschaltet.



Das dargestellte Beispiel finden Sie in der Beispieldatei *Bsp08\_03.doc* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap08*.

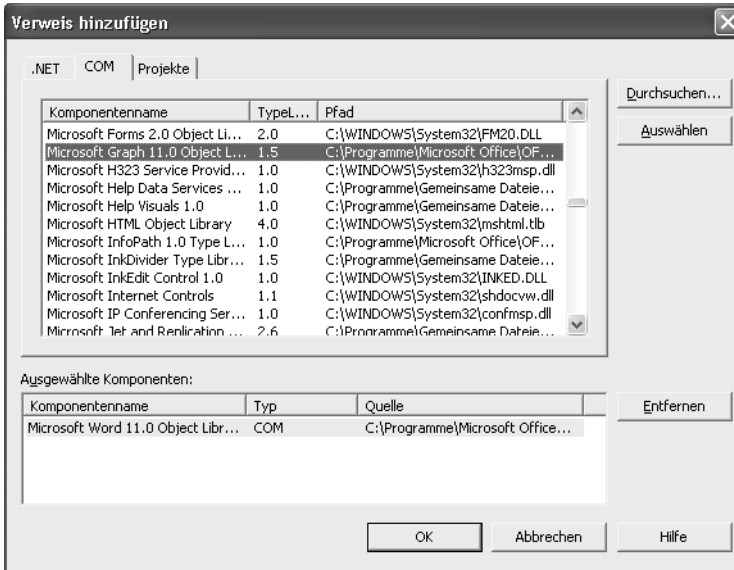
## Über das .NET Framework

Microsoft hat das .NET Framework als Alternative zur COM-Umgebung entwickelt, die seit mehr als einem Jahrzehnt das Kernstück des Betriebssystems Windows und aller darauf laufenden Anwendungen bildet. Neben den Sicherheitsfragen, die immer brisanter werden, hat COM gewisse innewohnende Nachteile, die erst im Laufe der Zeit deutlich wurden. Von der neuen Umgebung verspricht man sich hinsichtlich dieser Nachteile eine Verbesserung.

Wichtig ist die Erkenntnis, dass eine .NET-Anwendung eine COM-Applikation über eine besondere Schnittstelle ansprechen muss, die zwischen den zwei Umgebungen vermittelt und übersetzt: die so genannten »Interop Assemblies« (IAs). Wenn Sie in Visual Studio im Dialogfeld *Verweis hinzufügen* (siehe Abbildung 8.5) vermeintlich einen Verweis auf eine COM-Bibliothek festlegen, analysiert Visual Studio die Objekt-Bibliothek und erstellt die nötige Schnittstelle – IA –, um damit kommunizieren zu können.

**HINWEIS** Wir bieten in diesem Buch nur eine Übersicht. Für eine vertiefte Einsicht in die Automatisierung von Office-Anwendungen aus der .NET-Umgebung empfehlen wir das Buch »Microsoft .NET Development for Microsoft Office« von Andrew Whitechapel, erschienen bei Microsoft Press.

Abbildg. 8.5 Das Dialogfeld *Verweis hinzufügen* (Registerkarte COM) wird über das *Projekt*-Menü eingeblendet.



Für Office 2003 sowie Office XP stellt Microsoft »Primary Interop Assemblies« (PIAs) zur Verfügung, die für diese Aufgabe optimiert sind. Die Office 2003-PIAs gehören zum Lieferumfang, können aber erst installiert werden, wenn das .NET Framework auf dem Rechner vorhanden ist. Die PIAs für Office XP stehen zum Herunterladen auf Microsoft.com bereit: <http://support.microsoft.com/default.aspx?scid=kb;en-us;328912>. Liegen die PIAs auf dem System vor, werden sie automatisch geladen, wenn dem Projekt ein Verweis hinzugefügt wird.

**HINWEIS** Wenn Ihnen nach dem Hinzufügen eines Verweises zu Word 2002 oder Word 2003 auffällt, dass Visual Studio doch ein Interop Assembly im Projektordner erstellt hat, sind die PIAs auf dem System nicht korrekt installiert.

Microsoft empfiehlt, für jede Word-Version ein eigenständiges Projekt zu entwickeln, das auf die entsprechende Version verweist. Wie bei der Automatisierung in der COM-Umgebung ist es jedoch durchaus möglich, auf eine frühere Version zu verweisen und das Projekt mit einer späteren zu verwenden. Dabei müssen Sie sich bewusst sein, dass

- Sie das Projekt mit der ältesten Word-Version entwickeln
- Office 2000 die älteste Version ist, worauf verwiesen werden kann
- Microsoft für diese Version keine PIAs zur Verfügung gestellt hat. Sie müssen Visual Studio einen Satz IAs erstellen lassen

- Sie diese IAs mit der .NET-Anwendung verteilen und installieren müssen

Late Binding kann hier Abhilfe schaffen. Wie schon in Kapitel 6 in den Diskussionen zu WordBasic sowie Suchen und Ersetzen demonstriert, ist es möglich, wenngleich etwas umständlich, Office-Anwendungen mit Late Binding zu automatisieren. Diese Funktionalität gehört der Klasse `System.Reflection.Marshalling` an. Weitere Informationen zu C# und Late Binding finden Sie im Knowledge Base-Artikel »Binding for Office automation servers with Visual C# .NET« (<http://support.microsoft.com/default.aspx?scid=KB;EN-US;302902>).

Die Steuerung der Word-Anwendung ist allgemein langsamer von der .NET-Umgebung aus, da der Programmablauf über zwei Schnittstellen ausgeführt wird: .NET auf COM, und dann über die COM-Bibliothek. Auch in dieser Hinsicht offeriert Late Binding eine Alternative. Mehr zum Thema Late Binding und Automatisierung von Office-Anwendungen finden Sie im Artikel »INFO: Use DISPID Binding to Automate Office Applications Whenever Possible« (<http://support.microsoft.com/default.aspx?scid=kb;en-us;247579>).

## Zusammenfassung

In diesem Kapitel wurden die Grundlagen der Fernsteuerung vermittelt. Dabei ging es um in erster Line allgemeingültiges Wissen.

- Als Erstes wurde aufgezeigt, wie ein Verweis auf eine Programmbibliothek dem aktuellen Projekt hinzugefügt wird, damit die zugehörige Applikation ferngesteuert werden kann (Seite 466).
- Im Weiteren wurde gezeigt, wie zusätzliche Steuerelemente eingebunden und auf einem Userform-Objekt verwenden werden können (Seite 469).
- Es wurden die Vor- und Nachteile von Early Binding gegenüber von Late Binding erläutert (Seite 470). Und es wurde aufgezeigt, wie man sich die Vorteile beider Varianten zu Nutze machen kann (Seite 474).
- Abschließend wurde das Zusammenspiel für das Fernsteuern innerhalb des .NET Frameworks kurz aufgezeigt (Seite 477).



## Kapitel 9

# Word von anderen Umgebungen aus steuern

### In diesem Kapitel:

Fernsteuern von Microsoft Word	482
Programmzeilen zentral verwalten und gemeinsam nutzen	490
Fernsteuerung aus Office-fremden Umgebungen	495
Zusammenfassung	508

Dieses Kapitel beschäftigt sich mit dem Fernsteuern von Word. Es wird aufgezeigt, wie ein Zugriff von außen auf dieses Programm möglich ist. Bei diesem Zugriff stehen verschiedene Möglichkeiten zur Steuerung zur Verfügung.

Eine Möglichkeit besteht darin, dass vorhandene Makros innerhalb von Word gestartet werden. Eine zweite Möglichkeit ist der direkte Zugriff auf das Objektmodell von Word. Dieses steht dem Programmierer uneingeschränkt zur Verfügung, sobald eine entsprechende Objektvariable angelegt und einer Instanz von Word zugewiesen wird.

Einen weiteren Schwerpunkt soll die Verwendung einer Programmbibliothek darstellen. Hier wird aufgezeigt, wie allgemeine Prozeduren, Funktionen, Variablen, Konstanten und Dialogfelder in ein zentrales Add-In ausgelagert und von verschiedenen Dokumentvorlagen gemeinsam genutzt werden können.

Zudem wird die Fernsteuerung aus einer Office-fremden Umgebung heraus kurz angesprochen sowie ein Beispiel zu Visual Studio Tools for Office (VSTO) vorgestellt.

#### **HINWEIS**

Neben den auf einer Dokumentvorlage (\*.dot) basierenden Add-Ins können alternativ so genannte COM-Add-Ins zum Einsatz gelangen. Wie solche Add-Ins erzeugt und genutzt werden, übersteigt den Rahmen dieses Buches. Informationen dazu finden Sie auf der MSDN-Webseite, beispielsweise im Artikel »Building a COM Add-In for Microsoft Office XP Using Microsoft Visual Basic 6.0« ([http://msdn.microsoft.com/library/en-us/dnoxp/ht/odc\\_comaddinb6.asp](http://msdn.microsoft.com/library/en-us/dnoxp/ht/odc_comaddinb6.asp)).

## **Fernsteuern von Microsoft Word**

Wird Word von außen angesprochen, können zu seiner Steuerung die vorhandenen Makros gestartet werden. Wird dabei direkt auf das Objektmodell zugegriffen, kann Word uneingeschränkt ferngesteuert werden.

Es besteht sogar die Möglichkeit, dass Word selbst eine andere Instanz von Word fernsteuert. Ein entsprechendes Beispiel ist im Abschnitt »Versteckte Word-Instanz steuern« in diesem Kapitel detailliert beschrieben.

## **Makros von außen anstoßen**



Die einfachste Möglichkeit, Word fernzusteuern, ist das Aufrufen eines vorhandenen Makros. Das Objektmodell von Word stellt dazu die Run-Methode zur Verfügung. Bevor jedoch das Beispiel aus Listing 9.1 genutzt werden kann, muss das theoretische Verständnis vorhanden sein.

Das Application-Objekt von Word bietet mit der Run-Methode die Möglichkeit, ein bestehendes Makro zu starten. Der Aufruf dieses Makros wird synchron abgearbeitet. Dies bedeutet, dass der aufrufende Prozess stehen bleibt, bis das gestartete Makro abgearbeitet ist und die Kontrolle wieder an diesen zurückgegeben wird.

Grundsätzlich können durch die Run-Methode alle Makros aufgerufen werden, die im Menübefehl *Extras/Makro/Makros* im Listenfeld *Makroname* enthalten sind. Welche Bedingungen ein Makro erfüllen muss, um im genannten Listenfeld aufgeführt zu werden, ist in Kapitel 1 zusammengefasst.

Grundsätzlich kann der Aufruf eines Makros mit der Run-Methode direkt über dessen Namen erfolgen, sofern dieser eindeutig ist:

```
Application.Run "Makroname"
```

Da die Bezeichnung eines Makros aber nicht in jedem Fall eindeutig ist, sollte der Aufruf stets über den vollen Kontext des Makros erfolgen. So ist sichergestellt, dass nicht fälschlicherweise ein Makro mit der gleichen Bezeichnung ausgeführt wird, das sich auf ein anderes Projekt bezieht. Die Rangfolge, die Word verwendet, wenn Makros mit gleichen Bezeichnungen vorhanden sind, wurde in Kapitel 1 vorgestellt.

```
Application.Run "Projektname.Modulname.Makroname"  
Application.Run "'Documentname'!Modulname.Makroname"
```

Der Aufruf der Run-Methode kann mit zusätzlichen Parametern versehen werden, um Daten an das Makro zu übergeben. Maximal können 30 Parameter deklariert werden.

Diese Makros müssen unbedingt als öffentliche Prozedur (Public Sub) deklariert und dem Entwickler bekannt sein, denn diese Prozeduren werden in der Liste der vorhandenen Makros nicht angezeigt:

```
Application.Run "Makroname", "Var1", "Var2", ..., "Var30"
```

Das Übermitteln von Werten mittels Parametern ist zwar sehr einfach und in den meisten Fällen auch ausreichend. Doch die beschränkte Anzahl von maximal 30 Argumenten kann teilweise zu einem Engpass führen. Weiterhin muss beachtet werden, dass der Aufruf der Run-Methode keine Werte an das aufrufende Programm zurückgeben kann.

**WICHTIG** Wird beim Aufruf der Run-Methode von der Möglichkeit Gebrauch gemacht, zusätzliche Argumente zu übermitteln, stimmt die Syntax nicht mit den Angaben in der VBA-Hilfe überein.

Der Aufruf der Methode kann nur über den einfachen Namen erfolgen. Ansonsten wird zur Laufzeit ein Programmfehler auftreten. Das Verwenden des einfachen Namens kann wiederum zu Problemen führen, wenn mehrere Makros mit der gleichen Bezeichnung in Einsatz sind:

```
Application.Run "Makroname", "Var1"      'Aufruf funktioniert fehlerfrei  
Application.Run "Projektname.Modulname.Makroname", "Var1"  'Erzeugt Laufzeitfehler
```

In Listing 9.1 wird ein bestehendes Word-Makro aus einem Excel-Makro heraus aufgerufen. Bei dieser Konstellation kann nicht von einer Fernsteuerung im eigentlichen Sinne gesprochen werden, denn das Word-Objekt wird nicht gesteuert, sondern es wird die Kontrolle an ein anderes Makro übergeben. Damit die beiden Makros miteinander Daten austauschen können, wurde in diesem Beispiel bewusst der Umweg über eine INI-Datei gewählt. Auf diese Art wird die Problematik mit dem absoluten Namen umgangen. Gleichzeitig können Werte an die aufrufende Prozedur auf dem gleichen Kommunikationsweg übertragen werden.

Alle in Word benötigten Werte werden vor dem Aufruf der Run-Methode durch das Excel-Makro in eine INI-Datei zwischengespeichert. Das Wordmakro kann die entsprechende Datei einlesen und die Werte weiter verarbeiten. Eventuelle Rückgabewerte an das aufrufende Excel-Makro würden auf dem gleichen Weg übermittelt. Weitere Informationen zum Thema »INI-Dateien« sind in Kapitel 12 zusammengefasst.

**HINWEIS** Damit das Beispielmakro aus Listing 9.1 fehlerfrei abgearbeitet werden kann, muss unbedingt die Datei *Bsp09\_03.dot* in den *Startup*-Ordner von Word kopiert werden. Welches Verzeichnis von Word als *Startup*-Ordner verwendet wird und wie dieses Verzeichnis festgelegt werden kann, wurde in Kapitel 1 detailliert beschrieben.

Das Abspeichern der Datei *Bsp09\_03.dot* im *Startup*-Ordner von Word ändert den Status dieser Datei. Aus Sicht von Word handelt es sich dabei nicht mehr um eine Dokumentvorlage, jetzt steht ein so genanntes Add-In zur Verfügung. Mehr zum Thema »Add-In« kann in Kapitel 13 nachgeschlagen werden.

**Listing 9.1** Ein Excel-Makro steuert Word durch Verwendung der *Run*-Methode.

```
Sub Demo_WordFernsteuernRun()
'Damit dieses Beispiel funktioniert, muss die Datei Bsp09_03.dot in den
'StartUp-Ordner von Word kopiert werden.
    Dim strText As String
    Dim docApp As Object
    Dim bWordWurdeGestartet As Boolean

    strText = InputBox("Bitte den Namen eingeben.")

    If Not Len(Trim$(strText)) = 0 Then
        Open Environ$("Temp") & "\MakroTest.ini" For Output As #1
        Print #1, "[Anwender]"
        Print #1, "Name=" & strText
        Close #1

        On Error Resume Next
        Set docApp = GetObject(, "Word.Application")
        On Error GoTo 0

        If docApp Is Nothing Then
            Set docApp = CreateObject("Word.Application")
            bWordWurdeGestartet = True
        End If
        docApp.Visible = True
        docApp.Activate

        'Makro starten
        docApp.Run "Demo_ApplicationRun_1"

        If bWordWurdeGestartet Then
            docApp.Quit
        End If

        Set docApp = Nothing
    End If
End Sub
```

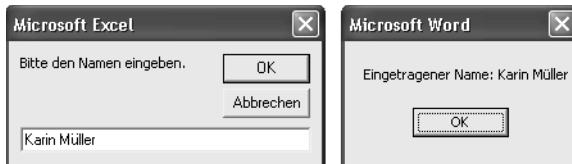




Die Prozedur *Demo\_WordFernsteuernRun* befindet sich in der Datei *Bsp09\_02.xls* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap09*. Diese Datei wurde mit Excel erstellt, das Makro muss aus diesem Programm heraus gestartet werden. Um das Makro zu starten, öffnen Sie in Excel die Datei *Bsp09\_02.xls* und wählen Sie den Menübefehl *Extras/Makro/Makros*. In der Liste der verfügbaren Makros wird die Prozedur *Demo\_WordFernsteuernRun* ausgewählt und die Schaltfläche *Ausführen* betätigt.

Die Funktionsweise des Makros ist schnell erklärt. Der Anwender soll seinen Namen eintragen. Dazu wird die Funktion *InputBox* verwendet. Der eingetragene Wert wird in einer INI-Datei gespeichert. Im nächsten Schritt wird versucht, eine laufende Instanz von Word zu ermitteln. Schlägt dieser Versuch fehl, wird eine neue Instanz angelegt. Anschließend erfolgt der eigentliche Aufruf des Makros mit der *Run*-Methode. Die Interaktion des Makros mit dem Anwender ist in Abbildung 9.1 dargestellt.

Abbildg. 9.1 Eingabe des Werts in Excel, Ausgabe desselben in Word



Aus dem in Listing 9.1 aufgezeigten Excel-Makro erfolgt ein Aufruf eines Word-Makros anhand der *Run*-Methode. Die entsprechende Programmsequenz ist in Listing 9.2 aufgeführt.

Listing 9.2 Das aus Excel heraus aufgerufenes Word-Makro *Demo\_ApplicationRun\_1*

```
Public Sub Demo_ApplicationRun_1()
    Dim strText As String

    strText = System.PrivateProfileString(
        FileName:=Environ$("Temp") & "\MakroTest.ini", _
        Section:="Anwender", _
        Key:="Name")

    MsgBox "Eingetragener Name: " & strText
End Sub
```

Das Makro liest die von Excel übermittelten Werte aus der INI-Datei ein und zeigt diese anhand der Funktion *MsgBox* am Bildschirm an (Abbildung 9.1).

### Makros dynamisch nachladen

Damit ein Makro anhand der *Run*-Methode ausgeführt werden kann, muss diese als öffentliche Prozedur deklariert werden (*Public Sub*). Die Prozedur selbst kann in einem beliebigen Dokument, einer Dokumentvorlage oder einem Add-In (*.dot*) gespeichert sein.

Als Voraussetzung, dass das Listing 9.1 fehlerfrei ausgeführt werden kann, musste die Datei *Bsp09\_03.dot* zuerst als Add-In zur Verfügung gestellt werden. Diese Voraussetzung ist jedoch nicht zwingend nötig, denn ein so genanntes Add-In kann zur Laufzeit aus irgendeinem beliebigen Ord-

ner nachgeladen und zu einem späteren Zeitpunkt wieder entladen werden. Die Möglichkeiten des Addin-Objekts wurden bereits in Kapitel 6 näher vorgestellt.

In Listing 9.3 ist ein Auszug der Prozedur *Demo\_WordFernsteuernAddinLaden* aufgeführt. Die betreffenden Zeilen steuern das Laden und Entladen des zusätzlichen Add-Ins. Da der Aufruf der Run-Methode synchron abgearbeitet wird, besteht keine Gefahr, dass das Add-In bereits wieder entladen wird, bevor das Makro abgearbeitet wurde (die eigentliche Prozedur befindet sich in der Datei *Bsp09\_02.xls*).

**Listing 9.3** Zusätzliche Add-Ins können bei Bedarf dynamisch geladen und entladen werden.

```
Const strADDIN As String = "C:\WordBuch\Beispiele\Kap09\Bsp09_04.dot"
'Addin laden
docApp.AddIns.Add Filename:=strADDIN, Install:=True
docApp.Run "Demo_ApplicationRun_2"
docApp.AddIns(strADDIN).Delete
```

Beim Laden des Add-Ins steht der Parameter *Install* zur Verfügung. Wird der Wert auf *True* gesetzt, so wird das Add-In geladen und gleichzeitig aktiviert:

```
docApp.AddIns.Add Filename:=strADDIN, Install:=True
```

Das Entladen eines einzelnen Add-Ins erfolgt mit der *Delete*-Methode. Diese Methode deaktiviert und entlädt das Add-In. Das Add-In wird jedoch nicht auf dem Datenträger gelöscht, sondern lediglich aus der Liste der verfügbaren Add-Ins entfernt:

```
docApp.AddIns(strADDIN).Delete
```

## Word effektiv fernsteuern

Bei den beiden vorhin aufgeführten Beispielen kann eigentlich noch nicht von Fernsteuerung gesprochen werden. Denn außer einem Verbindungsaufbau zu Word und dem synchronen Aufruf eines Makros wurde nichts gesteuert.

In Listing 9.4 findet eine echte, wenn auch nur einfache Fernsteuerung von Word statt. In diesem Beispiel liegen bereits persönliche Adressen in einer Excel-Tabelle vor. Der Aufruf des Makros startet eine Abfrage an den Benutzer, damit der gewünschte Datensatz angegeben werden kann. Die entsprechende Adresse wird in ein neues Dokument übertragen, das auf der Dokumentvorlage *Bsp09\_Brief.dot* basiert.

**Listing 9.4** Aus Excel heraus Word fernsteuern und die Empfängeradresse in den Brief eintragen

```
Sub Demo_AdresseInDokumentÜbertragen()
    Const strDOT_BRIEF As String = "C:\WordBuch\Beispiele\Kap09\Bsp09_Brief.dot"

    #Const EARLYBINDING = False 'oder True

    'Variablen und Objekte anlegen
    #If EARLYBINDING Then
        'Wird mit Early Binding gearbeitet, muss ein Verweis auf
        'die "Microsoft Word 11.0 Object Library" aktiviert werden.
```

Listing 9.4 Aus Excel heraus Word fernsteuern und die Empfängeradresse in den Brief eintragen (Fortsetzung)

```

    Dim docApp As Word.Application
    Dim docDoc As Word.Document
#Else
    Dim docApp As Object
    Dim docDoc As Object
#End If

Dim strZeile As String
Dim strName As String
Dim strStrasse As String
Dim strOrt As String
Dim strAdresse As String
Dim intAntwort As Integer

'Empfängeradresse bestimmen
Do
    strZeile = InputBox("Geben Sie die Zeilennummer für die Empfängeradresse ein.", _
        "Adressnummer wählen", 2)

    If strZeile = "" Then
        Exit Sub
    End If

    strName = ActiveWorkbook.ActiveSheet.Range("A" & strZeile).Text
    strStrasse = ActiveWorkbook.ActiveSheet.Range("B" & strZeile).Text
    strOrt = ActiveWorkbook.ActiveSheet.Range("C" & strZeile).Text
    strAdresse = strName & vbCr & strStrasse & vbCr & strOrt

    intAntwort = MsgBox("Wurde die richtige Adresse gewählt?" & vbCr & _
        strAdresse, vbYesNoCancel + vbQuestion)
Loop While intAntwort = vbNo

If intAntwort = vbYes Then
    #If EARLYBINDING Then
        Set docApp = New Word.Application
    #Else
        Set docApp = CreateObject("Word.Application")
    #End If

'Brief erzeugen und Empfängeradresse übertragen.
If Not (docApp Is Nothing) Then
    With docApp
        .Visible = True

        Set docDoc = .Documents.Add(Template:=strDOT_BRIEF)
        With docDoc
            strAdresse = Replace(strAdresse, vbCr, Chr$(11))
            .Bookmarks("EmpfängerAdresse").Range.Text = strAdresse
        End With

    End With
    docApp.Activate
Else
    MsgBox "Neue Instanz von Word konnte nicht erzeugt werden."
End If

```

**Listing 9.4** Aus Excel heraus Word fernsteuern und die Empfängeradresse in den Brief eintragen *(Fortsetzung)*

```

    Set docDoc = Nothing
    Set docApp = Nothing
End If
End Sub

```

Sobald die Verbindung zum Word-Objekt aufgebaut ist, steht das ganze Objektmodell zur Verfügung.

Die einzelnen Adresszeilen werden als einzelne Absätze (vbCr) in die Variable eingelesen, damit die Kontrolle der gewählten Adresse in einem Meldungsfeld erfolgen kann. Im Dokument werden die Adresszeilen jedoch mit einer Zeilenschaltung (Chr\$(11)) voneinander getrennt. Aus diesem Grunde wird die Replace-Funktion eingesetzt:

```
strAdresse = Replace(strAdresse, vbCr, vbVerticalTab)
```

Die Zuweisung der Adressdaten an die entsprechende Textmarke wird unter Verwendung des Range-Objekts ausgeführt. Im Zusammenhang mit Makros bzw. der Fernsteuerung von Word wurde schon mehrmals darauf hingewiesen, dass unbedingt das Range-Objekt anstelle des Selection-Objekts zum Einsatz gelangen sollte. Mehr zum Range-Objekt kann in Kapitel 6 nachgeschlagen werden.

```
docDoc.Bookmarks("EmpfängerAdresse").Range.Text = strAdresse
```



Die dargestellten Beispiele finden Sie in der Beispieldatei *Bsp09\_02.xls*. Als zusätzliche Hilfsdateien werden die drei Dokumentvorlagen *Bsp09\_03.dot*, *Bsp09\_04.dot* und *Bsp09\_Brief.dot* benötigt. Sämtliche Dateien befinden sich auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap09*.

## Versteckte Word-Instanz steuern



Zum Steuern einer anderen Applikation soll ein einfaches Szenario aufgebaut werden. Die Aufgabe besteht darin, eine bestehende Datei mit einer unsichtbaren Instanz des zugeordneten Programms zu öffnen und ihre erste Seite auf dem Standarddrucker auszugeben. Anschließend soll die betreffende Datei geschlossen und die neu erstellte Programminstanz beendet werden.

In diesem Fall wird Word von Word heraus gesteuert. Das gleiche Szenario wird in Kapitel 10 beim Fernsteuern von weiteren Anwendungen mehrmals zum Einsatz kommen.

**Listing 9.5** Ausdrucken eines Dokuments mittels Fernsteuerung von Word

```

Sub Demo_WordFernsteuern()
    Const strDATEI_NAME As String = "C:\WordBuch\Beispiele\Kap09\Bsp_Demo.doc"

    System.Cursor = wdCursorWait

    'Variablen und Objekte anlegen
    Dim docApp As Word.Application
    Dim docDoc As Word.Document

```

Listing 9.5 Ausdrucken eines Dokuments mittels Fernsteuerung von Word

```

'Neue Word-Instanz erzeugen
Set docApp = New Word.Application

'Prüfen, ob eine neue Instanz erzeugt werden konnte
If Not (docApp Is Nothing) Then
    With docApp

'Applikation am Bildschirm verbergen
        .Visible = False

        Set docDoc = .Documents.Open( _
            FileName:=strDATEI_NAME, _
            AddToRecentFiles:=False) _

'Dokument ausdrucken
        With docDoc
            .PrintOut Background:=False, Copies:=1,
                Range:=wdPrintFromTo, From:="1", To:="1"

            .Saved = True
            .Close
        End With
        .Quit
    End With
Else
    MsgBox "Neue Instanz von Word konnte nicht erzeugt werden."
End If

Set docDoc = Nothing
Set docApp = Nothing
System.Cursor = wdCursorNormal
End Sub

```

Anhand der eingefügten Kommentarzeilen sollte das Listing 9.5 selbsterklärend sein. Trotzdem sollen zwei Programmzeilen detaillierter besprochen werden.

Die Open-Methode für ein Dokument stellt den optionalen Parameter `AddToRecentFiles` zur Verfügung. Wird, wie im Beispiel, `False` als Wert übergeben, wird die geöffnete Datei nicht in die Liste der zuletzt geöffneten Dateien aufgenommen:

```
Set docDoc = .Documents.Open(FileName:=strDATEI_NAME, AddToRecentFiles:=False)
```

Damit nur die erste Seite des Dokuments und nicht gleich das ganze Dokument auf dem Drucker ausgegeben wird, muss für die `PrintOut`-Methode des Dokuments der entsprechende Seitenbereich als Parameter übergeben werden. In unserem Fall müssen beide Werte auf eins (1) und der Druckbereich (Range) zusätzlich auf `wdPrintFromTo` gesetzt werden:

```
docDoc.PrintOut Background:=False, Range:=wdPrintFromTo, From:="1", To:="1", Copies:=1
```

Als weiterer Parameter wird der `PrintOut`-Methode der Wert `Background:=False` übergeben. Damit ist sichergestellt, dass zuerst der ganze Ausdruck erstellt wird, bevor das Makro weiter abgearbeitet

wird. In unserem Beispiel könnte dies sonst zur Folge haben, dass Word bereits geschlossen wird, bevor der Ausdruck in die Druckerwarteschlange gestellt werden konnte.

Die Möglichkeiten der PrintOut-Methode und deren Argumente wurden bereits in Kapitel 6 vorgestellt.

**WICHTIG** Wir Autoren empfehlen, grundsätzlich bei jeder Verwendung der PrintOut-Methode den Parameter Background:=False zu setzen. So ist sichergestellt, dass das Makro synchron abgearbeitet wird.



Das dargestellte Beispiel finden Sie in der Beispieldatei *Bsp09\_01.doc* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap09*.

## Programmzeilen zentral verwalten und gemeinsam nutzen

Wird die Umgebung von Word durch eine große Anzahl von Makros erweitert, so werden einzelne Funktionen und Prozeduren schon bald mehrmals verwendet. Als Beispiele können die Dateisystem-Operationen aus Kapitel 3 oder die deklarierten APIs aus Kapitel 4 angeführt werden.

Solange diese Funktionen in der gleichen *.dot*-Datei (Dokumentvorlage) gespeichert sind, können sie in einem eigenen Modul stehen und als öffentliche Funktion (Public Function) bzw. Prozedur (Public Sub) deklariert werden. So stehen die entsprechenden Programmzeilen innerhalb des ganzen Projekts uneingeschränkt zur Verfügung.

Werden jedoch mehrere Dokumentvorlagen entwickelt, müssen diese allgemeinen Funktionen entweder in jeder Datei zur Verfügung stehen oder an einer zentralen Stelle gespeichert werden.

Im ersten Fall müssen bei einer Anpassung einer Funktion sämtliche betroffenen Dokumentvorlagen geändert und neu an die Anwender verteilt werden. Um diesen unnötigen Wartungsaufwand zu eliminieren, wird hier die zweite Möglichkeit erläutert: Funktionen *zentral* zur Verfügung stellen.

## Dokumentvorlagen (.dot) als Add-In



Vorlage  
als Add-  
In

Aus der Sicht von Word kann eine *.dot*-Datei für zwei Aufgaben eingesetzt werden:

- Als *Dokumentvorlage*, wenn sie im Benutzervorlagen- oder Arbeitsgruppenvorlagen-Ordner abgespeichert wurde.
- Als *Add-In*, wenn die Datei im *StartUp*-Ordner von Word gespeichert oder über den Menübefehl *Extras/Vorlagen und Add-Ins* manuell als Add-In geladen wurde.

**WICHTIG** Damit in einem VBA-Projekt die allgemeinen Funktionen und Prozeduren eines Add-Ins verwendet werden können, muss ein Verweis auf das betreffende Add-In gesetzt werden. Wie ein Verweis einem Projekt hinzugefügt werden kann, wurde in Kapitel 8 beschrieben.

Dieser Verweis muss nicht gesetzt werden, wenn nur ein direkter Aufruf der Makros mittels der Run-Methode erfolgt (vgl. den Abschnitt »Makros von außen anstoßen« in diesem Kapitel).

Das zentrale Add-In mit den allgemeinen Funktionen und Prozeduren sollte aus zwei Gründen unbedingt im *Startup*-Ordner von Word abgespeichert werden:

- Das Add-In wird bei jedem Start von Word automatisch geladen.
- Der Verweis auf das Add-In wird von Word dynamisch aktualisiert, wenn die beiden Dateien (Dokumentvorlage und Add-In) auf einer anderen Arbeitsstation mit einer anderen Verzeichnisstruktur zum Einsatz kommen.

## Funktionen bzw. Prozeduren aus Add-Ins aufrufen



Das Beispiel in Listing 9.6 nutzt zur Berechnung des Alters die allgemeine Funktion, die im Add-In zur Verfügung gestellt wird. Dank des Verweises auf das Add-In wird der Entwickler während der Erfassung der Programmzeilen durch IntelliSense unterstützt (Abbildung 9.2).

**Listing 9.6** Aufruf der allgemeinen Funktion *funcAlterBestimmen* aus dem Add-In

```
Sub Demo_AlterBestimmen()
    Dim dateMeinGeburtstag As Date

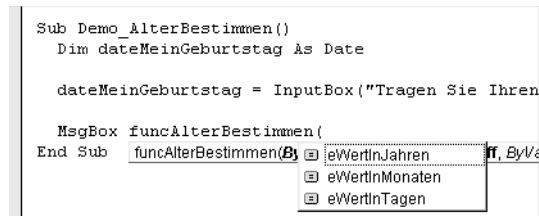
    dateMeinGeburtstag = InputBox("Tragen Sie Ihren Geburtstag ein.", , "24.12.2001")

    MsgBox funcAlterBestimmen(eWertInJahren, dateMeinGeburtstag), vbInformation, cAppName
    MsgBox funcAlterBestimmen(eWertInMonaten, dateMeinGeburtstag), vbInformation, cAppName
    MsgBox funcAlterBestimmen(eWertInTagen, dateMeinGeburtstag), vbInformation, cAppName
End Sub
```

In einem Add-In können neben den allgemeinen Funktionen und Prozeduren zusätzlich auch Konstanten, Variablen sowie benutzerdefinierte Datentypen und Aufzählungen definiert werden. Weitere Informationen zur Deklaration dieser Konstrukte sind in Kapitel 3 zu finden.

In diesem Beispiel wurde die Aufzählung *EDateDiff* und die Konstante *cAppName* zentral im Add-In deklariert.

**Abbildg. 9.2** Die Unterstützung durch IntelliSense funktioniert bei einem *.dot*-Add-In ebenfalls.



Die zentrale Funktion zur Berechnung des Alters kann das entsprechende Resultat in einem unterschiedlichen Format zurückgeben. In Listing 9.7 wird gezeigt, wie das gewünschte Rückgabeformat als Parameter an die Funktion übergeben werden kann.

**Listing 9.7** Zentrale Funktion zur Berechnung des Alters

```
Public Enum EDateDiff
    eWertInTagen = 0
    eWertInMonaten = 1
```

**Listing 9.7** Zentrale Funktion zur Berechnung des Alters (*Fortsetzung*)

```

    eWertInJahren = 2
End Enum

Public Function funcAlterBestimmen( _
    ByVal intInterval As EDateDiff, _
    ByVal dateGeburtstag As Date) As String

    Dim strInterval() As Variant
    Dim strIntervalEinheit() As Variant
    Dim lngAlter As Long

    strInterval() = Array("d", "m", "yyy")
    strIntervalEinheit() = Array("Tage", "Monate", "Jahre")

    lngAlter = DateDiff(strInterval(intInterval), dateGeburtstag, Now)
    funcAlterBestimmen = CStr(lngAlter) & " " & strIntervalEinheit(intInterval)
End Function

```

Der Parameter `intInterval` ist vom Typ `EDateDiff`. Dies stellt sicher, dass die Unterstützung durch `IntelliSense` funktioniert und beim Erfassen der Programmzeile die entsprechende Auswahl angezeigt wird (Abbildung 9.2).

```
ByVal intInterval As EDateDiff
```

Der Wert aus dem Parameter nimmt gleichzeitig Bezug auf die beiden Datenfeldvariablen. Diese wurden bewusst so bestückt, damit die Position der Werte mit dem Zahlenwert aus der Aufzählung übereinstimmt. Somit kann ohne `If...Then`- oder `Select Case`-Anweisungen eine Funktion definiert werden, die sich unterschiedlich verhält:

```

lngAlter = DateDiff(strInterval(intInterval), dateGeburtstag, Now)
funcAlterBestimmen = CStr(lngAlter) & " " & strIntervalEinheit(intInterval)

```



Das dargestellte Beispiel aus Listing 9.6 finden Sie in der Beispieldatei *Bsp09\_06.doc*. Als zusätzliche Hilfsdatei wird die Datei *Bsp09\_05.dot* benötigt, welche in den *Startup*-Ordner von Word kopiert werden muss. Sämtliche Dateien befinden sich auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap09*.

## UserForm aus Add-In aufrufen



Bis jetzt wurde aufgezeigt, wie in einem zentralen Add-In allgemeine Funktionen und Prozeduren sowie globale Konstanten und Variablen zur Verfügung gestellt werden können. Neben diesen Elementen können auch benutzerdefinierte Dialogfelder (UserForm-Objekte) zentral gespeichert und aus verschiedenen Dokumentvorlagen heraus aufgerufen werden. Informationen zum Thema »benutzerdefinierte Dialogfelder« können dem Kapitel 14 entnommen werden.

Da es sich bei einem UserForm-Objekt um eine spezielle Art von Klassen handelt, besteht keine Möglichkeit, diese direkt aus dem aktuellen Makro heraus aufzurufen oder eine neue Instanz des



UserForm-Objekts zu erzeugen und zu starten. Stattdessen muss wie in Listing 9.8 der Umweg über eine allgemeine Funktion im Add-In gewählt werden, die diesen Aufruf erledigt.

Ein aktiver Datenaustausch aus dem Makro zur UserForm und zurück muss ebenfalls über diese zusätzliche Funktion abgewickelt werden. Die entsprechenden Programmzeilen sind in Listing 9.9 aufgeführt.

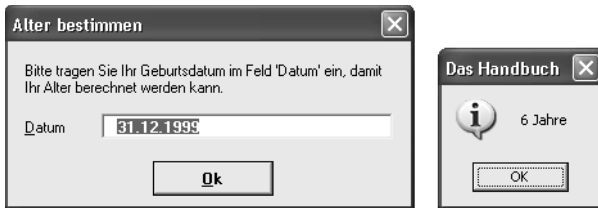
**Listing 9.8** Indirekter Aufruf der UserForm über die allgemeine Funktion *funcUserFormAlterBestimmen*

```
Sub Demo_AlterBestimmen_UserForm()
    Dim dateMeinGeburtstag As Date

    dateMeinGeburtstag = funcUserFormAlterBestimmen()

    MsgBox funcAlterBestimmen(eWertInJahren, dateMeinGeburtstag), vbInformation, cAppName
End Sub
```

**Abbildg. 9.3** Abfrage des Alters anhand eines benutzerdefinierten Dialogfelds, das in einem Add-In hinterlegt ist



Die gesamte Funktionalität für den einwandfreien Aufruf des Dialogfelds sowie die Kommunikation zwischen dem Makro und der UserForm muss in der allgemeinen Funktion *funcUserFormAlterBestimmen* abgewickelt werden.

**Listing 9.9** Datenaustausch zwischen der UserForm und dem aufrufenden Makro

```
Public Function funcUserFormAlterBestimmen() As String
    frmAlterBestimmen.Show
    funcUserFormAlterBestimmen = frmAlterBestimmen.txtDatum.Text
    Unload frmAlterBestimmen
End Function
```



Das Beispiel aus Listing 9.8 finden Sie in der Beispieldatei *Bsp09\_06.doc*. Als zusätzliche Hilfsdatei wird die Datei *Bsp09\_05.dot* benötigt, welche in den *Startup*-Ordner von Word kopiert werden muss. Sämtliche Dateien befinden sich auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap09*.

## Add-In-Prozeduren in anderen Umgebungen nutzen

Sollen die allgemeinen Prozeduren des Add-Ins aus einer anderen Programmumgebung heraus genutzt werden, geschieht dies am einfachsten mit Hilfe der *Run*-Methode, wie zu Beginn des Kapi-

tels beschrieben. Eventuell in solchen Prozeduren verursachte Meldungen (wie MsgBox-Funktionen) finden in der Word-Umgebung statt und müssen in dieser Umgebung quittiert werden.

Soll jedoch neben den allgemeinen Prozeduren auch auf die integrierten Funktionen des Add-Ins zugegriffen werden, kann der Aufruf nicht mehr mittels der Run-Methode erfolgen. In diesem Fall muss die Datei explizit als Document-Objekt in Word geöffnet werden und der Aufruf der Prozeduren und Funktionen darüber erfolgen. Zudem müssen sich diese Prozeduren im *ThisDocument*-Klassenmodul der Vorlage befinden. Nur dann werden sie als Eigenschaften des Document-Objekts erkannt.

Das Listing 9.10 veranschaulicht die Vorgehensweise. Die Prozedur *WordFernsteuern* befindet sich in einem Excel-Modul; es könnte aber genauso gut in Visual Basic oder PowerPoint sein. Eine laufende Instanz der Word-Anwendung wird durch *GetObject* instanziiert und zuvorderst gebracht, da das Meldungsfeld der Prozedur *ZentimeterNachZoll* in der Word-Umgebung eingeblendet wird.

Danach wird das Add-In als ein Dokument unsichtbar geöffnet und zwei weitere Prozeduren aufgerufen. Diese befinden sich im *ThisDocument*-Modul und rufen ihrerseits die gleichen beiden Prozeduren auf, wie im vorherigen Abschnitt beschrieben. Die UserForm wird in der Word-Umgebung eingeblendet, während das MsgBox-Ergebnis in Excel erscheint (weil MsgBox in diesem Fall von einem Excel-Modul aus aufgerufen wird).

**Listing 9.10** Prozeduren in einem Word-Add-In außerhalb von Word aufrufen

```
Sub WordFernsteuern()
    Dim wdapp As Word.Application
    Dim doc As Word.Document
    Dim sDateiName As String
    Dim dateMeinGeburtstag As Date

    On Error Resume Next
    Set wdapp = GetObject("Word.Application.11")
    If Err.Number = 429 Then
        'Keine Word-Instanz vorhanden
        Exit Sub
    End If
    On Error GoTo 0
    'Bildschirmflackern minimieren
    wdapp.ScreenUpdating = False
    'Word zuvorderst stellen
    wdapp.Activate
    'Wird in der Word-Umgebung eingeblendet
    wdapp.Run "ZentimeterNachZoll", 2.54

    'Um eine Prozedur in einem Add-In aufzurufen,
    'muss dieses als Dokument geöffnet werden.
    'Dateipfadangabe vom Add-In ermitteln
    sDateiName = wdapp.AddIns("Bsp09_05.dot").Path & "\" _
        & wdapp.AddIns("Bsp09_05.dot").Name
    'Add-In unsichtbar öffnen
    Set doc = wdapp.Documents.Open(Filename:=sDateiName, _
        AddToRecentFiles:=False, Visible:=False)

    'Funktion im ThisDocument-Modul
    dateMeinGeburtstag = doc.AltersEingabe
    wdapp.Tasks("Microsoft Excel").Activate
    'Wird in Excel angezeigt
    MsgBox doc.AlterBestimmen(2, dateMeinGeburtstag), vbInformation
End Sub
```

Listing 9.10 Prozeduren in einem Word-Add-In außerhalb von Word aufrufen (Fortsetzung)

```

wdapp.Tasks("Microsoft Excel").Activate
'Aufräumen
doc.Close
wdapp.ScreenUpdating = True
Set doc = Nothing
Set wdapp = Nothing
End Sub

'Prozedur im ThisDocument-Modul des Vorlagen-Add-Ins
Public Sub ZentimeterNachZoll(sngCM As Single)
    MsgBox CStr(sngCM) & " ergeben " & _
        CStr(Application.PointsToInches(Application.CentimetersToPoints(sngCM))) & _
        " Zoll.", vbInformation, cAppName
End Sub

```



Das dargestellte Beispiel finden Sie in der Beispieldatei *Bsp09\_07.xls*. Diese befindet sich auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap09*. Es ruft Prozeduren in der Vorlage *Bsp09\_05.dot* auf, die sich im *StartUp*-Ordner von Word befinden muss.

## Fernsteuerung aus Office-fremden Umgebungen

Die Fernsteuerung von Word aus anderen Programmierumgebungen heraus, wie dem klassischen Visual Basic oder Visual Studio, ist grundsätzlich nicht anders als die Fernsteuerung aus einer anderen Office-Anwendung wie Excel. Wie in Kapitel 8 beschrieben, kann sie sowohl über Early als auch Late Binding erfolgen.

Kapitel 6 enthält reichlich Beispiele für die Fernsteuerung der Word-Anwendung aus der .NET-Umgebung. Der Code aller dieser Lösungen ist von Word und seinen Dokumenten abgekoppelt, im Vergleich zu den Prozeduren, die in Word-VBA vorgestellt wurden, die sich innerhalb eines Word-Dokuments oder einer Word-Dokumentvorlage befinden.

Manchmal ist eine anwendungsunabhängige Lösung wie eine .exe-Datei oder ein COM-Add-In genau das, was gesucht wird. Es gibt jedoch Aufgaben, für die eine Datei-orientierte Methode besser geeignet ist: Die Art des Dokuments (Brief, Memo, Bericht) und die Werkzeuge dafür bilden eine Einheit, und die spezialisierte Funktionalität ist nur in diesem Zusammenhang verfügbar. Für die .NET-Umgebung hat Microsoft deshalb VSTO – Visual Studio Tools for Office – entwickelt und bereitgestellt.

### VSTO

Nebst den Vorteilen der .NET-Umgebung, wie der Zugang zu den Framework-Klassen und Windows-Forms, gibt es auch Sicherheits- und administrative Aspekte, die für eine VSTO-Lösung sprechen:

- Makrosicherheit in der Office-Anwendung kann auf »Hoch« gesetzt werden, da sich der Code in einem getrennten .NET-Assembly (.dll) befindet.
- Die Berechtigungen für das Assembly werden im .NET Framework festgelegt und unterliegen den üblichen Bestimmungen (das Assembly kann beispielsweise mit einem »Strong name« und digitalen Zertifikat signiert werden).
- Da der Code vom Dokument getrennt ist, müssen nicht beide zusammen am gleichen Ort gespeichert werden. Das Dokument kann beispielsweise lokal auf dem Rechner gespeichert werden, während der Code im Netzwerk oder sogar im Internet bereit liegt.
- Weil Code und Dokument getrennt sind, ist es relativ einfach, die Lösung zu aktualisieren, auch wenn mehrere Benutzer mit ihren Dokumentkopien arbeiten.

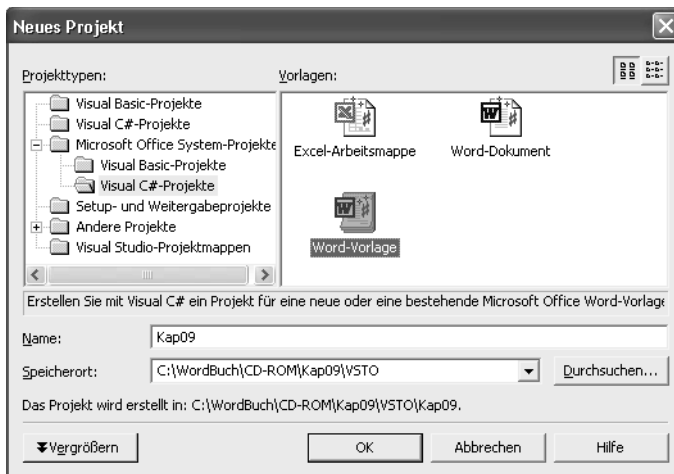
Als Nachteile sind die lange Ladezeit beim Öffnen eines Lösungs-Dokuments sowie die Komplexität der Verteilung und Installation einer Lösung zu nennen. Zudem enthalten nur die Stand-alone- und Office-Professional-Versionen von Word die notwendigen Zusätze, um eine VSTO-Lösung zu laden. Benutzern mit der Standard-Version von Office steht diese Funktionalität nicht zur Verfügung.

VSTO gibt es (bislang) nur für Office 2003 und wurde als Zusatz zu Visual Studio 2003 entwickelt. Ist VSTO installiert, erscheint im Visual Studio-Dialogfeld *Neues Projekt* der Ordner *Microsoft Office System-Projekte* (Abbildung 9.4), worin sich Vorlagen für eine Excel-Arbeitsmappe, ein Word-Dokument sowie eine Word-Dokumentvorlage befinden.

#### HINWEIS

VSTO 2003 ist nur die erste Version. Mit Visual Studio 2005, die im November 2005 offiziell erschienen ist, steht VSTO 2005 zur Verfügung, die Datencaching und einen teilweise intuitiveren Zugang zu den Office-Objektmodellen bereitstellen. Da Microsoft für die Zukunft eindeutig auf die Karte .NET setzt, ist zu erwarten, dass VSTO mit der Zeit immer besser in die Office-Anwendungen integriert werden. Die Grundlagen sind jedoch gleich geblieben, wie hier vorgestellt.

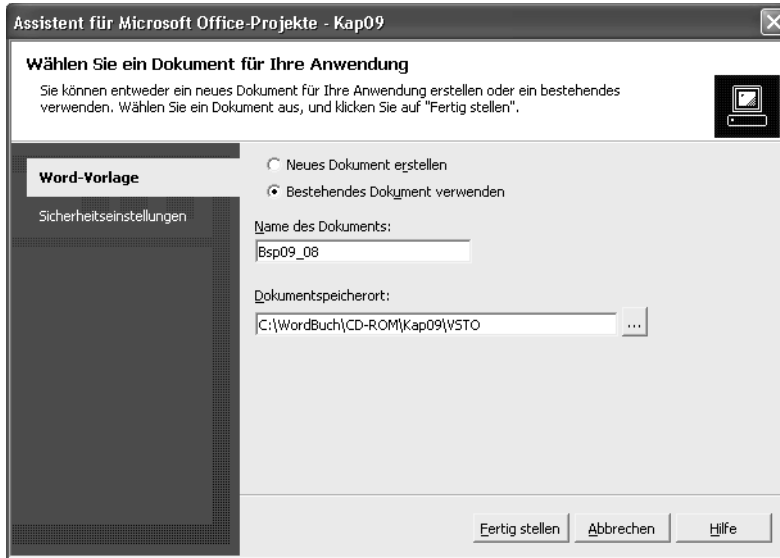
Abbildung 9.4 VSTO installiert einen eigenen Ordner mit vordefinierten Projektarten.



Beim Erstellen des neuen Projekts kann eine vorhandene Datei als Grundlage dienen, oder der Assistent erstellt ein neues Dokument der ausgewählten Art. Dies wird im Assistenten-Dialogfeld in

Abbildung 9.5 festlegt. Falls eine vorhandene Datei genutzt werden soll, muss der Dateipfad in das Textfeld *Dokumentspeicherort* eingetragen werden.

**Abbildg. 9.5** Festlegen, ob das Projekt auf einem neuen oder vorhandenen Dokument basieren soll



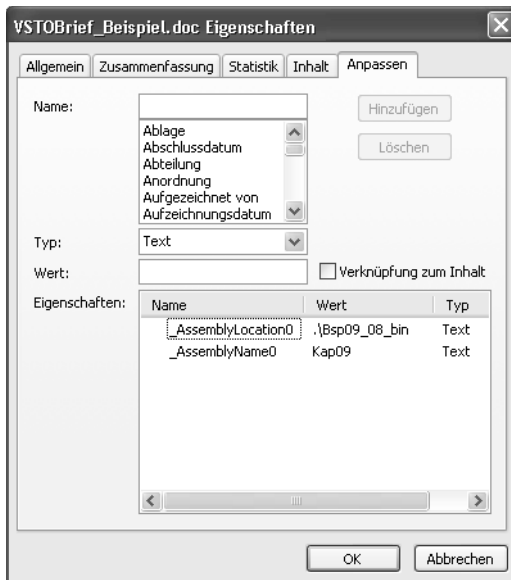
Unabhängig von der gewählten Option kann das Dokument problemlos in seiner Anwendung geöffnet und weiterbearbeitet werden. Von einem herkömmlichen Dokument unterscheidet es sich lediglich durch zwei benutzerdefinierte Dokumenteigenschaften (*CustomDocumentProperties*), die der Anwendung mitteilen, dass das Dokument mit einem »managed« Code-Assembly zu verbinden ist: *\_AssemblyLocation0* sowie *\_AssemblyName0*. Diese enthalten den Pfad zum Assembly und seinen Namen, so dass der »Office Loader« (der Office 2003-Programmteil, der die Verbindung zum »managed« Code herstellt) weiß, wo sie zu suchen sind (siehe Abbildung 9.6).

#### HINWEIS

Assemblies sind die Grundbausteine von .NET Framework-Anwendungen. Sie bilden die Basiseinheit für Weitergabe, Versionskontrolle, Wiederverwendung, Aktivierungs-Scoping und Sicherheitsberechtigungen. Ein Assembly ist eine Auflistung von Typen und Ressourcen, die so erstellt wurden, dass sie zusammenarbeiten und eine logische funktionelle Einheit bilden.

#### HINWEIS

VSTO 2005 in Visual Studio 2005 stellt für die Bearbeitung des Lösungs-Dokuments einen internen Entwurfmodus zur Verfügung, so dass nicht immer zwischen der Office-Anwendung und Visual Studio gewechselt werden muss. Das Dokument wird beim Öffnen des Projekts automatisch darin geöffnet.

**Abbildg. 9.6** Die von VSTO benötigten benutzerdefinierten Eigenschaften im gleichnamigen Dialogfeld von Word


Der Assistent erstellt ein Code-Projekt, das im Vergleich zu einem herkömmlichen Visual Studio-Projekt, wie es bislang im Buch vorgestellt wurde, Verschiedenes für den Entwickler schon erledigt hat. Verweise zu bestimmten Office- und Anwendungsbibliotheken sind beispielsweise schon vorhanden:

```
using Office = Microsoft.Office.Core;
using Word = Microsoft.Office.Interop.Word;
using MSForms = Microsoft.Vbe.Interop.Forms;
```

Weiter werden Programmzeilen eingefügt, die das Projekt mit dem »Office Loader« verbinden. Die Klasse heißt immer `OfficeCodeBehind`, und es werden immer Eigenschaften (sowie die dazu gehörenden Objektvariablen (Felder)) für die Office-Anwendung und -Dokument vordefiniert:

```
internal Word.Application ThisApplication
{
    get { return thisApplication;}
}
internal Word.Document ThisDocument
{
    get { return thisDocument;}
}

private Word.Application thisApplication = null;
private Word.Document thisDocument = null;
```

Zudem werden Ereignisse wie `Document_Open`, `Document_New` sowie `Document_Close` definiert und dafür Prozedurskelette bereitgestellt. Der Entwickler muss lediglich die Programmzeilen eingeben,

die bei der Auslösung des Ereignisses auszuführen sind. Hier dienen als Beispiel die standardmäßigen Angaben für `Document_Close`:

```
private Word.DocumentEvents2 CloseEventHandler closeEvent;
protected void ThisDocument_Close()
{
    // Ensure that Close is not called twice.
    designModeClose = false;
}
```

Noch etwas erledigt der VSTO-Assistent, der das Projekt erstellt: er legt die Sicherheitsberechtigungen fest, so dass das Dokument auf dem Entwicklungsrechner voll lauffähig ist. Und wahrlich, die Wahl des Deployments sowie die Festlegung der Berechtigungen sind die größten Herausforderungen für den im .NET-Framework unerfahrenen Entwickler. Eine eingehende Diskussion dieser Materie übersteigt den Rahmen dieses Buches. Wir verweisen wieder auf das Werk »Microsoft .NET Development for Microsoft Office« von Andrew Whitechapel (Microsoft Press) sowie die Dokumentation auf der Microsoft-Webseite (unter <http://msdn.microsoft.com/smartclient/understanding/vsto>).

## Beispiel-Lösung

Als Beispiel wurde eine Dokumentvorlage für Firmenbriefe gewählt. Das Dokument ist als ein Formular geschützt; der zweite Abschnitt, für den Briefkörper, ist ungeschützt. ActiveX-Steuerelemente stehen für die Benutzereingaben im oberen Teil bereit.

### HINWEIS

Das gleiche Grunddokument diente in Kapitel 6 als Beispiel für Formularfelder und den Seriendruck. Es wird auch in Kapitel 11 im Abschnitt über ActiveX herangezogen, so dass Sie die verschiedenen Funktionalitäten vergleichen können.

Beim Eintreten in ein Steuerelement wird der Textinhalt markiert, so dass der Anwender mit der Eingabe sofort beginnen kann.

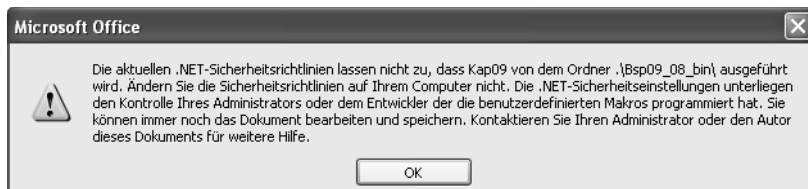
Klickt der Anwender auf die Symbolschaltfläche *Kontaktliste anzeigen*, blendet die Lösung, wie in Abbildung 9.8 zu sehen ist, ein Windows-Formular mit einer Datentabelle ein. (Die Daten werden aus *Nordwind.mdb* dynamisch geholt.) Wählt der Anwender einen Datensatz aus und klickt auf *Daten übernehmen*, werden die Informationen aus den Spalten in die passenden Adressfelder eingetragen.

Diese Form wird auch eingeblendet, wenn der Anwender ein Fragezeichen (?) in das Steuerelement für den Empfänger oder das Land eingibt. Stehen vor dem Fragezeichen Textzeichen, gefolgt von einem Sternchen (\*), werden die Datensätze entsprechend gefiltert. Alle Nachnamen bzw. Länder, die mit diesen Zeichen anfangen, werden aufgelistet. (Beispiel: »US\*?« zeigt alle Datensätze an, die mit den Zeichen »US« anfangen.)

Beim Schließen des Dokuments wird es von der VSTO-Vorlage getrennt und mit *Normal.dot* verbunden. Zudem werden die Symbolleiste sowie die Dokumenteigenschaften `_AssemblyName0` und `_AssemblyLocation0` entfernt. Damit kann der Anwender das Dokument problemlos wieder öffnen und bearbeiten.

**WICHTIG** Standardmäßig wird das Assembly immer relativ zum Dokumentpfad gesucht. Wird es beim Öffnen eines Dokuments nicht gefunden, erscheint die Meldung in Abbildung 9.7 und die Funktionalität steht nicht zur Verfügung, obwohl das Dokument sonst ganz normal bearbeitet werden kann.

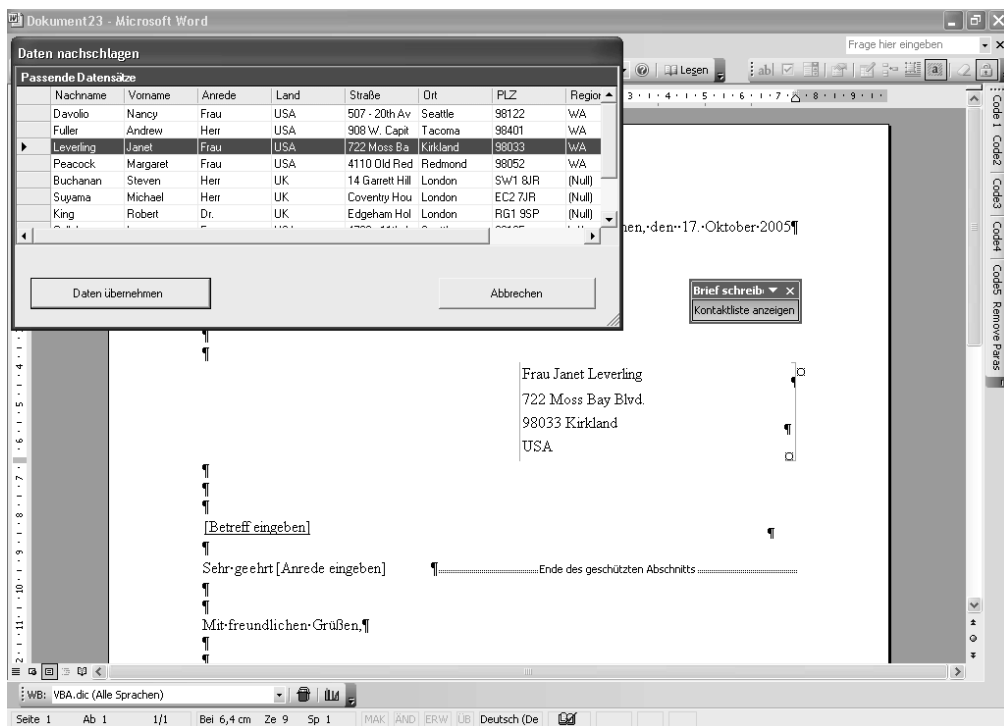
Abbildg. 9.7 Wenn das Assembly nicht gefunden werden kann, zeigt Word eine Meldung an.



Da der Anwender ein Dokument meist in einem anderen Ordner speichern wird (oder muss), ist es besser, ihn mit einer solchen Meldung nicht zu verwirren. Durch Entfernen der Dokumenteigenschaften sowie Verknüpfung mit einer anderen Vorlage »weiß« das Dokument nicht mehr, dass es einst Teil einer VSTO-Lösung war.

Soll das Dokument weiterhin mit der Lösung verbunden bleiben, kann die Dokumenteigenschaft `_AssemblyLocation0` so angepasst werden, dass sie auf den korrekten Pfad zur `.dll` zeigt.

Abbildg. 9.8 Beispieldokument mit Windows-Form. Die Datentabelle ist mit einer Datenbank verbunden.





Document  
\_New

In diesem Beispiel werden die Vorbereitungen in der Ereignis-Prozedur *ThisDocument\_New* (Listing 9.11) getroffen. Dieses Ereignis wird ausgelöst, wenn von der Vorlage ein neues Dokument erstellt wird. Ereignisse werden deklariert und initialisiert, die Steuerelemente werden initialisiert, die Symbolleiste wird erstellt, und das erste ActiveX-Feld wird angesprungen und sein Inhalt markiert. Alle diese Handlungen werden unten einzeln vorgestellt.

**Listing 9.11** Die Ereignis-Prozedur *ThisDocument\_New*

```
protected void ThisDocument_New()
{
    selChangeEvent = new Word.ApplicationEvents4_WindowSelectionChangeEventHandler
        (ThisApplication_WindowSelectionChange);
    ThisApplication.WindowSelectionChange +=new
        Microsoft.Office.Interop.Word.ApplicationEvents4_WindowSelectionChangeEventHandler
        (ThisApplication_WindowSelectionChange);

    SteuerelementenInitialisieren();
    SymbolleisteErstellen();
    //Das Datum aktualisieren
    ThisDocument.Fields[1].Update();
    //Den Inhalt der ersten Textbox markieren
    Word.Selection sel = thisApplication.Selection;
    Word.InlineShape ils;
    if (IstMarkierungSteuerelement(sel, out ils))
    {
        object aktuellesElement = ils.OLEFormat.Object;
        if (IstTextElement(aktuellesElement))
        {
            MSForms.TextBox tb = (MSForms.TextBox) aktuellesElement;
            TextInhaltMarkieren(tb);
        }
    }
    //Ende ActiveX-Textfeld
}
```

Symbol-  
leiste  
erstellen

Die Einzelheiten zum Erstellen einer Symbolleiste werden in Kapitel 15 erläutert; der Code für dieses Beispiel steht in Listing 9.12. Bitte beachten Sie, dass das Argument *Temporary* (temporär) auf »Wahr« gesetzt wird. Damit verschwindet die Symbolleiste aus dem Dokument, sobald dieses geschlossen wird; sie muss damit nicht explizit entfernt werden.

Beim Hinzufügen der Symbolschaltfläche ist darauf zu achten, dass Office ein Steuerelement eines allgemeinen Typs zur Verfügung stellt: *Office.MsoControlType.msoControlButton*. Dieses muss in C# oder VB.NET explizit mit aktivem *Option Explicit* in eine Schaltfläche der gewünschten Art umwandelt werden (*Office.CommandBarButton*).

**Listing 9.12** Eine Symbolleiste im Dokument erstellen

```
private void SymbolleisteErstellen()
{
    Office.CommandBar cb;
    try
    {
        object objMissing = System.Reflection.Missing.Value;
        ThisApplication.CustomizationContext = ThisDocument;
        cb = ThisApplication.CommandBars.Add("Brief schreiben",
```

**Listing 9.12** Eine Symbolleiste im Dokument erstellen (Fortsetzung)

```

        Office.MsoBarPosition.msoBarFloating, false, true);
        Office.CommandBarButton btn = (Office.CommandBarButton)
            cb.Controls.Add(Office.MsoControlType.msoControlButton,
                objMissing, objMissing, objMissing, objMissing);
        btn.Caption = "Kontaktliste anzeigen";
        btn.Enabled = true;
        btn.Style = Office.MsoButtonStyle.msoButtonCaption;
        btn.ToolTipText = "Formular mit Liste der Kontakten zur Auswahl einblenden";
        btn.Visible = true;
        btn.Click += new Microsoft.Office.Core._CommandBarButtonEvents_ClickEventHandler
            (btn_Click);
        cb.Visible = true;
    }
    catch (System.Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}

```

Im Gegensatz zur Erstellung einer Symbolschaltfläche in einem Office-VBA-Projekt wird hier die Eigenschaft `OnClick` nicht benutzt. Stattdessen muss ein Klick-Ereignis explizit deklariert werden. Das `btn_Click`-Ereignis ruft die Prozedur *DatenNachschlagen* auf (mehr darüber weiter unten).

Ereignisse

Alle beschriebenen Handlungen dieser Lösung werden durch Ereignisse ausgelöst:

- Die Markierung des Textinhalts beim Eintreten durch das Anwendungsereignis `WindowSelectionChange`
- Die Einblendung des Windows Formulars durch das `Change`-Ereignis des Steuerelements bzw. das `Click`-Ereignis der Symbolschaltfläche
- Die Umwandlung in ein »normales« Word-Dokument durch das `ThisDocument_Close`-Ereignis beim Schließen des Dokuments

Mit Ausnahme des `ThisDocument_Close`-Ereignisses müssen alle vom Entwickler explizit deklariert werden.

**HINWEIS**

Mehr über Ereignisse in Word finden Sie in Kapitel 7. In diesem Kapitel werden nur die VSTO-spezifischen Aspekte behandelt.

Window-  
Selection-  
Change

Das Ereignis `WindowSelectionChange` wird wie folgt in der Prozedur *ThisDocument\_New* deklariert:

```
private Word.ApplicationEvents4_WindowSelectionChangeEventHandler selChangeEvent;
```

**HINWEIS**

Es fällt auf, dass Dokument- sowie Anwendungsereignisse im IntelliSense mehrmals aufgelistet sind, lediglich gekennzeichnet durch eine Ziffer wie 2, 3 oder 4. Diese Ziffern entsprechen den Word-Versionen 2000, 2002 und 2003. Nicht alle Ereignisse werden in jeder dieser Word-Versionen unterstützt. Zudem ist es möglich, dass sie sich unterschiedlich verhalten. Programmieren Sie eine Anwendung, die mit mehreren Word-Versionen arbeiten soll, muss das Ereignis für die älteste Version gewählt werden. Da VSTO bislang nur mit Office 2003 eingesetzt werden kann, stellt sich die Frage (noch) nicht.

Die eigentliche Handlung, die das Ereignis ausführt, befindet sich in der spezifizierten Prozedur *ThisApplication\_WindowSelectionChange* (Listing 9.13). Da dieses Ereignis durch fast alle Handlungen im Dokument, außer der Texteingabe, ausgelöst wird, wird als allererstes geprüft, ob sich die Markierung in einem Steuerelement befindet. Wenn ja, wird geprüft, ob es vom Typ »Textbox« ist. Erst dann wird die Markierung auf den ganzen Text erweitert.

**HINWEIS** Eigentlich besitzt ein Steuerelement die Eigenschaft *EnterFieldBehavior*, die auf *frmEnterFieldBehaviorSelectAll* (alles markieren) festgelegt werden kann. Diese hat jedoch keine Wirkung, wenn das Steuerelement in einem Dokument eingebettet ist.

**Listing 9.13** Diese *WindowSelectionChange*-Ereignisprozedur prüft, ob sich die Markierung in einem Steuerelement befindet.

```
private void ThisApplication_WindowSelectionChange(
    Microsoft.Office.Interop.Word.Selection Sel)
{
    try
    {
        Word.Selection sel = thisApplication.Selection;
        //Für ein ActiveX-Textfeld den Inhalt markieren
        Word.InlineShape ils;
        if (IstMarkierungSteuerelement(sel, out ils))
        {
            object aktuellesElement = ils.OLEFormat.Object;
            if (IstTextElement(aktuellesElement))
            {
                MSForms.TextBox tb = (MSForms.TextBox) aktuellesElement;
                TextInhaltMarkieren(tb);
            }
        }
        //Ende ActiveX-Textfeld
    }
    catch (System.Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}
```

Change-  
Ereignis

Wie Listing 9.13 andeutet, hält Word ActiveX-Steuerelemente für eingebettete Grafiken (als *InlineShapes* oder *Shapes*, siehe dazu den Abschnitt über Grafiken in Kapitel 6). Es ist daher nur über Umwege möglich, ein Steuerelement als solches zu identifizieren oder ein spezifisches anzusprechen. VSTO leistet in dieser Hinsicht mit der vom Assistenten erstellten Funktion *FindControl* etwas Abhilfe. Ihr wird eine Zeichenkette mit dem Namen des Steuerelements übergeben; sie gibt das Steuerelement als Objekt zurück.

Ihr Einsatz wird in Listing 9.14 veranschaulicht. Das Steuerelement für die Empfängeranschrift wird als Feld deklariert und in der Prozedur *SteuerelementenInitialisieren* (die beim Erstellen des Dokuments aufgerufen wurde) mittels *FindControl* festgelegt. Somit steht das Steuerelement während der Laufzeit immer zur Verfügung.

Dann wird das Change-Ereignis – das durch Texteingabe im Feld ausgelöst wird – deklariert.

**TIPP**

Wenn Sie für ein Steuerelement den Steuerelementnamen eingeben, gefolgt von einem Punkt, zeigt IntelliSense eine Liste der Eigenschaften, Methoden und Ereignisse an. Wählen Sie das *Change*-Ereignis, geben Sie dann nacheinander die Zeichen **+** und **=** ein. IntelliSense zeigt ein QuickInfo an, die die »new«-Deklaration vorschlägt. Drücken Sie die **↵**-Taste, um den Vorschlag zu übernehmen. Drücken Sie nochmals die **↵**-Taste, um auch den vorgeschlagenen Prozedurnamen für das Ereignis zu übernehmen. Somit bleiben Ihnen etliche Tastenschläge erspart.

**Listing 9.14** Ein Steuerelement deklarieren und instanzieren sowie dafür ein Ereignis festlegen

```
private MSForms.TextBox txtEmpfänger;
private void SteuerelementenInitialisieren()
{
    try
    {
        txtEmpfänger = (MSForms.TextBox) this.FindControl("txtEmpfänger");
        txtEmpfänger.Change +=new
            Microsoft.Vbe.Interop.Forms.MdcTextEvents_ChangeEventHandler(txtEmpfänger_Change);
        //Weitere Codezeilen aus Platzgründen unterdrückt...
    }
    catch (System.Exception ex)
    {
        MessageBox.Show(ex.Message + "\n" + ex.Source);
    }
}
```

Die Ereignishandlung für dieses Beispiel sehen Sie in Listing 9.15. Sie kontrolliert das letzte Zeichen im Steuerelement. Falls es sich um ein Fragezeichen handelt, wird die Zeichenkette für den Datenfilter zusammengestellt und dann die Prozedur aufgerufen, die das Windows-Formular mit der Datentabelle einblendet.

**Listing 9.15** Prozedur für das *Change*-Ereignis eines Steuerelements

```
private const string datenNachschlagZeichen = "?";
private void txtEmpfänger_Change()
{
    if (txtEmpfänger.Text.EndsWith(datenNachschlagZeichen))
    {
        string filter = txtEmpfänger.Text;
        DatenNachschlagen(DataFilter(filter), "Nachname");
    }
}
```

Document  
\_Close-  
Ereignis

Die Deklaration und Instanzierung des Ereignisses wurden in der *\_Startup*-Prozedur, erstellt vom VSTO-Assistenten, vorgenommen. Die Ereignis-Prozedur *Document\_Close* ist in Listing 9.16 enthalten. Die ersten vier sowie die letzte Programmzeile wurden ebenfalls vom VSTO-Assistenten erzeugt. Die Zeilen ab dem Kommentar »Dokument von VSTO abhängen« wurden vom Entwickler hinzugefügt.

Falls es sich um ein Dokument und keine Vorlage handelt, wird die Vorlage *Normal.dot* angehängt.

Dann werden die Dokumenteigenschaften *\_AssemblyName0* sowie *\_AssemblyLocation0* gelöscht. Da Dokumenteigenschaften in C# nur über Late Binding angesprochen werden können, ist das Vorgehen etwas involviert, wie die Prozedur *DokEigenschaftenLöschen* veranschaulicht.

Listing 9.16 Beim Schließen das Dokument von der VSTO-Lösung abhängen

```

protected void ThisDocument_Close()
{
    // Sicherstellen, dass 'Close' nicht zweimal aufgerufen wird.
    designModeClose = false;
    //Dokument vom VSTO abhängen
    try
    {
        if (thisDocument.Type == Word.WdDocumentType.wdTypeDocument)
        {
            object objNormalTemplate = (object) thisApplication.NormalTemplate;
            thisDocument.set_AttachedTemplate(ref objNormalTemplate);

            object[] eigenschaften = new object[] { "_AssemblyName0", "_AssemblyLocation0";
            DokEigenschaftenLoeschen(eigenschaften);
        }
    }
    catch (System.Exception ex)
    { MessageBox.Show(ex.Message); }
}

private void DokEigenschaftenLoeschen(object[] liste)
{ //Liste enthält ein Datenfeld (Array) mit allen Dokumenteigenschaften,
  //die gelöscht werden sollen.
  try
  {
      object propAssemblyProps = thisDocument.CustomDocumentProperties;
      Type typeCustomProps = propAssemblyProps.GetType();
      object[] objArgs = new object[] { };
      if (propAssemblyProps != null)
      {
          //Für jedes Element des Datenfelds ...
          for (int i = 0; i < liste.Length; i++)
          {
              object[] propName = new object[] { liste[i] };
              //Die Dokumenteigenschaft einem Objekt zuweisen ...
              object propAssemblyProp = typeCustomProps.InvokeMember("Item",
                  BindingFlags.Default | BindingFlags.GetProperty, null,
                  propAssemblyProps, propName);
              Type typeProp = propAssemblyProp.GetType();
              //... und dieses löschen.
              typeProp.InvokeMember("Delete", BindingFlags.Default | BindingFlags.InvokeMethod,
                  null, propAssemblyProp, objArgs);
          }
      }
  }
  catch (System.Exception ex)
  { MessageBox.Show(ex.Message); }
}

```



Die Beispieldatei *Bsp09\_08.dot* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap09\VSTO*; die dazugehörige *.dll* im Unterordner *Bsp09\_08\_bin*. Der Quellcode befindet sich im Unterordner *Kap09* (siehe auch Abbildung 9.9).

## Das Beispiel lauffähig machen

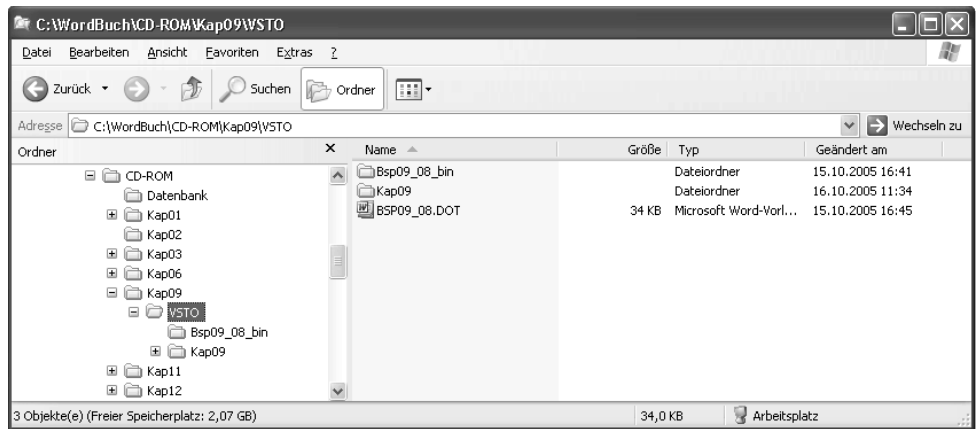
Die Kehrseite der Sicherheitsvorteile ist, dass eine VSTO-Lösung nicht einfach auf den Rechner kopiert und benutzt werden kann. Die Sicherheit muss dafür explizit konfiguriert werden. Mit diesem weit reichenden Thema befasst sich dieses Werk nicht. Wir beschreiben hier einfache Schritte, um das auf der CD-ROM beiliegende Beispiel schnell lauffähig zu machen, weisen aber darauf hin, dass dieses Vorgehen nicht der höchstmöglichen Sicherheitsstufe entspricht.

Sie müssen Visual Studio nicht installiert haben, um die VSTO-Lösung zu öffnen; das .NET-Framework 1.1 muss jedoch auf dem Rechner installiert sein:

1. Das Assembly (*Kap09.dll*) muss sich, relativ zum Projekt-Dokument, in einem Unterordner namens *Bsp09\_08\_bin* befinden.
2. Die *Nordwind.mdb* Datenbank muss sich, relativ zum Projekt-Dokument, zwei Ebenen höher in einem Ordner namens *Datenbank* befinden (Abbildung 9.9).

Abbildg. 9.9

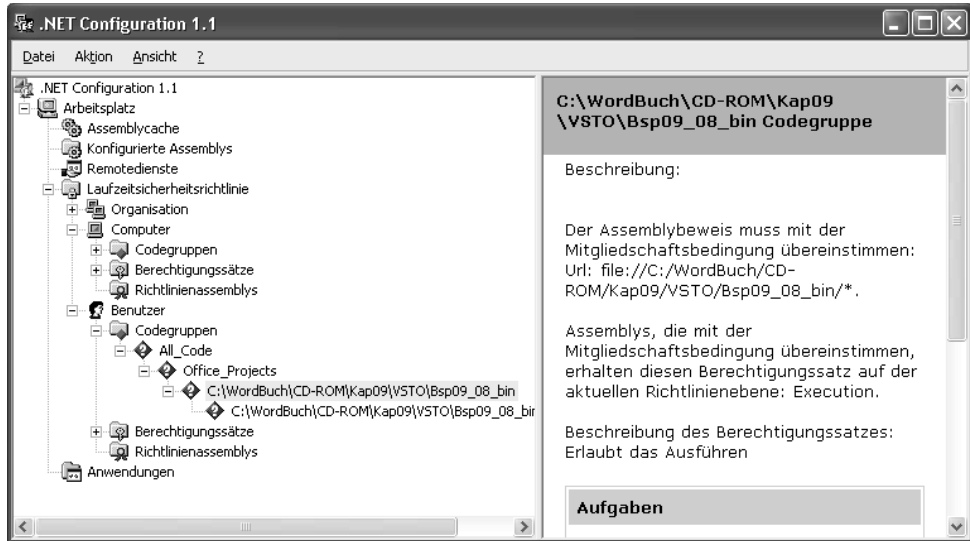
Relative Datenangaben für die VSTO-Lösungsdateien



3. In der Systemsteuerung, unter der Rubrik *Verwaltung*, starten Sie den *Microsoft .NET Framework 1.1-Konfiguration-Assistenten*.
4. Unter *Arbeitsplatz/Laufzeitsicherheitsrichtlinie/Benutzer/Codegruppen* (Abbildung 9.10) wählen Sie den Eintrag *All\_Code*. Wenn Sie schon einmal ein VSTO-Projekt erstellt haben, muss sich ein Eintrag *Office\_Projects* darunter befinden. Wenn nicht, wird er wie folgt erstellt:
  - Den Eintrag *All\_Code* wählen.
  - Im Fenster rechts *Untergeordnete Codegruppe hinzufügen* anklicken, um den Assistenten zu starten.
  - Tragen Sie den Namen *Office\_Projects* im ersten Schritt ein und klicken Sie auf *Weiter*.
  - Im folgenden Schritt auf *Weiter* klicken (den Vorschlag *Gesamter Code* annehmen).
  - Im dritten Schritt *Nothing* als *Berechtigungssatz* festlegen und auf *Weiter* klicken.
  - Im letzten Schritt *Fertig st.* anklicken.

Abbildg. 9.10

Benutzer-spezifische Berechtigungen für ein Assembly erstellen



5. Den neu erstellten Eintrag *Office\_Projects* wählen und einen Eintrag für den Ordner erstellen, wo sich das Assembly (die *.dll*) befindet:
  - Im Fenster rechts *Untergeordnete Codegruppe hinzufügen* anklicken, um den Assistenten zu starten.
  - Einen aussagekräftigen Namen eintragen und auf *Weiter* klicken.
  - Im folgenden Schritt *URL* aus der Liste wählen. In das eingblendete Textfeld *URL* die vollständige Pfadangabe zur *.dll* eingeben, gefolgt von */\**. Auf *Weiter* klicken.
  - Im dritten Schritt *Execution* als *Berechtigungssatz* festlegen und *Weiter* anklicken.
  - Im letzten Schritt *Fertig st.* anklicken.
6. Diesen neuen Eintrag wählen und einen Eintrag für die *.dll* erstellen:
  - Im Fenster rechts *Untergeordnete Codegruppe hinzufügen* anklicken, um den Assistenten zu starten.
  - Einen aussagekräftigen Namen eintragen und auf *Weiter* klicken.
  - Im folgenden Schritt *URL* aus der Liste wählen. In das eingblendete Textfeld *URL* die vollständige Pfadangabe zur *.dll* eingeben (inklusive Dateiname). Auf *Weiter* klicken.
  - Im dritten Schritt *FullTrust* als *Berechtigungssatz* festlegen und auf *Weiter* klicken.
  - Im letzten Schritt *Fertig stellen* anklicken.

# Zusammenfassung

In diesem Kapitel wurde gezeigt, wie Word von außen ferngesteuert werden kann. In einem zweiten Schwerpunkt wurde erläutert, wie die Prozeduren eines Add-Ins angesprochen werden:

- Dieses Kapitel hat Ihnen gezeigt, wie Makros von außen mittels der Run-Methode angestoßen werden (Seite 482) und wie Word effektiv ferngesteuert werden kann (Seite 486).
- Als weiterer Schwerpunkt wurde dargestellt, wie bereits erstellte Programmsequenzen gemeinsam genutzt werden können (Seite 490) und wie diese in einem *.dot*-Add-In zur Verfügung gestellt werden können.
- Es wurde erläutert, was beachtet werden muss, damit benutzerdefinierte Dialogfelder ebenfalls in einem Add-In zentral verwaltet und genutzt werden können (Seite 492).
- Zudem wurde veranschaulicht, wie Prozeduren in globalen Add-Ins von anderen Programmierumgebungen aus aufgerufen werden können (Seite 493).
- Wie mit Visual Studio eine dokumentspezifische Aufgabe mit VSTO erstellt werden kann, wurde ab Seite 495 beschrieben.



## Kapitel 10

# Andere Programme aus Word heraus steuern

### In diesem Kapitel:

Fernsteuern von Microsoft Excel	510
Fernsteuern von Microsoft PowerPoint	513
Fernsteuern von Microsoft Visio	518
Fernsteuern von Microsoft Outlook	520
Fernsteuern von Microsoft Access	528
Auf Datenbanken zugreifen	530
Zusammenfassung	538

Dieses Kapitel widmet sich der Fernsteuerung anderer Applikationen aus Word heraus. Dabei stehen natürlich die Programme von Microsoft Office im Vordergrund. Es werden Beispiele für den Zugriff auf Excel, PowerPoint, Visio, Outlook und Access sowie den Zugriff auf Datenbanken präsentiert.

In Kapitel 9 wurde ein kurzes Szenario erarbeitet, welches hier bei allen Applikationen als Beispiel umgesetzt wird. Anhand dieses Beispiels wird aufgezeigt, dass das Fernsteuern einer Applikation eigentlich sehr einfach ist. Allerdings hat jedes Programm auch seine besonderen Eigenheiten, die es zu beachten gilt. Eines sei im Voraus verraten: Obwohl eine gemeinsame Programmiersprache zur Verfügung steht, können die Programmzeilen selten direkt von einer Applikation in die andere übernommen werden. Denn zu unterschiedlich sind die Objektmodelle der einzelnen Programme. Das dieser Diskussion zu Grunde liegende Beispiel (die erste Seite einer Datei ausdrucken) für Word ist in Kapitel 9 aufgeführt.

Alle Beispiele dieses Kapitels sind so ausgelegt, dass sie für Early wie auch für Late Binding funktionieren. Innerhalb der ersten Zeilen ist jeweils eine entsprechende Compiler-Konstante definiert, über deren Wert die gewünschte Funktionalität gesteuert wird. Das Thema »Early und Late Binding« sowie Compiler-Anweisungen finden Sie eingehend in Kapitel 8 erläutert.

```
#Const EARLYBINDING = False    'oder True
```

**WICHTIG**    Bei Verwendung von Early Binding muss unbedingt der entsprechende Verweis auf die zugehörige Objektbibliothek gesetzt werden, damit die Prozedur fehlerfrei gestartet werden kann (siehe in Kapitel 8 den Abschnitt »Verweise auf externe Bibliotheken«).

**HINWEIS**    Damit die nachfolgenden Beispiele fehlerfrei funktionieren, kopieren Sie die zugehörige Datei in den entsprechenden Ordner oder passen Sie jeweils die Konstante in der ersten Zeile der Prozedur so an, dass der Pfad zur Beispieldatei den tatsächlichen Gegebenheiten entspricht. Hierzu ein Beispiel:

```
Const strDATEI_NAME As String = "C:\WordBuch\Beispiele\Kap10\Bsp_Demo.xls"
```

## Fernsteuern von Microsoft Excel



Dieser Abschnitt zeigt die verschiedenen Möglichkeiten zum Fernsteuern von Microsoft Excel auf, wobei neben dem in der Einleitung zu diesem Kapitel erwähnten Standardszenario auch Beispiele zur Kernkompetenz von Excel aufgegriffen werden.

Unter der Kernkompetenz von Excel ist das Berechnen von einfachen, aber auch komplexen Formeln zu verstehen. Einfache mathematische Funktionen stehen in VBA zur Verfügung oder sind schnell nachgebildet. Komplexe Formeln dagegen stehen nicht zur Verfügung und müssen selbst konstruiert werden. Viel einfacher ist es jedoch, diese Aufgabe dem entsprechenden Experten zu übergeben und die Berechnung mittels Fernsteuerung durch Excel erledigen zu lassen.

**HINWEIS**

Wie Daten in eine Excel-Arbeitsmappe geschrieben werden, ohne die Datei öffnen zu müssen, ist im Abschnitt »Excel-Tabelle ansprechen« in diesem Kapitel beschrieben. Mehr zur Fernsteuerung des Excel-Objektmodells finden Sie in Kapitel 11.

## Versteckte Excel-Instanz steuern

Anhand der Aufgabe aus dem Standardszenario soll aus einer Excel-Arbeitsmappe ein Ausdruck auf dem Standarddrucker erfolgen, und zwar nur die erste Seite des ersten Arbeitsblatts. Da Excel aber eigentlich nicht über Seiten, sondern lediglich über Arbeitsmappen und Arbeitsblätter verfügt, muss diese so genannte erste Seite zunächst definiert werden.

**Listing 10.1** Ausdrucken eines Arbeitsblatts mittels Fernsteuerung von Excel

```
Sub Demo_ExcelFernsteuern()
    Const strDATEI_NAME As String = "C:\WordBuch\Beispiele\Kap10\Bsp_Demo.xls"

    #Const EARLYBINDING = False 'oder True

    System.Cursor = wdCursorWait

    'Variablen und Objekte anlegen
    #If EARLYBINDING Then
        'Wird mit Early Binding gearbeitet, muss ein Verweis auf
        'die "Microsoft Excel 11.0 Object Library" aktiviert werden
        Dim xlsApp As Excel.Application
        Dim xlsWkb As Excel.Workbook
        Dim xlsWks As Excel.Worksheet

        'Neue Excel-Instanz erzeugen
        Set xlsApp = New Excel.Application
    #Else
        Dim xlsApp As Object
        Dim xlsWkb As Object
        Dim xlsWks As Object

        'Neue Excel-Instanz erzeugen
        Set xlsApp = CreateObject("Excel.Application")
    #End If

    'Prüfen ob eine neue Instanz erzeugt werden konnte
    If Not (xlsApp Is Nothing) Then
        With xlsApp
            'Applikation am Bildschirm verbergen
            .Visible = False
            Set xlsWkb = .Workbooks.Open( FileName:=strDATEI_NAME, AddToMru:=False)

            With xlsWkb
                Set xlsWks = xlsWkb.Worksheets(1)
            End With
        End With

        'Arbeitsblatt ausdrucken
        xlsWks.PrintOut From:=1, To:=1, Copies:=1
        .Saved = True
        .Close
    End With
End Sub
```

**Listing 10.1**    Ausdrucken eines Arbeitsblatts mittels Fernsteuerung von Excel (*Fortsetzung*)

```
.Quit  
End With  
Else  
    MsgBox "Neue Instanz von Excel konnte nicht erzeugt werden."  
End If  
  
Set xlsWks = Nothing  
Set xlsWkb = Nothing  
Set xlsApp = Nothing  
System.Cursor = wdCursorNormal  
End Sub
```

Anhand der eingefügten Kommentarzeilen sollte das Listing 10.1 selbsterklärend sein. Trotzdem sollen zwei Programmzeilen detaillierter besprochen werden.

Die `Open`-Methode für eine Arbeitsmappe stellt den optionalen Parameter `AddToMru` zur Verfügung. Wird, wie im Beispiel, `False` als Wert übergeben, so wird die geöffnete Datei nicht in die Liste der zuletzt geöffneten Dateien aufgenommen:

```
Set xlsWkb = .Workbooks.Open(Filename:=strDATEI_NAME, AddToMru:=False)
```

Damit tatsächlich nur die erste Seite und nicht das ganze Arbeitsblatt gedruckt wird, muss für die `PrintOut`-Methode des Arbeitsblatts der entsprechende Seitenbereich als Parameter übergeben werden. In unserem Fall müssen beide Werte auf eins (1) gesetzt werden:

```
xlsWks.PrintOut From:=1, To:=1, Copies:=1
```

## Berechnungen von Excel erledigen lassen

Müssen komplexe mathematische, arithmetische oder andere komplexe Funktionen innerhalb eines Word-Makros berechnet werden, stehen standardmäßig nur wenige Funktionen im Sprachumfang von VBA zur Verfügung.

Zwar könnte man die benötigten Funktionen selbst entwickeln, viel einfacher ist es jedoch, auf bestehende Funktionen anderer Programmbibliotheken zurückzugreifen. In der Bibliothek von Excel (*Microsoft Excel 11.0 Object Library*) werden solche Berechnungsfunktionen zur Verfügung gestellt und können somit auch von einem Word-Makro genutzt werden.

In Listing 10.2 ist eine Möglichkeit zur Berechnung von Kombinationen aufgeführt. (Berechnen der Anzahl möglicher Gruppen, die aus einer bestimmten Anzahl von Elementen gebildet werden können.) Dieses Beispiel baut auf Early Binding auf. So kann direkt auf alle Funktionen von Excel zugegriffen werden, ohne dass eine Arbeitsmappe definiert werden muss.

Soll jedoch Late Binding zum Einsatz kommen, muss auf eine Arbeitsmappe zugegriffen und deren Zellen direkt bearbeitet werden. Die zugehörige Programmsequenz zu Late Binding ist in der Beispieldatei enthalten. Weitere entsprechende Programmbeispiele sind in Kapitel 11 aufgeführt.

**Listing 10.2** Mittels Early Binding wird die mögliche Kombination von Gruppen zu einer Anzahl von Elementen in Excel berechnet.

```
Sub Demo_MitExcelRechnen_Combin()
    'Wird mit Early Binding gearbeitet, muss ein Verweis auf
    'die "Microsoft Excel 11.0 Object Library" aktiviert werden
    Dim xlsApp As Excel.Application
    Dim dblZahl As Double
    Dim dblBasis As Double
    Dim dblResultat As Double

    'Eingabewerte festlegen
    dblZahl = CDbI(InputBox("Anzahl Elemente eingeben", "Kombinationen berechnen", "64"))
    dblBasis = CDbI(InputBox("Anzahl Elemente in jeder Kombination eingeben", _
        "Kombinationen berechnen", "4"))

    'Neue Excel-Instanz erzeugen
    Set xlsApp = New Excel.Application
    With xlsApp
        dblResultat = .WorksheetFunction.Combin(dblZahl, dblBasis)
        .Quit
    End With

    MsgBox "Kombination mit Excel COMBIN-Funktion berechnet" & vbCrLf & _
        "Anzahl Elemente" & vbCrLf & dblZahl & vbCrLf & _
        "Anzahl Elemente in einer Kombination" & vbCrLf & dblBasis & vbCrLf & _
        "Anzahl mögliche Kombinationen" & vbCrLf & dblResultat & vbCrLf

    Set xlsApp = Nothing
End Sub
```



Die oben dargestellten Beispiele finden Sie in der Beispieldatei *Bsp10\_01.doc* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap10*.

## Fernsteuern von Microsoft PowerPoint



In diesem Abschnitt geht es um die verschiedenen Möglichkeiten, das Präsentationsprogramm PowerPoint fernzusteuern. Nebst dem in der Einleitung erwähnten Standardszenario werden auch Beispiele zur Kernkompetenz von PowerPoint aufgegriffen: dem Erstellen von Folien und Bildschirmpräsentationen.

In PowerPoint bereits erfasste Texte werden in einem Beispiel als Ausgangsstruktur für ein neues Dokument verwendet, das später den detaillierten Text zur Präsentation enthalten soll.

### Versteckte PowerPoint-Instanz steuern

Anhand der Aufgabe aus dem Standardszenario soll ein Ausdruck aus einer PowerPoint-Präsentation auf den Standarddrucker ausgegeben werden. Da PowerPoint, im Gegensatz zu anderen Programmen, über eigentliche Seiten verfügt, können diese direkt der PrintOut-Methode übergeben werden.

**Listing 10.3    Ausdrucken einer Folie mittels Fernsteuerung von PowerPoint**

```

Sub Demo_PowerpointFernsteuern()
    Const strDATEI_NAME As String = "C:\WordBuch\Beispiele\Kap10\Bsp_Demo.ppt"

    #Const EARLYBINDING = False    'oder True

    Dim bBackground As Boolean
    Dim bVisible As Boolean

    System.Cursor = wdCursorWait

'Variablen und Objekte anlegen
    #If EARLYBINDING Then
        'Wird mit Early Binding gearbeitet, muss ein Verweis auf
        'die "Microsoft Powerpoint 11.0 Object Library" aktiviert werden
        Dim pptApp As PowerPoint.Application
        Dim pptPrs As PowerPoint.Presentation

        'Neue Powerpoint-Instanz erzeugen
        Set pptApp = New PowerPoint.Application
    #Else
        Dim pptApp As Object
        Dim pptPrs As Object

        'Neue Powerpoint-Instanz erzeugen
        Set pptApp = CreateObject("Powerpoint.Application")
    #End If

'Prüfen ob eine neue Instanz erzeugt werden konnte
    If Not (pptApp Is Nothing) Then
        With pptApp
'Applikation am Bildschirm verbergen
            bVisible = False    'oder True
            If bVisible Then
                .Visible = bVisible
            End If

            Set pptPrs = .Presentations.Open(Filename:=strDATEI_NAME, WithWindow:=bVisible)

            With pptPrs
                bBackground = .PrintOptions.PrintInBackground
                .PrintOptions.PrintInBackground = False
            End With

'Präsentation ausdrucken
            .PrintOut From:=1, To:=1, Copies:=1
            .PrintOptions.PrintInBackground = bBackground
            .Saved = True
            .Close
        End With
        .Quit
    End With
    Else
        MsgBox "Neue Instanz von Powerpoint konnte nicht erzeugt werden."
    End If

    Set pptPrs = Nothing
    Set pptApp = Nothing
    System.Cursor = wdCursorNormal
End Sub

```

Anhand der eingefügten Kommentarzeilen sollte das Listing 10.3 selbsterklärend sein. Auf drei Punkte wird dennoch im Detail eingegangen.

Gemäß dem Standardszenario sollen die gestarteten Instanzen unsichtbar sein, trotzdem sind die Beispiele so aufgebaut, dass die einzelnen Programme zu Testzwecken sichtbar gestartet werden können. Grundsätzlich wird jede neu angelegte Instanz von PowerPoint versteckt gestartet. Ist aber die `Visible`-Eigenschaft auf `True` gesetzt, wird die Applikation sichtbar. Also sollte es an dieser Stelle auch möglich sein, die Eigenschaft auf `False` zu setzen. Ein entsprechender Versuch wird allerdings mit einer Fehlermeldung quittiert: »Hiding the application window is not allowed«. Dieser Eigenschaft kann nur `True`, aber nicht `False` zugewiesen werden, weshalb die `Visible`-Eigenschaft in eine `If...Then`-Anweisung geschachtelt wurde.

Damit die Applikation zu Testzwecken trotzdem sichtbar gestartet werden kann, wurde die Variable `bVisible` definiert. Jetzt kann an einer Stelle festgelegt werden, ob PowerPoint sichtbar oder unsichtbar gestartet wird. Die entsprechenden Abhängigkeiten (siehe `Open`-Methode) sind in der Prozedur bereits berücksichtigt.

```
bVisible = False 'oder True
If bVisible Then
    .Visible = bVisible
End If
```

Die `Open`-Methode für eine Präsentation stellt den optionalen Parameter `WithWindow` zur Verfügung. Dieser Parameter steuert, ob eine Präsentation sichtbar oder unsichtbar geöffnet wird. Wird versucht, eine Präsentation sichtbar zu öffnen, obwohl die Applikation versteckt gestartet wurde, wird dies mit der Fehlermeldung »The PowerPoint frame windows does not exist« quittiert.

```
Set pptPrs = .Presentations.Open(FileName:=strDATEI_NAME, WithWindow:=bVisible)
```

PowerPoint kann auf zwei verschiedene Arten drucken: im Vordergrund (synchron) oder im Hintergrund (asynchron). Wie bereits in Kapitel 6 erläutert, wird beim Einsatz von Makros innerhalb von Word immer das synchrone Drucken im Vordergrund empfohlen. Die gleichen Gründe gelten auch für PowerPoint.

In der Variablen `bBackground` wird der aktuelle Status der Option *Drucken im Hintergrund* gespeichert, damit die veränderte Einstellung nach erfolgreichem Ausdruck zurückgesetzt werden kann:

```
bBackground = .PrintOptions.PrintInBackground
pptPrs.PrintOptions.PrintInBackground = False
pptPrs.PrintOut From:=1, To:=1, Copies:=1
pptPrs.PrintOptions.PrintInBackground = bBackground
```

## Text der Präsentation als Inhaltsstruktur in ein Dokument übernehmen

Muss zu einer bereits vorliegenden Präsentation ein zusätzlicher Bericht erstellt werden, ist es wünschenswert, dass Struktur und Inhalt des neuen Dokuments mit der Präsentation übereinstimmen. Aus diesem Grunde ist es sinnvoll, die bereits erfassten Texte der Präsentation in das Word-Dokument zu übertragen.

**Abbildg. 10.1**    Der Quelltext auf der Folie wird in das Dokument übernommen.


Das Listing 10.4 liest alle Texte aus den Platzhaltern ein und überträgt sie unformatiert in das Dokument. Das Beispiel ist bewusst einfach aufgebaut. Selbstverständlich wäre es möglich, den einzelnen Absätzen zusätzlich eine bestimmte Formatvorlage zuzuweisen oder die Programmzeilen so zu erweitern, dass nebst den Texten aus Platzhaltern auch Texte aus Textfeldern, Kommentaren oder Notizenseiten übertragen werden.

**Listing 10.4**    Einlesen der Texte aus den Platzhaltern und übertragen in ein Dokument

```
Sub Demo_PowerpointStruktur_Übertragen()
    Const strDATEI_NAME As String = "C:\WordBuch\Beispiele\Kap10\Bsp_Demo.ppt"

    #Const EARLYBINDING = False 'oder True

    'Variablen und Objekte anlegen
    #If EARLYBINDING Then
        'Wird mit Early Binding gearbeitet, muss ein Verweis auf
        'die "Microsoft PowerPoint 11.0 Object Library" aktiviert werden
        Dim pptApp As PowerPoint.Application
        Dim pptPrs As PowerPoint.Presentation
        Dim pptSld As PowerPoint.Slide
        Dim pptShp As PowerPoint.Shape
        Dim pptPara As PowerPoint.TextRange

        'Neue Powerpoint-Instanz erzeugen
        Set pptApp = New PowerPoint.Application
    #Else
        Dim pptApp As Object
        Dim pptPrs As Object
        Dim pptSld As Object
        Dim pptShp As Object
        Dim pptPara As Object

        'Neue Powerpoint-Instanz erzeugen
        Set pptApp = CreateObject("Powerpoint.Application")
    #End If

    Dim doc As Word.Document

    Set doc = Documents.Add
```



Listing 10.4 Einlesen der Texte aus den Platzhaltern und übertragen in ein Dokument (Fortsetzung)

```

'Prüfen ob eine neue Instanz erzeugt werden konnte
If Not (pptApp Is Nothing) Then
    With pptApp

        Set pptPrs = .Presentations.Open( _
            FileName:=strDATEI_NAME, _
            WithWindow:=False)

'Präsentation einlesen, auf Dokument übertragen
    With pptPrs
        For Each pptSld In pptPrs.Slides
            For Each pptShp In pptSld.Shapes.Placeholders
                If pptShp.HasTextFrame Then
                    Set pptPara = pptShp.TextFrame.TextRange
                    doc.Range.InsertAfter pptPara.Text & vbCrLf
                End If
            Next pptShp
        Next pptSld
        .Saved = True
        .Close
    End With
    .Quit
End With
Else
    MsgBox "Neue Instanz von PowerPoint konnte nicht erzeugt werden."
End If

Set doc = Nothing
Set pptPara = Nothing
Set pptShp = Nothing
Set pptSld = Nothing
Set pptPrs = Nothing
Set pptApp = Nothing
End Sub

```

Die Funktionsweise der Prozedur ist schnell erklärt, denn neben dem eigentlichen, bereits bekannten, Verbindungsaufbau zu PowerPoint bleibt nur noch eine verschachtelte Schleife übrig, die es näher zu betrachten gilt.

In einer ersten For...Each-Schleife wird die Slides-Auflistung der entsprechenden Präsentation durchlaufen:

```
For Each pptSld In pptPrs.Slides
```

Mit der zweiten For...Each-Schleife werden alle Platzhalter der entsprechenden Folie bearbeitet. Würde die Schleife die Shapes-Auflistung durchlaufen, so würden die Textfelder ebenfalls berücksichtigt:

```
For Each pptShp In pptSld.Shapes.Placeholders
```

Die Prüfung, ob das Placeholder-Objekt einen Text beinhaltet, stellt sicher, dass kein Platzhalter bearbeitet wird, der gar keinen Text enthalten kann (Diagramm, Organigramm usw.):

```
If pptShp.HasTextFrame Then
```

Der Text des Platzhalters wird eingelesen und am Ende des Word-Dokuments unformatiert eingefügt:

```
Set pptPara = pptShp.TextFrame.TextRange
doc.Range.InsertAfter pptPara.Text & vbCrLf
```



Die obigen Beispiele finden Sie in der Beispieldatei *Bsp10\_01.doc* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap10*.

## Fernsteuern von Microsoft Visio



Auch Microsoft Visio, ein weit verbreitetes Programm, um technische Zeichnungen unterschiedlichster Art zu erstellen, kann von Word aus ferngesteuert werden.

### Versteckte Visio-Instanz steuern

Anhand der Aufgabe aus dem Standardszenario soll aus einer Visio-Zeichnung ein Ausdruck auf den Standarddrucker erfolgen. Wie PowerPoint verfügt Visio über eigentliche Seiten, die direkt der Print-Methode übergeben werden können. Doch leider ist das nicht ganz so einfach, denn die Print-Methode unterstützt keine Parameter.

Die so genannte erste Seite muss vorab definiert werden, um im Programmbeispiel tatsächlich nur das erste Zeichnungsblatt auf den Drucker auszugeben.

**Listing 10.5**    Ausdrucken eines Zeichnungsblatts mittels Fernsteuerung von Visio

```
Sub Demo_VisioFernsteuern()
    Const strDATEI_NAME As String = "C:\WordBuch\Beispiele\Kap10\Bsp_Demo.vsd"

    #Const EARLYBINDING = False    'oder True

    System.Cursor = wdCursorWait

    'Variablen und Objekte anlegen
    #If EARLYBINDING Then
        'Wird mit Early Binding gearbeitet, muss ein Verweis auf
        'die "Microsoft Visio 11.0 Type Library" aktiviert werden.
        Dim vsdApp As Visio.Application
        Dim vsdDoc As Visio.Document
        Dim vsdPag As Visio.Page

        'Neue Visio-Instanz erzeugen
        Set vsdApp = New Visio.Application
```

Listing 10.5 Ausdrucken eines Zeichnungsblatts mittels Fernsteuerung von Visio (Fortsetzung)

```

#Else
    Dim vsdApp As Object
    Dim vsdDoc As Object
    Dim vsdPag As Object

    'Neue Visio-Instanz erzeugen
    'Set vsdApp = CreateObject("Visio.Application")
    Set vsdApp = CreateObject("Visio.InvisibleApp")
#End If

'Prüfen ob eine neue Instanz erzeugt werden konnte
If Not (vsdApp Is Nothing) Then
    With vsdApp

        'Applikation am Bildschirm verbergen
        .Visible = False
        Set vsdDoc = .Documents.Open(strDATEI_NAME)

        With vsdDoc
            Set vsdPag = .Pages(1)
            With vsdPag

                'Seite ausdrucken
                .Print
            End With
            .Saved = True
            .Close
        End With
        .Quit
    End With
Else
    MsgBox "Neue Instanz von Visio konnte nicht erzeugt werden."
End If

Set vsdPag = Nothing
Set vsdDoc = Nothing
Set vsdApp = Nothing
System.Cursor = wdCursorNormal
End Sub

```

Anhand der eingefügten Kommentarzeilen sollte das Listing 10.5 selbsterklärend sein. Trotzdem soll eine Programmzeile detaillierter besprochen werden.

Visio kennt ein eigenes Objekt, um die Applikation versteckt zu starten. Dieses Objekt unterstützt die `Visible`-Eigenschaft, wodurch es möglich wäre, die Instanz sichtbar am Bildschirm darzustellen:

```

'Set vsdApp = CreateObject("Visio.Application")
Set vsdApp = CreateObject("Visio.InvisibleApp")

```

Wird die neue Instanz von Visio als Objekt der Klasse `Visio.Application` gestartet, so bedeutet dies einen großen Nachteil. Der Standardwert für die `Visible`-Eigenschaft ist bei Visio auf `True` gesetzt. Das Programm ist also für kurze Zeit am Bildschirm sichtbar, auch wenn die `Visible`-Eigenschaft auf `False` gesetzt wird.



Das obige Beispiel finden Sie in der Beispieldatei *Bsp10\_01.doc* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap10*.

## Fernsteuern von Microsoft Outlook



In diesem Abschnitt werden die verschiedenen Möglichkeiten zum Fernsteuern von Microsoft Outlook vorgestellt. Nebst dem in der Einleitung erwähnten Standardszenario werden Beispiele zur Kernkompetenz von Outlook aufgegriffen.

Zur Kernkompetenz von Outlook gehören unter anderem das Verwalten von Kontakten und das Versenden von E-Mails. Die Kontakte sollen in einem Beispiel als Basis für die Empfängeradresse in einem Brief verwendet werden. In einem weiteren Beispiel wird aufgezeigt, wie das aktuelle Dokument als E-Mail versendet werden kann.

### Versteckte Outlook-Instanz steuern

Anhand der Aufgabe aus dem Standardszenario soll ein Ausdruck aus Outlook erstellt werden. Da Outlook eigentlich über keine eigentlichen Seiten verfügt, wird stellvertretend der zuerst erfasste Kalendereintrag ausgedruckt.

**Listing 10.6**    Ausdrucken eines Kalendereintrags mittels Fernsteuerung von Outlook

```
Sub Demo_OutlookFernsteuern()
    #Const EARLYBINDING = False    'oder True

    #If EARLYBINDING Then
        'Wird mit Early Binding gearbeitet, muss der Verweis auf
        'die "Microsoft Outlook 11.0 Object Library" aktiviert werden.
        Dim olApp As Outlook.Application
        Dim olOrdner As Outlook.MAPIFolder
        Dim olKalenderEintrag As Object

        'Neue Outlook-Instanz erzeugen
        Set olApp = New Outlook.Application
    #Else
        Const olFolderCalendar As Integer = 9
        Dim olApp As Object
        Dim olOrdner As Object
        Dim olKalenderEintrag As Object

        'Neue Outlook-Instanz erzeugen
        Set olApp = CreateObject("Outlook.Application")
    #End If

    'Prüfen ob eine neue Instanz erzeugt werden konnte
    If Not (olApp Is Nothing) Then
        With olApp

            'Applikation am Bildschirm verbergen
            .Visible = False    'Eigenschaft nicht vorhanden
```

**Listing 10.6** Ausdrucken eines Kalendereintrags mittels Fernsteuerung von Outlook (*Fortsetzung*)

```

'Kalender-Ordner setzen
    Set o1Ordner = o1App.Session.GetDefaultFolder(o1FolderCalendar)
    Set o1KalenderEintrag = o1Ordner.Items(1)

'Kalendereintrag ausdrucken
    o1KalenderEintrag.PrintOut
    .Quit
    End With
Else
    MsgBox "Neue Instanz von Outlook konnte nicht erzeugt werden."
End If

Set o1KalenderEintrag = Nothing
Set o1Ordner = Nothing
Set o1App = Nothing
End Sub

```

Anhand der eingefügten Kommentarzeilen sollte das Listing 10.6 selbsterklärend sein. Trotzdem sollen zwei Programmzeilen detaillierter besprochen werden.

Das Objektmodell von Outlook verfügt über keine `Visible`-Eigenschaft. Wird eine neue Instanz von Outlook erzeugt, so kann diese nur als versteckte Instanz ferngesteuert werden. Anders verhält es sich, wenn eine bereits aktive Instanz gesteuert wird. Hier kann die `Visible`-Eigenschaft umgeschaltet werden.

```

'.Visible = False 'Eigenschaft nicht vorhanden

```

Wird der Kalender nicht explizit sortiert, entspricht der erste Eintrag im Kalender nicht unbedingt dem Eintrag mit dem kleinsten Datum. Hier handelt es sich um den zuerst erfassten Kalendereintrag.

```

Set o1KalenderEintrag = o1Ordner.Items(1)

```



Das obige Beispiel finden Sie in der Beispieldatei *Bsp10\_01.doc* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap10*.

## Kontakt als Empfängeradresse im Brief nutzen

In diesem Beispiel wird eine Briefvorlage so aufgebaut, dass die bestehenden Adressen aus dem *Kontakte*-Ordner von Outlook als Empfängeradressen zur Verfügung gestellt werden. Ein analoges Beispiel unter Verwendung einer Access-Datenbank wird im Abschnitt »Access-Datenbank ansprechen« in diesem Kapitel aufgezeigt.

Wie in Abbildung 10.2 ersichtlich, können zur gewählten Adresse zusätzliche Eigenschaften des neuen Dokuments im gleichen Dialogfeld erfasst und übertragen werden.

**Abbildg. 10.2**    Eigenschaften des neuen Dokuments bestimmen

Die Dokumentvorlage *Bsp10\_02a.dot* enthält eine Prozedur mit dem Namen *AutoNew*. Dieses Makro stellt sicher, dass das Dialogfeld *Brief* automatisch beim Anlegen eines neuen Dokuments geöffnet wird. Mehr zum Thema »Auto-Makros« ist in Kapitel 7 beschrieben.

Der Zugriff auf Outlook ist in vier eigenständige Prozeduren unterteilt. Diese werden anschließend kurz besprochen.

**Listing 10.7**    Deklaration und Funktion zum Starten von Outlook

```
Option Explicit
#Const EARLYBINDING = False 'oder True

#If EARLYBINDING Then
'Wird mit Early Binding gearbeitet, muss der Verweis auf
'die "Microsoft Outlook 11.0 Object Library" aktiviert werden.
Dim olAnw As Outlook.Application
Dim olOrdner As Outlook.MAPIFolder
Dim olKontakt As Object
Dim olItem As Outlook.Items
Dim olItemFilter As Outlook.Items
#Else
Const olFolderContacts As Integer = 10
Const olContact As Integer = 40
Dim olAnw As Object
Dim olOrdner As Object
Dim olKontakt As Object
Dim olItem As Object
Dim olItemFilter As Object
#End If

Dim bOutlookNeuGestartet As Boolean

Public Function funcOL_Starten()
'Aktive Outlook-Instanz ermitteln ...
```

Listing 10.7 Deklaration und Funktion zum Starten von Outlook (Fortsetzung)

```

On Error Resume Next
Set olAnw = GetObject(, "Outlook.Application")
On Error Resume Next

'... oder eine neue Instanz erzeugen
If (olAnw Is Nothing) Then
    bOutlookNeuGestartet = True
    #If EARLYBINDING Then
        Set olAnw = New Outlook.Application
    #Else
        Set olAnw = CreateObject("Outlook.Application")
    #End If
End If

If (olAnw Is Nothing) Then
    MsgBox "Outlook konnte nicht gestartet werden"
End If
End Function

```

Wie in anderen Beispielen bereits aufgezeigt, sind die Programmzeilen so aufgebaut, dass Early und Late Binding unterstützt werden. Die Deklaration der Objektvariablen wurde auf Modulebene vorgenommen, damit diese in allen Prozeduren dieses Moduls zur Verfügung stehen.

Wie in Listing 10.7 gezeigt, versucht die Funktion *funcOL\_Starten* eine Objektvariable auf eine bereits laufende Instanz von Outlook zu setzen. Dies ist erfolgreich, sofern die Applikation bereits gestartet wurde. Ansonsten wird eine neue Instanz erzeugt. Anhand der Variable *bOutlookNeuGestartet* wird dieser Status festgehalten.

Listing 10.8 Funktion zum Beenden von Outlook

```

Public Function funcOL_Beenden()
    If bOutlookNeuGestartet Then
        olAnw.Quit
    End If

    Set olItemFilter = Nothing
    Set olItem = Nothing
    Set olKontakt = Nothing
    Set olOrdner = Nothing
    Set olAnw = Nothing
End Function

```

Die Funktion *funcOL\_Beenden* dient dem sauberen Beenden von Outlook. In Listing 10.8 wird der Status der Variable *bOutlookNeuGestartet* ausgewertet. Wurde eine neue Instanz von Outlook angelegt, wird diese Instanz wieder beendet. In einem zweiten Schritt werden alle Objektvariablen wieder freigegeben.

**HINWEIS**

Die Freigabe der Objektvariablen sollte immer umgekehrt zu der Reihenfolge geschehen, in der sie erzeugt wurden. Allgemeines zum Thema »Objektvariablen« ist in Kapitel 5 zusammengefasst.

**Listing 10.9**    Funktion zum Übertragen aller Kontakte in das Dialogfeld

```
Function funcOL_KontakteEinlesen()
    If Not (oAnw Is Nothing) Then

        System.Cursor = wdCursorWait

        'Kontakte-Ordner setzen und sortieren
        Set oOrdner = oAnw.Session.GetDefaultFolder(oFolderContacts)
        Set oItem = oOrdner.Items
        oItem.Sort "[FileAs]"

        'Alle Kontakte einlesen (nur das Feld »Speichern unter«)
        For Each oKontakt In oItem
            With oKontakt
                If oKontakt.Class = olContact Then 'nur Kontakte
                    If Not Len(Trim$(.FileAs)) = 0 Then
                        frmBrief.1stKontakteKürzel.AddItem .FileAs
                    End If
                Else
                    'Bei allen anderen nichts machen
                End If
            End With
        Next oKontakt
    End If
    System.Cursor = wdCursorNormal
End Function
```

Die Funktion *funcOL\_KontakteEinlesen* dient zum Einlesen aller Kontakte und zum Eintragen derselben in das Listenfeld *Kontakte in Outlook* (Abbildung 10.2). In einem ersten Schritt werden nur die Kurzbezeichnungen eingelesen. Dies sind, wie in Abbildung 10.3 ersichtlich, die Daten aus dem Feld *Speichern unter*.

Eigentlich wäre es einfacher, in der gleichen Schleife bereits alle benötigten Felder zu jedem Datensatz in eine Tabelle (Array) einzulesen. Aus Gründen der Verarbeitungsgeschwindigkeit wird das aber bewusst unterlassen.

**Abbildg. 10.3**    Die Kurzbezeichnungen des Kontakts werden im Feld *Speichern unter* abgelegt.




Listing 10.10 Funktion zum Übertragen der Detaildaten in das Dialogfeld

```

Public Function funcOL_KontaktEintragen( _
    ByVal intNummer As Integer)

    Dim strItemFilter As String

    'Details zur gewählten Adresse in OL nachlesen
    If Not intNummer = -1 Then
        'Filter setzen
        strItemFilter = "[FileAs] = " & Chr$(34) & frmBrief.lstKontakteKürzel.Text & Chr$(34)
        Set olItemFilter = olItem.Restrict(strItemFilter)

        'Details zum Kontakt einlesen
        Set olKontakt = olItemFilter(1)
        With olKontakt
            frmBrief.txtKürzelOL.Text = .FileAs
            frmBrief.txtFirmaOL.Text = .CompanyName
            frmBrief.txtAnredeOL.Text = .Title
            frmBrief.txtVornameOL.Text = .FirstName
            frmBrief.txtNachnameOL.Text = .LastName
            frmBrief.txtStrasseOL.Text = Replace(.BusinessAddressStreet, vbCrLf, ";")
            frmBrief.txtPlzOL.Text = .BusinessAddressPostalCode
            frmBrief.txtOrtOL.Text = .BusinessAddressCity
            frmBrief.txtTelefonOL.Text = .BusinessTelephoneNumber
            frmBrief.txtTelefaxOL.Text = .BusinessFaxNumber
        End With
    End If
End Function

```

Mit der Funktion *funcOL\_KontaktEintragen* werden die Detaildaten (und zwar nur die geschäftlichen Einträge) eines einzelnen Datensatzes aus den Kontakten gelesen und in das Dialogfeld übertragen. Das Einlesen des Datensatzes erfolgt beim Click-Ereignis des Listenfelds.

Um den gewünschten Datensatz zu finden, wird ein Filter auf den *Kontakte*-Ordner gelegt. Als Filterkriterium wird der bereits eingelesene Wert aus dem Feld *Speichern unter* verwendet:

```

strItemFilter = "[FileAs] = " & Chr$(34) & frmBrief.lstKontakteKürzel.Text & Chr$(34)
Set olItemFilter = olItem.Restrict(strItemFilter)

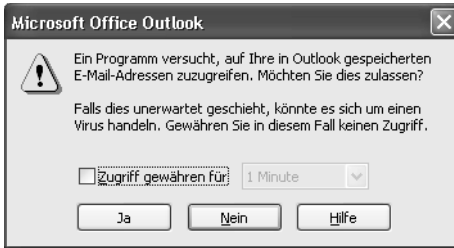
```

**HINWEIS**

Dieses Programmbeispiel greift bewusst nur auf die Geschäftsadressen der eingetragenen Kontakte zu. Selbstverständlich stehen die anderen Adresstypen im Objektmodell von Outlook ebenfalls zur Verfügung und können bei Bedarf berücksichtigt werden.

Die Einträge im Feld *Speichern unter* müssen eigentlich nicht eindeutig sein. Damit das Beispiel übersichtlich und einfach bleibt, wurde darauf verzichtet, diesem Umstand speziell Rechnung zu tragen. Es wird grundsätzlich der erste Eintrag (*olItemFilter(1)*) aus gefilterten Datensätzen eingelesen.

Im aktuellen Beispiel taucht die Sicherheitswarnung (vgl. Abbildung 10.4) von Outlook nicht auf. Sie wird nur eingeblendet, wenn ein anderes Programm auf die E-Mail-Adressen von Outlook zugreifen möchte.

**Abbildg. 10.4**    Sicherheitswarnung von Outlook


Die restlichen Funktionen und Prozeduren, die für ein einwandfreies Funktionieren dieses Beispiels benötigt werden, sind von allgemeiner Bedeutung und haben keinen direkten Zusammenhang zu Outlook. Die Programmzeilen können direkt in der entsprechenden Beispieldatei eingesehen werden.



Das dargestellte Beispiel finden Sie in der Beispieldatei *Bsp10\_02a.dot* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap10*.

## Die aktuelle Datei über Outlook versenden

Eine weitere oft benötigte Aufgabe im Zusammenspiel zwischen Word und Outlook liegt im Versenden des aktuellen Word-Dokuments über Outlook. Wie die grundlegende Verbindung von Word nach Outlook dabei hergestellt wird, wurde bereits im Abschnitt »Versteckte Outlook-Instanz steuern« in diesem Kapitel beschrieben.

Sobald die Verbindung zu dem Outlook-Objekt oder der Outlook-Instanz besteht, wird ein neues MailItem-Objekt erstellt, das eine neue Mail darstellt:

```
Set olEmailItem = olApp.CreateItem(olMailItem)
```

Ist das neue MailItem-Objekt erstellt, wird es, wie in Listing 10.11 beschrieben, mit allen notwendigen Angaben wie Empfänger, Betreff und Mitteilungstext ausgefüllt. Zum Schluss wird die aktuelle Word-Datei als Anhang der Mail zugewiesen.

**Listing 10.11**    Das MailItem-Objekt mit Daten füllen

```
With olEmailItem
    On Error GoTo err_Sub
    ' Empfänger festlegen
    .To = "Christian Freßdorf <Christian.Fressdorf@127.0.0.1>"
    ' Weiteren Empfänger im Feld CC: eintragen
    Set olRecipients = .Recipients.Add("Thomas Gahler <Thomas.Gahler@127.0.0.1>")
    olRecipients.Type = olCC
    ' Betreff festlegen
    .Subject = "das überarbeitete Dokument"
    strMSG = "Hallo Christian, Hallo Thomas," & vbCrLf & _
        "im Anhang erhalten Ihr das geänderte Dokument. Bitte prüfen!" & vbCrLf & "MfG"
```

Listing 10.11 Das MailItem-Objekt mit Daten füllen (Fortsetzung)

```

' Wichtigkeit festlegen
.Importance = olImportanceHigh
' Textkörper um Namen des Anhangs ergänzen
.Body = strMSG & vbCrLf & "<<< " & strAttachment & " >>>"
' Anhang anfügen
.Attachments.Add strAttachment
' Empfangsbestätigung anfordern
.ReadReceiptRequested = True
' Mail im Postausgang speichern
.Save
' Mail versenden
.Send
MsgBox "Das Dokument wurde versendet.", vbInformation, c_Title
err_Sub:
If Err.Number > 0 Then
    MsgBox "Es ist ein Fehler beim Zugriff auf Outlook aufgetreten.", vbCritical, c_Title
    GoTo end_Sub
End If
End With

```

Da nur eine gespeicherte Datei als Anhang hinzugefügt werden kann, wird ausführlich geprüft, ob die Datei überhaupt schon gespeichert wurde (was bei neu angelegten Dateien nicht der Fall ist). In diesem Fall wird das Dialogfeld `Dialogs(wdDialogFileSaveAs)` zum Speichern der Datei angezeigt. Nur wenn der Dialog über *Speichern* verlassen wird und das Dokument anschließend gespeichert ist (`ActiveDocument.Saved = True`), wird die Datei als Anhang an die Mail angehängt. Andernfalls wird der Dateiversand abgebrochen.

Auch wenn die Datei nach dem letzten Speichern noch geändert und anschließend nicht wieder gespeichert wurde, erfolgt eine entsprechende Abfrage.

Listing 10.12 Prüfen, ob das aktuelle Dokument gespeichert ist

```

'Prüfen ob aktuelles Dokument gespeichert ist und ggf. speichern
Dim bSaved As Boolean: bSaved = True
Do While ActiveDocument.Saved = False Or ActiveDocument.Path = ""
    If ActiveDocument.Path = "" Then ' Neues Dokument, noch nicht gespeichert
        bSaved = False ' Nicht gespeichert
        MsgBox "Das Dokument wurde noch nicht gespeichert." & vbCrLf & _
            "Bitte erst speichern!", vbInformation, c_Title
        With Dialogs(wdDialogFileSaveAs) ' Speichern-Dialog aufrufen
            intRet = .Show
        End With
        If intRet = -1 And ActiveDocument.Saved = True Then ' Über 'Speichern' gespeichert
            bSaved = ActiveDocument.Saved
        ElseIf intRet = 0 Or ActiveDocument.Path = "" Then ' Abbruch gewählt
            MsgBox "Das Dokument wurde nicht gespeichert, das Makro wird beendet.", _
                vbCritical, c_Title
            GoTo end_Sub
        End If
    ElseIf ActiveDocument.Saved = False Then
        intRet = MsgBox("Das aktuelle Dokument muss gespeichert werden" & vbCrLf & _
            "Jetzt speichern?", vbInformation + vbYesNo, c_Title)
        If intRet = vbYes Then
            ActiveDocument.Save

```

**Listing 10.12**    Prüfen, ob das aktuelle Dokument gespeichert ist

```

ElseIf intRet = vbNo Then
    MsgBox "Das Dokument wurde nicht gespeichert, das Makro wird beendet.", _
        vbCritical, c_Title
    GoTo end_Sub
End If
End If
Loop
strAttachment = ActiveDocument.FullName

```



Das obige Beispiel finden Sie in der Beispieldatei *Bsp10\_02b.doc* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap10*.

## Fernsteuern von Microsoft Access



Bei der Integration zwischen Word und Access gibt es zwei Betrachtungsaspekte: die Manipulation der Benutzerschnittstelle und den reinen Zugriff auf die Daten. Ein Zugriff auf die Benutzerschnittstelle ist nur begrenzt vorgesehen. Das Anzeigen eines Formulars oder eines Berichts stellt allerdings kein großes Problem dar. Dem Zugriff auf die Daten ist ein eigener Abschnitt »Auf Datenbanken zugreifen« in diesem Kapitel gewidmet.

Wie bei der Automatisierung von allen Office-Anwendungen ist der Einstiegspunkt das Application-Objekt. Ist die gewünschte Datenbank geöffnet, können Formulare oder Berichte mit der Eigenschaft DoCmd geöffnet und, wie in der Access-Schnittstelle, weiter manipuliert werden. In der Datenbank vorhandene Makros und Code-Module werden mit den RunCommand- bzw. Run-Methoden ausgeführt. Mehr Informationen stehen in der *Access Language Reference* (Hilfedatei) bereit.

Das kurze Beispiel in Listing 10.13 zeigt, wie eine neue Instanz von Access gestartet und eine Datenbank geöffnet wird. Ein vorhandenes Formular innerhalb der Datenbank wird geöffnet und die Kunden aus Deutschland (WhereCondition) werden aufgelistet. Da der DataMode auf acReadOnly gesetzt wurde, dürfen die Daten nur gelesen, aber nicht geändert werden.

**Listing 10.13**    Die Access-Anwendung öffnen und ein Formular anzeigen

```

Sub AccessFormularEinblenden()
    Dim accApp As Access.Application
    Dim strDB As String

    strDB = "C:\WordBuch\Datenbank\Nordwind.mdb"
    Set accApp = New Access.Application
    accApp.AutomationSecurity = msoAutomationSecurityLow
    accApp.OpenCurrentDatabase filepath:=strDB, Exclusive:=False, _
        bstrPassword:=""
    accApp.DoCmd.OpenForm FormName:="Kunden", View:=acNormal, FilterName:="", _
        WhereCondition:="Land='Deutschland'", DataMode:=acFormReadOnly, _
        WindowMode:=acDialog, OpenArgs:=""
    Set accApp = Nothing
End Sub

```

Die `OpenReport`-Methode wird verwendet, um einen Bericht zu öffnen. Die Ansicht (View) `acViewNormal` druckt den Bericht sofort; `acViewPreview` zeigt ihn zuerst dem Benutzer, der entscheidet, ob und wie er auszudrucken ist. In Listing 10.14 wird der Bericht direkt gedruckt und die Access-Instanz anschließend wieder geschlossen.

**Listing 10.14** Einen Access-Bericht ausdrucken

```
Sub AccessBerichtDrucken()
    Dim accApp As Access.Application
    Dim strDB As String

    strDB = "C:\WordBuch\Datenbank\Nordwind.mdb"
    Set accApp = New Access.Application
    On Error GoTo Fehlerbehandlung
    accApp.AutomationSecurity = msoAutomationSecurityLow
    accApp.OpenCurrentDatabase filepath:=strDB, Exclusive:=False, _
        bstrPassword:=""
    accApp.DoCmd.OpenReport ReportName:="Katalog", View:=acViewNormal, _
        FilterName:="", WhereCondition:=""
    accApp.CloseCurrentDatabase

Aussteigen:
    Set accApp = Nothing
    Exit Sub

Fehlerbehandlung:
    Select Case Err.Number
        Case 7866
            MsgBox "Die Datenbank ist im exklusiven Modus geöffnet, " & _
                "oder ein Datenbankelement ist im Entwurfsmodus geöffnet." & _
                "Der Bericht kann erst gedruckt werden, wenn die Datenbank " & _
                "freigegeben wurde", vbInformation + vbOKOnly
            Resume Aussteigen
        Case Else
            MsgBox "Unerwartete Fehlernummer " & Err.Number & vbCr & vbCr _
                & Err.Description, vbCritical + vbOKOnly
            Resume Aussteigen
    End Select
End Sub
```



Die Beispieldatei *Bsp10\_03.doc* finden Sie auf der CD-ROM zum Buch im Ordner `\Beispiele\Kap10`. Die Beispieldatenbank *Nordwind.mdb* befindet sich im Ordner `\Datenbank`.

#### HINWEIS

Access implementiert sowohl die Makrosicherheit als auch eine zusätzliche Datenbanksicherheit. Um eventuelle Meldungen während der Automatisierung zu unterdrücken, stellen Office 2002 und 2003 die Application-Eigenschaft `AutomationSecurity` zur Verfügung.

Ist die Datenbank im exklusiven Modus geöffnet oder hat irgendein Benutzer ein Datenbank-Element im Entwurfsmodus geöffnet, kann die Datenbank nicht automatisiert werden. Das Listing 10.14 zeigt, wie dieser Fehler abgefangen werden kann.

# Auf Datenbanken zugreifen



In diesem letzten Abschnitt wird anhand einer Access-Datenbank sowie einer Excel-Tabelle der direkte Zugriff auf die Daten einer Datenbank erläutert, wobei die Zugriffe auf die Excel-Tabelle ebenfalls mittels Datenbankfunktionalitäten ausgeführt werden.

## Access-Datenbank ansprechen

In den meisten Fällen muss nicht Access selbst ferngesteuert werden, sondern es muss lediglich ein Zugriff auf die Daten der Datenbank erfolgen. So können diese Daten als Tabelle in Word erscheinen, in definierte Zielbereiche eines Dokuments geschrieben werden, zur Auswahl innerhalb einer Liste stehen oder gar von Word aus aktualisiert und ergänzt werden. Ein Zugriff auf die Datenbank kann sogar dann erfolgen, wenn auf der Arbeitsstation Access gar nicht installiert ist.

Datenbanken für den Desktop-Rechner gibt es seit den frühesten Tagen des PC. Jede dieser Anwendungen speichert die Daten auf eine eigene Art und Weise, was den Datenaustausch zwischen verschiedenen Anwendungen erschwert. Aus diesem Grund wurden schon früh Schnittstellen für den Datenaustausch und -zugriff entwickelt. Am Anfang stand die zeichengetrennte Textdatei. Zu Beginn der neunziger Jahre wurde ODBC (Open Database Connectivity) eingeführt. Ende des letzten Jahrhunderts wurde ADO (ActiveX Data Objects) entwickelt, und seit kurzem werden XML und ADO.NET – Teil des .NET Frameworks – als die ultimative Lösung angepriesen. Zudem hat Access eine eigene Zugriffsmethode: DAO (Data Access Objects).

Alle diese Schnittstellen erlauben den direkten Zugriff auf die Daten, ohne dass die Anwendung auf der Arbeitsstation aktiv ist bzw. installiert sein muss.

Access bietet für alle diese Methoden eine Schnittstelle an. Da sich dieses Buch mit Word und nicht Access befasst, werden wir uns auf ein kurzes Beispiel mit ADO beschränken.

Feldfunktion  
Database



Word bietet die Möglichkeit, eine Tabelle mit dem Inhalt einer Datenbanktabelle oder -abfrage in ein Word-Dokument einzufügen. Diese kann statisch oder mit einer dynamischen Verknüpfung zur Datenquelle ausgestattet sein. In der Benutzerschnittstelle wird die Tabelle über die Schaltfläche *Datenbank einfügen* aus der Symbolleiste *Datenbank* eingefügt.

### HINWEIS

Wir empfehlen diesen Weg auch dem Entwickler, um die Syntax für die *Database-Feldfunktion* zu ermitteln. Das Word-Objektmodell stellt zwar eine *InsertDatabase-Methode* zur Verfügung, diese ist jedoch schwer zu bedienen. Mehr Informationen zu den Feldfunktionen sind in Kapitel 6 zusammengefasst.

Das Einfügen einer Datenbanktabelle benötigt vier Schritte. Diese sind in Abbildung 10.5 zusammengestellt. Im ersten Schritt wird die Verbindung zur Datenbank hergestellt, meistens stehen für Access drei Verbindungsmethoden zur Verfügung: DDE (Dynamic Data Exchange), ODBC (außer für Word 2000) sowie OLE DB. Wir empfehlen die Verwendung von ODBC aus zwei Gründen:

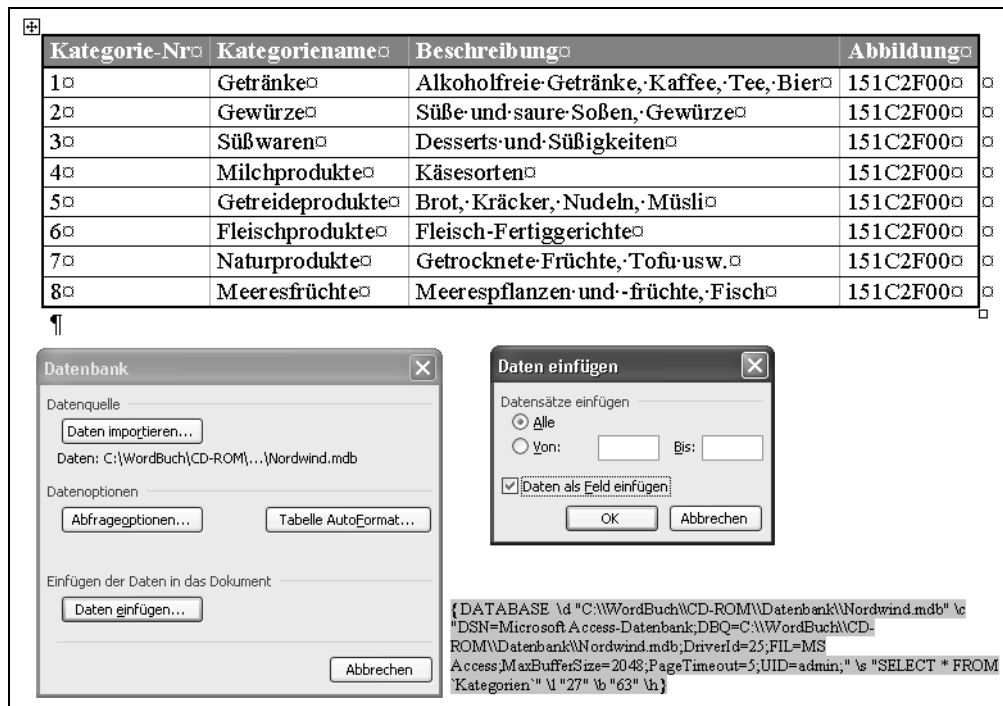
- Die Schnittstelle wird von allen Word-Versionen unterstützt.
- Wird eine dynamische Verknüpfung aufgebaut, funktioniert diese Schnittstelle am zuverlässigsten.

Mit der Schaltfläche *Abfrageoptionen* können die Daten gefiltert werden. Die Schaltfläche *Tabelle AutoFormat* stellt eine Auswahl an Formatierungen zur Verfügung, die auch bei einer Aktualisierung

der Daten beibehalten werden. (Tabellenformatvorlagen werden in dieser Liste nicht aufgeführt.) Im letzten Schritt, *Daten einfügen*, wird die Tabelle eingefügt. Bitte beachten Sie das Kontrollkästchen *Daten als Feld einfügen*: nur wenn dieses aktiviert ist, besteht eine dynamische Verbindung zur Datenquelle.

Die resultierende *Database*-Feldfunktion, die im Text-Argument der *Fields.Add*-Methode zu benutzen ist, erscheint unten rechts in der Abbildung. Sie enthält die volle Pfadangabe zur Datenbank (relative Pfadangaben werden von der *Database*-Feldfunktion nicht unterstützt), die Verbindungsangabe für den ODBC-Treiber sowie eine *Select*-Anweisung, um die gewünschten Datensätze (in diesem Fall alle) festzulegen. Ferner bestimmen die Schalter *\l* und *\b* das Tabellen-AutoFormat und der Schalter *\h*, dass die Tabelle im HTML-Format von Word darzustellen ist.

Abbildg. 10.5 Daten aus einer Access-Datenbank mit der Feldfunktion *Database* in Word als Tabelle verknüpfen



Die Beispieldatei *Bsp10\_03.doc* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap10*. Die Beispieldatenbank *Nordwind.mdb* befindet sich im Ordner *\Datenbank*.

Daten  
direkt  
anspre-  
chen

Für die direkte Daten-Verbindung aus dem Programm heraus empfehlen wir DAO oder ADO. Access unterstützt ODBC nur über einen Untersatz von DAO – ODBCDirect. Diese Verbindung würde somit einen »Umweg« darstellen. Im aktuellen Beispiel zeigen wir die Verwendung einer ADO-Verbindung auf.

Bei ODBC wird die Kommunikation zwischen der Anwendung und der Datenbank mit einem anwendungsspezifischen ODBC-Treiber umgesetzt. Dies besorgt bei ADO ein OLE DB-Provider für die Datenschnittstelle. Das Verbindungsprotokoll (»Connection string«), das die Verbindung her-

stellt, ist OLE DB-Provider-spezifisch. Eine umfassende Liste der »Connection strings« für mehrere Datenbanktypen finden Sie unter <http://www.carlprothman.net/Default.aspx?tabid=81>.

Für unser Beispiel verwenden wir als Datenquelle die Tabelle *Personal* (Abbildung 10.6) der Beispieldatenbank *Nordwind.mdb*. Diese Datei ist im Lieferumfang von Office Professional enthalten. Der Code in der Dokumentvorlage (*Bsp10\_04.dot*) wird eine Liste der Namen erzeugen und zur Auswahl vorlegen, so dass der Anwender einen Brief an die ausgewählte Person schreiben kann. Dabei werden Adresse, Betreffzeile sowie Anrede vom Programm in das neue Dokument übernommen.

**Abbildg. 10.6** Die *Personal*-Tabelle der Access-Datenbank *Nordwind.mdb*

Personal : Tabelle								
	Personal-Nr	Nachname	Vorname	Position	Anrede	Geburtsdatum	Einstellung	Straße
+	1	Davolio	Nancy	Vertriebsmitarbeiterin	Frau	08. Dez. 1968	01. Mai. 1992	507 - 20th Ave. E.
+	2	Fuller	Andrew	Geschäftsführer	Herr	19. Feb. 1952	14. Aug. 1992	908 W. Capital Way
+	3	Leverling	Janet	Vertriebsmitarbeiterin	Frau	30. Aug. 1963	01. Apr. 1992	722 Moss Bay Blvd.
+	4	Peacock	Margaret	Vertriebsmitarbeiterin	Frau	19. Sep. 1958	03. Mai. 1993	4110 Old Redmond Rd.
+	5	Buchanan	Steven	Vertriebsmanager	Herr	04. Mrz. 1955	17. Okt. 1993	14 Garrett Hill
+	6	Suyama	Michael	Vertriebsmitarbeiter	Herr	02. Jul. 1963	17. Okt. 1993	Coventry House
+	7	King	Robert	Vertriebsmitarbeiter	Dr.	29. Mai. 1960	02. Jan. 1994	Edgeham Hollow
+	8	Callahan	Laura	Vertriebskoordinatorin	Frau	09. Jan. 1958	05. Mrz. 1994	4726 - 11th Ave. N.E.
+	9	Dodsworth	Anne	Vertriebsmitarbeiterin	Frau	02. Jul. 1969	15. Nov. 1994	7 Houndstooth Rd.
*	(AutoWert)							

Datensatz: 1 von 9

Das Resultat sowie die Symbolleiste mit der Liste sehen Sie in Abbildung 10.7. Um die eingefügten Daten sind Textmarkenklammern ersichtlich. Die Textmarken kennzeichnen die Zielbereiche und werden um die Daten wieder hergestellt (Textmarken wurden in Kapitel 6 näher vorgestellt).

Die Symbolleiste wird beim Erstellen des Dokuments neu angelegt und mit einer Liste der Namen aus der Access-Tabelle gefüllt. Es wird angenommen, dass der Brief nach dem Erstellen nicht geändert wird. Die Symbolleiste ist also temporär und wird beim Schließen des Dokuments gelöscht. Mehr zum Erstellen und Verwalten von Symbolleisten ist in Kapitel 14 beschrieben. Das Datum bleibt ebenfalls statisch, es wird anhand einer *CreateDate*-Feldfunktion generiert.

Der Code für die beschriebene Aufgabe befindet sich in Listing 10.15. Um ADO mit Early Binding zu betreiben, muss ein Verweis auf eine der ADO-Bibliotheken (2.1 bis 2.8) gesetzt werden. Für ein einfaches Lesen und Schreiben von Daten reicht die Version 2.1 aus. Der Zugang zur ADO-Hilfe erfolgt am einfachsten mit der **[F1]**-Taste, sobald sich die Einfügemarke auf einem Ausdruck wie *Connection* oder *Recordset* befindet.

Die Prozedur *AutoNew* wird automatisch beim Erstellen eines neuen Dokuments ausgeführt. Ein *ADODB.Recordset* wird angelegt und in der Prozedur *AccessDatenHolen* mit den Informationen aus den Feldern *Nachname* und *Vorname* der Tabelle *Personal* gefüllt.

In der Prozedur *AccessDatenHolen* wird eine *ADODB.Connection* (Verbindung) zur Datenbank hergestellt, der *Recordset* vordefiniert und geöffnet. Da kein weiterer Datenaustausch mit diesen Daten stattfindet, wird danach die Verbindung getrennt, um Ressourcen auf dem Rechner und im Netzwerk wieder freizugeben.



Abbildg. 10.7 Die Daten aus der Tabelle wurden zusammengestellt und in die Textmarken geschrieben.

Northwind GmbH  
Personalabteilung

München, den 11. September 2005

Brief schreiben  
Empfänger auswählen Steven Buchar

Steven Buchanan  
14 Garrett Hill  
London SW1 8JR  
UK

Betreffend Ihrer Anfrage für eine Lohnerhöhung

Sehr geehrter Herr Buchanan

In *AutoNew* ist der nächste Schritt der Aufruf der Funktion *SymbolleisteMitComboErstellen*. Diese erstellt die temporäre Symbolleiste *Brief schreiben* mit einem Combobox-Steuerelement, das mit der Prozedur *BriefSchreiben* verbunden ist. Anschließend wird in einer Schleife die Liste mit den Daten aus dem Recordset gefüllt. Bitte beachten Sie, wie die Schleife bis zum »Dateiende« (EOF) ausgeführt wird und dass am Ende jeder Schleife ausdrücklich der nächsten Datensatz anzuwählen ist (RS.MoveNext).

Als letzte wichtige Handlung wird der Recordset geschlossen und die Variable freigestellt.

Auch die Prozedur *BriefSchreiben*, ausgelöst durch die Auswahl eines Listeneintrags, lässt *AccessDatenHolen* einen Recordset mit Daten füllen, dieses Mal mit allen Feldern (SELECT \* FROM Personal). Der Datensatz, der der Auswahl in der Combobox entspricht, wird angewählt. Die Daten werden mit statischem Text kombiniert und, zusammen mit dem Namen der Zieltextmarke und der Information, ob diese Textmarke markiert werden soll, der Funktion *DatenSchreiben* für das Einfügen in die Textmarken übergeben. Am Ende wird die Textmarke *InhaltAnfang* markiert, so dass der Anwender mit dem Schreiben des Briefinhalts beginnen kann.

**HINWEIS**

Mehr Informationen zu SQL-Anweisungen finden Sie in der Datei *SQL.pdf* auf der CD-ROM zum Buch im Ordner *\Beilagen\Zusatzmaterial Seriendruck*.

**Listing 10.15**    Daten über ADO-Verbindungen und Recordsets lesen und in ein Word-Dokument schreiben

```

'Wird automatisch bei der Erstellung eines neuen Dokuments ausgeführt
Sub AutoNew()
    Dim RS As ADODB.Recordset
    Dim cbo As Office.CommandBarComboBox

    Set RS = New ADODB.Recordset
    AccessDatenHolen RS, "SELECT Nachname, Vorname FROM Personal"
    'Die Symbolleiste wird in jedem Dokument neu erstellt.
    'Da sie temporär ist, geht sie nach dem Schließen des Dokuments verloren.
    Set cbo = SymbolleisteMitComboErstellen
    'Die Dropdown-Liste zeigt die Vor- und Nachnamen der möglichen Empfänger an.
    Do While Not RS.EOF
        cbo.AddItem RS.Fields("Vorname").Value & " " & RS.Fields("Nachname").Value
        RS.MoveNext
    Loop
    RS.Close
    Set RS = Nothing
End Sub

'Um diese Prozedur auszuführen, muss ein Verweis auf eine ADO-Bibliothek aktiviert sein.
Sub AccessDatenHolen(ByRef RS As ADODB.Recordset, strSQL As String)
    Dim conn As ADODB.Connection

    Set conn = New ADODB.Connection
    conn.Open "Provider=Microsoft.Jet.OLEDB.4.0;" &
        "Data Source=c:\WordBuch\Datenbank\nordwind.mdb;"
    Set RS.ActiveConnection = conn
    RS.CursorLocation = adUseClient
    RS.CursorType = adOpenStatic
    RS.LockType = adLockOptimistic
    RS.Open Source:=strSQL
    'Da Daten nur gelesen und nicht geschrieben werden, können wir
    'die Verbindung kappen und Ressourcen sparen.
    Set RS.ActiveConnection = Nothing

    conn.Close
    Set conn = Nothing
    'Debug.Print RS.Fields.Count, RS.RecordCount
End Sub

Private Function SymbolleisteMitComboErstellen() As Office.CommandBarComboBox
    Dim cb As Office.CommandBar
    Dim cbo As Office.CommandBarComboBox

    Application.CustomizationContext = ActiveDocument
    Set cb = Application.CommandBars.Add(Name:="Brief schreiben", _
        Position:=msoBarFloating, MenuBar:=False, Temporary:=True)
    Set cbo = cb.Controls.Add(Type:=msoControlComboBox)
    cbo.Caption = "Empfänger auswählen"
    cbo.Style = msoComboLabel
    cbo.DropDownLines = 5
    'Das Makro dieses Namens wird bei der Auswahl eines Eintrags ausgeführt.
    cbo.OnAction = "BriefSchreiben"

    Set SymbolleisteMitComboErstellen = cbo
    cb.Visible = True

```

Listing 10.15 Daten über ADO-Verbindungen und Recordsets lesen und in ein Word-Dokument schreiben (Fortsetzung)

```

End Function

Sub BriefSchreiben()
    Dim RS As ADODB.Recordset
    Dim doc As Word.Document
    Dim strAnredeZusatz As String

    Set RS = New ADODB.Recordset
    Set doc = ActiveDocument
    strAnredeZusatz = ""
    'Alle Datensätze holen
    AccessDatenHolen RS, "SELECT * FROM Personal"
    'Den Datensatz auswählen, der der Listenauswahl entspricht.
    RS.Move Application.CommandBars.ActionControl.ListIndex - 1
    'Die Informationen in die Textmarken schreiben
    DatenSchreiben "EmpfängerAdresse", RS.Fields("Vorname").Value & " " & _
        RS.Fields("Nachname").Value & vbCrLf & RS.Fields("Straße").Value & _
        vbCrLf & RS.Fields("Ort").Value & " " & RS.Fields("PLZ").Value & _
        vbCrLf & RS.Fields("Land").Value, doc, False
    'Den Benutzer zur Eingabe der Betreffzeile auffordern.
    DatenSchreiben "Betreffzeile", InputBox("Betreffzeile eingeben"), doc, False
    'Den Ausdruck "Sehr geehrte(r)" dem Geschlecht des Empfängers anpassen.
    If RS.Fields("Anrede").Value = "Herr" Then
        strAnredeZusatz = "r"
    End If
    DatenSchreiben "Anrede", "Sehr geehrte" & strAnredeZusatz & " " & _
        & RS.Fields("Anrede").Value & " " & RS.Fields("Nachname").Value, doc, False
    DatenSchreiben "InhaltAnfang", "", doc, True
    RS.Close
    Set RS = Nothing
End Sub

'Falls die Textmarke nicht existiert, wird "Falsch" zurückgegeben.
Function DatenSchreiben(strTextmarke As String, strInhalt As String, _
    doc As Word.Document, bMarkieren As Boolean) As Boolean
    Dim rng As Word.Range
    Dim bkm As Word.Bookmark
    Dim bErfolg As Boolean

    'Die Textmarken werden nach Einfügen der Daten wieder hergestellt.
    If doc.Bookmarks.Exists(strTextmarke) Then
        Set rng = doc.Bookmarks(strTextmarke).Range
        rng.Text = strInhalt
        Set bkm = doc.Bookmarks.Add(strTextmarke, rng)
        If bMarkieren Then
            bkm.Select
        End If
        bErfolg = True
    Else
        bErfolg = False
    End If
    DatenSchreiben = bErfolg
End Function

```



Die Beispieldatei *Bsp10\_04.dot* finden Sie auf der CD-ROM zum Buch im Ordner \Beispiele\Kap10.

## Excel-Tabelle ansprechen

Excel ist heute fast allgegenwärtig. Für das Erstellen von Listen sowie Berechnungen und Analysen von Daten ist es ein hervorragendes Werkzeug. Kein Wunder, dass es oft in Zusammenhang mit Word eingesetzt wird.

### HINWEIS

Das Einfügen und das Verknüpfen von Excel-Tabellen in Word-Dokumente wurde bereits in Kapitel 6 im Abschnitt zu den Feldfunktionen aufgezeigt. In Kapitel 11 wird das Erstellen und Bearbeiten von eingebetteten Excel-Objekten, zusammen mit einer kurzen Erklärung des Excel-Objektmodells, näher vorgestellt.

In diesem Abschnitt setzen wir die Diskussion über ADO-Verbindungen fort. Excel besitzt die ähnliche Funktionalität einer Datenbank und stellt eine entsprechende Schnittstelle zur Verfügung, die über den gleichen OLE DB-Provider wie Access angesprochen werden kann. Das folgende Beispiel soll aufzeigen, wie Daten von Word aus über eine solche Verbindung zurück an eine Datenbank geschrieben werden.

Diese Aufgabe kann auf mehreren Wegen erledigt werden. So ist es beispielsweise möglich, eine Verbindung zur Datenbank herzustellen, auf Basis einer Tabelle einen leeren Recordset zu erstellen, ihm neue Datensätze hinzuzufügen und diese mit Daten aus einem Word-Dokument zu füllen. Anschließend werden die Änderungen über die Verbindung zurück in die Datenbank geschrieben.

Da dieser Vorgang in Büchern und in der Dokumentation häufig vorkommt, haben wir uns für eine alternative Möglichkeit entschieden. Statt in mühsamer Arbeit Datensätze zu erstellen und die Felder eines nach dem andern mit Daten zu bestücken, werden die Daten in Zeichenketten gesammelt und mit der SQL-Anweisung `INSERT INTO` der Datentabelle hinzugefügt.

Abbildg. 10.8 Die Adressenangaben aus dem Formular werden einer Excel-Tabelle hinzugefügt.

Northwind-GmbH Personalabteilung	→	München, den 10. September 2005
Herr {Vorname} {Nachname} {Straße} {PLZ} {Ort}		
		SUBMIT Adresse

Die Ausgangslage ist in Abbildung 10.8 ersichtlich. Ein Word-Formular mit Formularfeldern für die Adresse steht bereit. Um eine neue Adresse in der Datenbank zu erfassen, befindet sich rechts daneben eine *Macrobutton*-Feldfunktion mit der Beschriftung *SUBMIT Adresse* in einem Positionsrahmen. Ein Doppelklick darauf führt die Prozedur in Listing 10.16 aus.

Die Syntax der Anweisung `Insert Into` lautet: `INSERT INTO [Tabellenname] ([Liste der Feldnamen]) VALUES ([Liste der Werte])`. Die Elemente beider Listen werden mit Kommas getrennt. Zeichenkettenwerte müssen von einfachen Anführungszeichen umgeben sein. Die Prozedur *DatenEinreichen* baut für das Formular in Abbildung 10.8 die folgende Zeichenkette auf:

```
INSERT INTO [PersonalTabelle$] (Anrede, Vorname, Nachname, Straße, PLZ, Ort) VALUES ('Herr', '[Vorname]', '[Nachname]', '[Straße]', '[PLZ]', '[Ort]')
```

Darin sehen Sie, wie eine Excel-Tabelle anzugeben ist: in eckigen Klammern, mit einem `$`-Zeichen am Schluss. Die Listen der Feldnamen und Werte werden in einer Schleife zusammengestellt, die alle Formularfelder durchläuft und deren Namen und Werte (`Result`-Eigenschaft) liest (mehr über Formularfelder steht in Kapitel 6 beschrieben).

Nachdem die Anweisung bereitsteht, wird eine Verbindung zur Excel-Datei geöffnet und die SQL-Anweisung mit der `Execute`-Methode ausgeführt. Abschließend wird die Verbindung wieder getrennt.

**Listing 10.16** Daten aus Word-Formularfeldern über eine ADO-Verbindung als neue Zeile einer Excel-Tabelle hinzufügen

```
Sub DatenEinreichen()
    Dim conn As ADODB.Connection
    Dim ffld As Word.FormField
    Dim doc As Word.Document
    Dim strFeldListe As String
    Dim strWertListe As String
    Dim strSQL As String

    Set doc = ActiveDocument
    strSQL = ""
    strFeldListe = "("
    strWertListe = "("
    For Each fffd In doc.FormFields
        strFeldListe = strFeldListe & fffd.Name & ", "
        strWertListe = strWertListe & "'" & Trim(ffld.Result) & "', "
    Next
    'Die letzten Kommas entfernen
    strFeldListe = Mid(strFeldListe, 1, Len(strFeldListe) - 2)
    strFeldListe = strFeldListe & ")"
    strWertListe = Mid(strWertListe, 1, Len(strWertListe) - 2)
    strWertListe = strWertListe & ")"
    strSQL = "INSERT INTO [PersonalTabelle$] " & strFeldListe & " VALUES " & strWertListe
    Set conn = New ADODB.Connection
    conn.Open "Provider=Microsoft.Jet.OLEDB.4.0;" & _
        "Data Source=c:\WordBuch\Datenbank\Personalstamm.xls;" & _
        "Extended Properties=""Excel 8.0;HDR=Yes""
    conn.Execute strSQL
    conn.Close
    Set conn = Nothing
End Sub
```



Die Beispieldatei *Bsp10\_05.dot* finden Sie auf der CD-ROM zum Buch im Ordner \Beispiele\Kap10. Die Excel-Arbeitsmappe *Personalstamm.xls* mit dem Arbeitsblatt *PersonalTabelle* ist im Ordner \Datenbank abgelegt.

## Zusammenfassung

In diesem Kapitel wurden verschiedene Anwendungen aus Word heraus ferngesteuert. Ein zweiter Schwerpunkt war dem Ansprechen von Datenbanken gewidmet.

- In diesem Kapitel wurde zunächst gezeigt, wie Excel ferngesteuert (Seite 510 ff.) oder zur Berechnung von komplexen Funktionen (Seite 512) herangezogen werden kann.
- Beim Steuern von PowerPoint (Seite 513 ff.) wurde erläutert, wie der Inhalt einer Präsentation in ein Dokument eingelesen (Seite 515) werden kann.
- Ein einfaches Beispiel zum Steuern von Visio wurde auf Seite 518 behandelt.
- Beim Zugriff auf Outlook (Seite 520 ff.) wurde vermittelt, wie auf die Kontakte zugegriffen (Seite 521) oder ein Dokument als E-Mail versendet (Seite 526) werden kann.
- Das Fernsteuern von Access (Seite 528 ff.) und der Zugriff auf Datenbanken (Seite 530 ff.) wurde ebenfalls behandelt.

## Kapitel 11

# Eingebettete OLE-Objekte

### In diesem Kapitel:

Excel-Tabellenobjekte	543
Excel-Diagramme	550
MS Graph-Diagramme	555
WordArt	559
ActiveX-Steuerelemente	564
Zusammenfassung	572

Im Prinzip ist ein Word-Dokument eine lange Zeichenkette, bestückt mit Formatierungsinformationen, die von der Anwendung dynamisch interpretiert werden, um die Seiten WYSIWYG («What you see is what you get») auf dem Bildschirm darzustellen oder auszudrucken. So fing jedenfalls alles an. Dann wurden die Benutzer anspruchsvoller und wünschten sich Grafiken, Tabellen und sogar Elemente aus anderen Anwendungen, wie beispielsweise Excel-Tabellen.

Mit der Zeit wurde die so genannte OLE-Technologie – »Object Linking and Embedding« – entwickelt. Damit wird ein Element aus einer Anwendung (der »OLE-Server«) in das »Dokument« einer anderen (der »OLE-Client« (Kunde)) eingebettet, und zwar mit einigen seiner eigenen Strukturen. Bei der Aktivierung eines OLE-Objekts wird es in der ursprünglichen Umgebung des OLE-Servers geöffnet und kann mit dessen Werkzeugen weiter bearbeitet werden. Manche Kombinationen von OLE-Server und OLE-Client unterstützen sogar das so genannte »in-place editing«: Die Menüs und Symbolleisten des OLE-Clients werden, mit einigen Ausnahmen, mit denjenigen des OLE-Servers ersetzt, und der Benutzer arbeitet weiter im gleichen Fenster.

**WICHTIG**

Um ein OLE-Objekt öffnen zu können, muss die OLE-Server-Anwendung auf dem Rechner korrekt installiert und registriert sein. In letzter Zeit häufen sich Meldungen, dass der OLE-Server nicht gefunden oder nicht gestartet werden könne. Meist liegt dem Problem ein Programm eines anderen Herstellers zugrunde, das in den normalen Ablauf eingreift. Verschwindet das Problem beim Laden von Windows im abgesicherten Modus, gehen Sie automatisch geladener Software wie Antiviren- und Desktopverwaltungs-Anwendungen nach, bis der Verursacher lokalisiert ist.

Gelegentlich wird es zur Aufgabe des Entwicklers, OLE-Objekte in ein Word-Dokument einzufügen oder vorhandene zu bearbeiten. Dies geht, sofern der OLE-Server eine entsprechende Automatisierungsschnittstelle zur Verfügung stellt, was bei vielen Office-Anwendungen der Fall ist.

In diesem Kapitel zeigen wir Ihnen, wie OLE-Objekte programmatisch in ein Word-Dokument eingefügt und automatisiert werden. Um die Syntax für das Erstellen eines eingebetteten Objekts zu ermitteln, empfehlen wir den Einsatz des Makrorekorders.

AddOLE  
Object

Abhängig von der verwendeten Word-Version sowie in Word 2002 und 2003 zudem von der Option *Bild einfügen als (Extras/Optionen, Registerkarte Bearbeiten)* wird das Objekt entweder »mit Text in Zeile« oder mit einem Textflussumbruch eingefügt. Im ersten Fall wird die AddOLEObject-Methode der InlineShapes-Auflistung, im zweiten der Shapes-Auflistung aufgezeichnet. Die folgende Codezeile wurde bei der Erstellung eines Excel-Objekts aufgezeichnet:

```
Selection.InlineShapes.AddOLEObject ClassType:="Excel.Sheet.8", LinkToFile:=False, _
    DisplayAsIcon:=False
```

Die Syntax von AddOLEObject unterscheidet sich für InlineShape- und Shape-Objekte nur unwesentlich. Für InlineShape gilt:

```
AddOLEObject(ClassType, FileName, LinkToFile, DisplayAsIcon, IconFileName, IconIndex,
    IconLabel, Range)
```



Für Shape gilt:

```
AddOLEObject(ClassType, FileName, LinkToFile, DisplayAsIcon, IconFileName, IconIndex,
IconLabel, Left, Top, Width, Height, Anchor)
```

Alle Argumente sind optional. Lediglich ClassType **oder** FileName muss vorhanden sein, sie schließen sich jedoch gegenseitig aus. Wird FileName verwendet, darf für LinkToFile nur True eingesetzt werden.

Der Zielbereich für ein InlineShape kann mit dem Range-Argument festgelegt werden, ansonsten wird das Objekt an der gegenwärtigen Markierung eingefügt. Für die Positionierung des Shape-Objekts dagegen stellt die Methode drei Argumente zur Verfügung – die übrigens auch von der AddPicture-Methode unterstützt werden: Left (links), Top (oben) und Anchor (Verankerungsbereich). Weitere Informationen zum Thema InlineShape und Shape finden Sie in Kapitel 6 im Abschnitt zu Grafiken.

OLEFormat

Da OLE-Objekte in den InlineShapes- und Shapes-Auflistungen geführt werden, werden sie von Word offensichtlich als grafische Objekte betrachtet. Und tatsächlich stehen die üblichen Befehle zum Formatieren und Positionieren zur Verfügung. Aber wie lassen sich OLE-Objekte in ihrer Ursprungsumgebung öffnen?

Auch hier hilft der Makrorekorder. Der aufgezeichnete Code zeigt, dass ein Zugang zu den OLE-Fähigkeiten durch die Eigenschaft OLEFormat und die Aktivierung des Objekts über die Methode DoVerb erreicht wird:

```
Selection.InlineShapes(1).OLEFormat.DoVerb VerbIndex:=wdOLEVerbPrimary
```

DoVerb-Methode

Wie das OLE-Objekt durch die DoVerb-Methode zu aktivieren ist, wird über das Argument VerbIndex geregelt. Da die hierfür benötigten WdOLEVerb-Konstantwerte ausführlich in der VBA-Hilfe beschrieben sind, finden Sie unten nur die wichtigsten aufgeführt:

- wdOLEVerbPrimary. Die standardmäßige Handlung für das Objekt.
- wdOLEVerbOpen. Das Objekt wird im Anwendungsfenster des OLE-Servers geöffnet.
- wdOLEVerbInPlaceActivate. Das Objekt wird (sofern es dies unterstützt) im Word-Dokument mit den Menüs und Symbolleisten des OLE-Servers geöffnet.

Zu beachten ist, dass nicht jede Objektklasse sämtliche WdOLEVerb-Konstantwerte unterstützt.

Automatisierung

Bei der Automatisierung eines OLE-Objekts nach seiner Aktivierung kann der Makrorekorder in Word allerdings nicht weiterhelfen. Dem grafischen Objekt muss eine Objektvariable des Typs OLEFormat gleichgestellt werden. Diese wird aktiviert und einer Objekt-Variablen des Typs OLE-Server zugewiesen – beispielsweise Excel.Workbook (Arbeitsmappe) wie in Listing 11.1.

Listing 11.1

Ein vorhandenes Excel-Objekt aktivieren und die erste Zelle unter dem Datenbereich markieren

```
Sub ExcelTabelleBearbeiten()
    Dim xlMappe As Excel.Workbook
    Dim xlBlatt As Excel.Worksheet
    Dim xlRng As Excel.Range
    Dim oleF As Word.OLEFormat

    Set oleF = ActiveDocument.InlineShapes(1).OLEFormat
```

**Listing 11.1** Ein vorhandenes Excel-Objekt aktivieren und die erste Zelle unter dem Datenbereich markieren (Fortsetzung)

```
oleF.DoVerb VerbIndex:=wdOLEVerbInPlaceActivate
Set xlMappe = oleF.Object
Set xlBlatt = xlMappe.Sheets(1)
Set xlRng = xlBlatt.UsedRange
'Die erste Zelle der ersten Zeile unter den bisherigen Daten markieren.
xlRng.Offset(1, 0).Select
End Sub
```



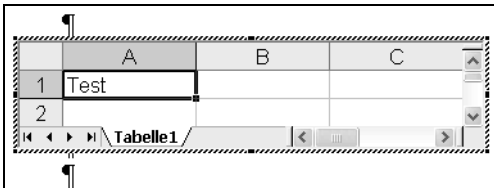
Die Beispieldatei *Bsp11\_01.doc* finden Sie auf der CD-ROM im Ordner *\Beispiele\Kap11*.

Danach muss mit dem Objektmodell des OLE-Servers gearbeitet werden, wie die letzten Codezeilen veranschaulichen. Das Resultat sehen Sie in Abbildung 11.1.

**PROFITIPP**

Zwar kann nicht der Makrorekorder von Word in einer OLE-Server-Anwendung weiterhelfen, unter Umständen aber deren Makrorekorder. Und ähnlich wie in Word sollten Sie das Resultat nachbearbeiten, um Begriffe wie *Selection* und *ActiveWorkbook* möglichst zu entfernen. Dies ist umso wichtiger, wenn der Code aus der Word-Umgebung heraus abgearbeitet wird, da Word ein Objekt zuerst in seiner eigenen Objekt-Bibliothek sucht. Muss aus irgendeinem Grund ein allgemeines Objekt wie *Selection* verwendet werden, sollte die Anwendung, auf die es sich bezieht, unbedingt präzisiert werden (zum Beispiel *xlApp.Selection*).

**Abbildg. 11.1** Eine im Word-Dokument eingebettete Excel-Arbeitsmappe wurde aktiviert und der Zelle A1 ein Wert zugewiesen.



Deakti-  
vieren

In der Benutzeroberfläche wird ein OLE-Objekt per Mausklick außerhalb des Objekts oder durch Drücken der **[ESC]**-Taste deaktiviert.

Bei der programmierten Steuerung steht die erste Option gar nicht zur Verfügung. Die zweite lässt sich zwar bedingt mit der Methode *SendKeys* realisieren, arbeitet jedoch unzuverlässig. Die Ausführung kann einige Sekunden dauern, zudem eignet sich die Methode nicht, wenn mehrere OLE-Objekte in Folge zu bearbeiten sind:

```
Application.SendKeys "{ESC}", True
```

Daher sollte generell die *Quit*-Methode des jeweiligen *Application*-Objekts verwendet werden, um ein OLE-Objekt zu verlassen. Je nach Anwendung wird diese bei der In-place-Aktivierung nicht unterstützt. In diesem Fall muss das Objekt in einem eigenen Anwendungsfenster geöffnet werden.

Bild-  
schirm

Die programmiertechnische Bearbeitung von OLE-Objekten ist auf dem Bildschirm sichtbar. Bisher wurde keine Möglichkeit gefunden, dies zu unterbinden. Falls sich das Geschehen im Hintergrund abspielen muss, sollten Sie das Objekt als *verknüpfte* Datei einbetten und die Quelldatei direkt bearbeiten.

## Excel-Tabellenobjekte

Wie im Abschnitt über Tabellen in Kapitel 6 erwähnt, eignet sich Excel bestens für Berechnungen und Datenanalysen. Solange der im Word-Dokument anzuzeigende Zellenbereich nicht größer ist als eine Seite, ist das Einfügen eines Excel-Tabellenobjekts eine Option, um Excel-Fähigkeiten in ein Word-Dokument einzubinden. (Die Arbeitsmappe darf durchaus über Inhalt verfügen, der im Dokument nicht sichtbar ist.)

Erstellen

Ein Beispiel für das Erstellen eines neuen Excel-Tabellenobjekts enthält das Listing 11.2. Es veranschaulicht den Gebrauch von Late Binding – alle Objektvariablen für das Excel-Objekt sind als Objekte deklariert. Somit ist kein Verweis auf eine Excel-Bibliothek notwendig. Bitte beachten Sie, wie das Arbeitsmappen-Objekt der Objektvariablen beim Erstellen zugewiesen wird.

Zum Schluss steht das Arbeitsblatt, wie in Abbildung 11.2 ersichtlich, bereit für die Benutzereingaben.

### HINWEIS

Da es sich hier um ein Word-Buch handelt, wird nicht im Detail auf das Excel-Objektmodell eingegangen. Häufig gebrauchte Objekte, Eigenschaften und Methoden können Sie den folgenden Beispielen entnehmen. Wie in Word liefert auch in Excel der Makrorekorder hilfreiche Informationen. Für vertiefte Diskussionen und Erklärungen verweisen wir auf ein Excel-Programmierbuch.

Abbildg. 11.2 Ein in einem Word-Dokument eingebettetes und aktiviertes Excel-Arbeitsblatt

	A	B	C	D	E	F	G
1	Artikel-Nr	Beschreibung	Anz	Preis	Total		
2							
3							
4							
5							
6							
7							
8							
9							
10							

Listing 11.2 Mit Late Binding ein Excel-Tabellenobjekt in einem Word-Dokument erstellen

```
Sub ExcelTabelleErstellen()
    Dim rng As Word.Range
    Dim doc As Word.Document
    Dim o As Object
    Dim oSheet As Object
```

**Listing 11.2** Mit Late Binding ein Excel-Tabellenobjekt in einem Word-Dokument erstellen *(Fortsetzung)*

```

Set doc = ActiveDocument
Set rng = doc.Content
'Das Tabellen-Objekt wird am Dokumentende eingefügt.
rng.Collapse wdCollapseEnd
Set o = ActiveDocument.InlineShapes.AddOLEObject( _
    "Excel.Sheet.8", Range:=rng).OLEFormat.Object)
Set oSheet = o.Sheets(1)
oSheet.Name = "Testdaten"
oSheet.Columns("C").HorizontalAlignment = xlCenter
oSheet.Columns("D").HorizontalAlignment = xlRight
oSheet.Columns("E").HorizontalAlignment = xlRight
oSheet.Columns("D").NumberFormat = "#,##0.00"
oSheet.Columns("E").NumberFormat = "#,##0.00 €"
oSheet.Range("A1").Value = "Artikel-Nr"
oSheet.Range("B1").Value = "Beschreibung"
oSheet.Range("C1").Value = "Anz"
oSheet.Range("D1").Value = "Preis"
oSheet.Range("E1").Value = "Total"
oSheet.Range("A2").Select
End Sub

```

Auffallend ist, dass das Objekt mehr Spalten anzeigt als benötigt. Leider lässt sich programmiertechnisch die Größe eines Excel-Tabellenobjekts nicht festlegen. Der Anwender kann jedoch die Anzahl sichtbarer Spalten und Zeilen ändern, indem er die Anfasser eines aktiven Objekts zieht; dafür gibt es in keinem Objektmodell ein Gegenstück. Die *Width*- und *Height*-Eigenschaften für *InlineShape*- und *Shape*-Objekte beziehen sich ausschließlich auf das grafische Objekt und nicht auf seinen OLE-Inhalt.

Die einzige Möglichkeit, auf den sichtbaren Excel-Bereich Einfluss zu nehmen, besteht darin, eine existierende Excel-Datei in Word zu importieren. Word berechnet dann die Anzahl der einzublen-denden Zeilen und Spalten anhand des belegten Bereichs im ersten Tabellenblatt. Die Abbildung 11.3 zeigt, dass die Prozedur aus Listing 11.3 beim Erstellen des Excel-Objekts auch dessen sichtbaren Bereich bestimmt.

**Abbildg. 11.3** Durch Einfügen einer bestehenden Arbeitsmappe kann die Anzahl sichtbarer Spalten und Zeilen festgelegt werden.

	A	B	C	D	E
1	Artikel-Nr	Beschreibung	Anz	Preis	Total
2	Start				

In dieser Prozedur wird zuerst die Excel-Anwendung unsichtbar gestartet, eine Arbeitsmappe angelegt und der Inhalt eingefügt. Im Gegensatz zu Listing 11.2 veranschaulicht dieses Beispiel den Einsatz von Early Binding. Nachdem alle nicht benötigten Arbeitsblätter entfernt sind, wird die Mappe gespeichert und geschlossen. Bitte beachten Sie, wie anschließend alle Excel-Objektvariablen freigestellt werden. Danach wird die Excel-Datei in Word importiert.

Listing 11.3 Eine Excel-Mappe erstellen, dann in Word importieren

```

Sub TabellenObjektAusDateiEinfügen()
    Dim xlApp As Excel.Application
    Dim xlMappe As Excel.Workbook
    Dim xlBlatt As Excel.Worksheet
    Dim lZaehler As Long
    Dim rng As Word.Range
    Dim fld As Word.Field
    Dim ils As Word.InlineShape
    Dim strArbeitsmappenPfad

    strArbeitsmappenPfad = ActiveDocument.Path & "\Bsp11_01.xls"
    'Die Arbeitsmappe zuerst erstellen und speichern
    Set xlApp = New Excel.Application
    xlApp.DisplayAlerts = False
    'Aktivieren beim Testen, für den Fall, dass etwas schief geht.
    'xlApp.Visible = True
    'Unterdrückt Meldungen
    xlApp.DisplayAlerts = False
    Set xlMappe = xlApp.Workbooks.Add
    Set xlBlatt = xlMappe.Sheets(1)
    xlBlatt.Name = "Testdaten"
    'Die Spalten mit Zahlen ausrichten.
    xlBlatt.Columns("C").HorizontalAlignment = xlCenter
    xlBlatt.Columns("D").HorizontalAlignment = xlRight
    xlBlatt.Columns("E").HorizontalAlignment = xlRight
    'Die Zahlenformatierung festlegen.
    xlBlatt.Columns("D").NumberFormat = "#,##0.00"
    xlBlatt.Columns("E").NumberFormat = "#,##0.00 €"
    'Die Spaltenüberschriften eingeben...
    xlBlatt.Range("A1").Value = "Artikel-Nr"
    xlBlatt.Range("B1").Value = "Beschreibung"
    xlBlatt.Range("C1").Value = "Anz"
    xlBlatt.Range("D1").Value = "Preis"
    xlBlatt.Range("E1").Value = "Total"
    '...und formatieren.
    xlBlatt.UsedRange.Font.Bold = True
    'Um sicher zu gehen, dass die zweite Zeile sichtbar ist,
    'müssen Daten darin stehen.
    xlBlatt.Range("A2").Value = "Start"
    xlBlatt.Range("A2").Select
    'Überflüssige Arbeitsblätter entfernen.
    For lZaehler = xlMappe.Sheets.Count To 2 Step -1
        xlMappe.Sheets(lZaehler).Delete
    Next
    'Die Mappe speichern, schließen und Excel beenden.
    xlMappe.Close SaveChanges:=True, FileName:=strArbeitsmappenPfad
    xlApp.Quit
    Set xlBlatt = Nothing
    Set xlMappe = Nothing
    Set xlApp = Nothing

    'In Word das Tabellen-Objekt einfügen und aktivieren.
    Set rng = ActiveDocument.Range
    rng.Collapse wdCollapseEnd
    Set ils = rng.InlineShapes.AddOLEObject(FileName:=strArbeitsmappenPfad, _
        LinkToFile:=False, DisplayAsIcon:=False)
    ils.OLEFormat.DoVerb VerbIndex:=wdOLEVerbPrimary
End Sub

```

**Listing 11.4** Dieses Listing für C# ist stellvertretend für alle *AddOLEObject*-Beispiele für Excel. Es zeigt einige Grundlagen der Excel-Automatisierung in der .NET-Umgebung auf.

```
private void ExcelTabellenObjektErstellen_CS()
{
    try
    {
        object objMissing = System.Reflection.Missing.Value;
        string arbeitsmappenPfad = "C:\\WordBuch\\Beispiele\\Kap11\\Bsp11_01.xls";
        //Die Arbeitsmappe zuerst erstellen und speichern
        xl.Application xlApp = new xl.Application();
        //Aktivieren beim Testen, für den Fall, dass etwas schief geht.
        //xlApp.Visible = true;
        xlApp.DisplayAlerts = false;
        xl.Workbook xlMappe = xlApp.Workbooks.Add(Type.Missing);
        xl.Worksheet xlBlatt = (xl.Worksheet) xlMappe.Sheets[1];
        xlBlatt.Name = "Testdaten";
        //Die Spalten mit Zahlen ausrichten.
        xlBlatt.get_Range("C1", Type.Missing).EntireColumn.HorizontalAlignment =
            xl.XlHAlign.XlHAlignCenter;
        xlBlatt.get_Range("D1", Type.Missing).EntireColumn.HorizontalAlignment =
            xl.XlHAlign.XlHAlignRight;
        xlBlatt.get_Range("E1", Type.Missing).EntireColumn.HorizontalAlignment =
            xl.XlHAlign.XlHAlignRight;
        //Die Zahlenformatierung festlegen.
        xlBlatt.get_Range("D1", Type.Missing).EntireColumn.NumberFormat = "#,##0.00";
        xlBlatt.get_Range("E1", Type.Missing).EntireColumn.NumberFormat = "#,##0.00 €";
        //Die Spaltenüberschriften eingeben ...
        xlBlatt.get_Range("A1", Type.Missing).Value2 = "Artikel-Nr";
        xlBlatt.get_Range("B1", Type.Missing).Value2 = "Beschreibung";
        xlBlatt.get_Range("C1", Type.Missing).Value2 = "Anz";
        xlBlatt.get_Range("D1", Type.Missing).Value2 = "Preis";
        xlBlatt.get_Range("E1", Type.Missing).Value2 = "Total";
        //... und formatieren.
        xlBlatt.UsedRange.Font.Bold = true;
        //Um sicher zu gehen, dass die zweite Zeile sichtbar ist,
        //müssen Daten darin stehen.
        xlBlatt.get_Range("A2", Type.Missing).Value2 = "Start";
        xlBlatt.get_Range("A2", Type.Missing).Select();
        //Überflüssige Arbeitsblätter entfernen.
        for (int zaehler = xlMappe.Sheets.Count; zaehler > 1; zaehler --)
        {
            xl.Worksheet xlVorigesBlatt = (xl.Worksheet) xlMappe.Worksheets[zaehler];
            xlVorigesBlatt.Delete();
            xlVorigesBlatt = null;
        }
        //Die Mappe speichern, schließen und Excel beenden.
        xlMappe.Close(true, arbeitsmappenPfad, Type.Missing);
        xlApp.Quit();
        xlBlatt = null;
        xlMappe = null;
        wdMarshal.ReleaseComObject(xlApp);
        xlApp = null;

        //In Word das Tabellen-Objekt einfügen und aktivieren.
        wd.Application wdApp = (wd.Application) wdMarshal.GetActiveObject("Word.Application");
        wd.Range rng = wdApp.ActiveDocument.Content;
        rng.InsertAfter("\n");
    }
}
```

**Listing 11.4** Dieses Listing für C# ist stellvertretend für alle *AddOLEObject*-Beispiele für Excel. Es zeigt einige Grundlagen der Excel-Automatisierung in der .NET-Umgebung auf. (Fortsetzung)

```
object objCollapseEnd = wd.WdCollapseDirection.wdCollapseEnd;
rng.Collapse(ref objCollapseEnd);
object fileName = arbeitsmappenPfad;
object objFalse = false;
object objRng = (object) rng;
wd.InlineShape ils = rng.InlineShapes.AddOLEObject(ref objMissing, ref fileName,
    ref objFalse, ref objMissing, ref objMissing, ref objMissing,
    ref objRng);
object oleVerbPrimary = wd.WdOLEVerb.wdOLEVerbPrimary;
rng = null;
ils.OLEFormat.DoVerb(ref oleVerbPrimary);
wdApp.Activate();
wdMarshal.ReleaseComObject(wdApp);
wdApp = null;
}
catch (System.Exception ex)
{
    MessageBox.Show("Fehlermeldung: " + ex.Message + " in " + ex.Source);
}
}
```



Die Beispieldatei *Bsp11\_01.doc* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap11*.

Bearbeiten

In Listing 11.1 wurde demonstriert, wie ein einzelnes Objekt für die Bearbeitung durch den Benutzer aktiviert wird. Der Vorgang, ein Objekt zu bearbeiten und zu schließen oder mehrere Objekte in Folge zu automatisieren, sieht ein wenig anders aus. Wie eingangs erwähnt, soll die *Quit*-Methode des *Application*-Objekts verwendet werden, um in das Word-Dokument zurückzukehren. Bei Excel geht das nur, wenn das Objekt in einem eigenen Excel-Fenster geöffnet wird.

In Listing 11.5 werden alle grafischen Objekte eines Dokuments durchlaufen – sowohl jene »mit Text in Zeile« als auch solche, die mit Textfluss formatiert sind. Handelt es sich um ein OLE-Objekt des Typs *Excel.Sheet.8* (Excel-Arbeitsblatt), wird jeweils die Prozedur *ExcelObjektBearbeiten* aufgerufen. Diese öffnet das *OLEFormat*-Objekt in einem eigenständigen Excel-Fenster, wo der benutzte Bereich ermittelt und formatiert wird. Auf diese Weise werden alle Tabellenobjekte im Dokument einheitlich formatiert (siehe Abbildung 11.4).

**Abbildg. 11.4** Alle Excel-Tabellenobjekte eines Dokuments wurden programmgesteuert gleich formatiert.

Test						

Artikel-Nr	Beschreibun	Anz	Preis	Total		
One						
Two						
Three						

Artikel-Nr	Beschreibun	Anz	Preis	Total
Start				

Bitte beachten Sie, wie in Word mit der Exists-Eigenschaft geprüft werden kann, ob eine Anwendung gegenwärtig aktiv ist. Falls ja, wird die Anwendung am Schluss nicht beendet, sondern in ihrem ursprünglichen Zustand belassen.

**Listing 11.5** Alle Excel-Tabellenobjekte eines Dokuments in einem Excel-Fenster öffnen und formatieren

```

Sub AlleExcelTabellenBearbeiten()
    Dim ils As Word.InlineShape
    Dim shp As Word.Shape
    Dim oleF As Word.OLEFormat
    Dim bExcel As Boolean

    'Falls Excel schon läuft, diesen Zustand festhalten.
    bExcel = Application.Tasks.Exists("Microsoft Excel")

    'Zuerst durch alle InlineShapes schleifen.
    For Each ils In ActiveDocument.InlineShapes
        'Nur wenn es sich um ein Excel-Tabellenobjekt handelt...
        If ils.Type = wdInlineShapeEmbeddedOLEObject Then
            Set oleF = ils.OLEFormat
            If oleF.ClassType = "Excel.Sheet.8" Then
                '...wird es weiter bearbeitet
                ExcelObjektBearbeiten oleF, bExcel
            End If
        End If
    Next
    'Anschließend durch alle Shapes schleifen.
    For Each shp In ActiveDocument.Shapes
        If shp.Type = msoEmbeddedOLEObject Then
            Set oleF = shp.OLEFormat
            If oleF.ClassType = "Excel.Sheet.8" Then
                ExcelObjektBearbeiten oleF, bExcel
            End If
        End If
    Next
End Sub

```



Listing 11.5 Alle Excel-Tabellenobjekte eines Dokuments in einem Excel-Fenster öffnen und formatieren (Fortsetzung)

```

        End If
    End If
Next
End Sub

'Excel-Tabelle in einem Excel-Fenster öffnen und formatieren
'Wenn Excel nicht schon läuft, wird es jedes Mal wieder beendet.
Private Sub ExcelObjektBearbeiten(oleF As Word.OLEFormat, bExcel As Boolean)
    Dim xlApp As Excel.Application
    Dim xlMappe As Excel.Workbook
    Dim xlRange As Excel.Range
    Dim lAnzZeilen As Long
    Dim lAnzSpalten As Long
    Dim lOffsetZeile As Long

    'Excel-Tabelle explizit in einem Excel-Fenster öffnen
    oleF.DoVerb VerbIndex:=wdOLEVerbOpen
    Set xlMappe = oleF.Object
    Set xlApp = xlMappe.Application
    'Nur den Bereich bearbeiten, der Daten enthält
    Set xlRange = xlMappe.ActiveSheet.UsedRange
    'Erste Zeile dunkel hinterlegt mit weißem, fettem Text formatieren
    With xlRange.Rows(1).CurrentRegion
        .Interior.Color = RGB(100, 100, 100)
        .Interior.Pattern = xlSolid
        .Font.FontStyle = "Fett"
        .Font.Color = RGB(255, 255, 255)
    End With
    lAnzZeilen = xlRange.Rows.Count
    lAnzSpalten = xlRange.Columns.Count
    'Die restlichen Zeilen, falls vorhanden, hellgrau hinterlegen
    If lAnzZeilen > 1 Then
        lOffsetZeile = 2
        Set xlRange = xlRange.Range(xlRange.Cells(lOffsetZeile, 1), _
            xlRange.Cells(lAnzZeilen, lAnzSpalten))
        With xlRange
            .Interior.Color = RGB(200, 200, 200)
            .Interior.Pattern = xlSolid
        End With
    End If
    'Die Mappe schließen
    xlMappe.Close False
    Set xlMappe = Nothing
    DoEvents
    'Falls Excel vor der Ausführung nicht schon lief, beenden
    If Not bExcel Then
        xlApp.Quit
    End If
    Set xlApp = Nothing
End Sub

```

**TIPP**

Auch wenn eine In-place-Aktivierung spezifiziert wird, öffnet Word unter gewissen Umständen ein OLE-Objekt in einem getrennten Anwendungsfenster:

- Die Feldcode-Anzeige ist für das Dokument aktiviert.
- Das Objekt ist mit einer Datei verknüpft.
- Word hat zu wenig Ressourcen, um die In-place-Aktivierung zu unterstützen.




Die Beispieldatei *Bsp11\_01.doc* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap11*.

## Excel-Diagramme

Nicht nur Excel-Tabellen können in ein Word-Dokument eingebettet werden. Auch Diagramme werden als OLE-Objekte unterstützt und verleihen dem Textinhalt eine zusätzliche Ausdruckskraft.

**PROFITIPP**

Wegen der mit ihnen gespeicherten Datei-Strukturen lassen OLE-Objekte die Dateigröße stark anwachsen. Sofern Sie kein dynamisches Diagramm brauchen, raten wir daher, das Diagramm als Grafik zu kopieren und einzufügen: Zunächst markieren Sie es in Excel und öffnen bei gedrückter -Taste das Menü *Bearbeiten*. Darin wählen Sie nun den Befehl *Bild kopieren*. Im Word-Dokument schließlich wird aus dem Dialogfeld zum Menübefehl *Bearbeiten/Inhalte einfügen* der gewünschte Grafik-Typ festgelegt.

Der Makrorekorder verrät folgende Syntax, um ein Excel-Diagramm einzufügen, wobei der Klassentyp »Excel.Chart.8« verwendet wird:

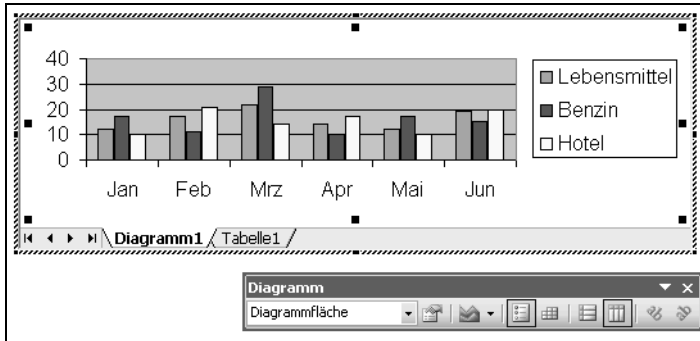
```
Selection.InlineShapes.AddOLEObject ClassType:="Excel.Chart.8",
    FileName:= "", LinkToFile:=False, DisplayAsIcon:=False
```

Während das Einfügen einer Excel-Tabelle nur ein einziges Arbeitsblatt im OLE-Objekt generiert, erhält das neue Diagramm-Objekt zuvorderst ein Diagrammblatt und dahinter ein Tabellenblatt (siehe Abbildung 11.5), welches die Daten für das Diagramm enthält. Der Automatisierungscode hat demzufolge zwei Aufgaben zu erledigen:

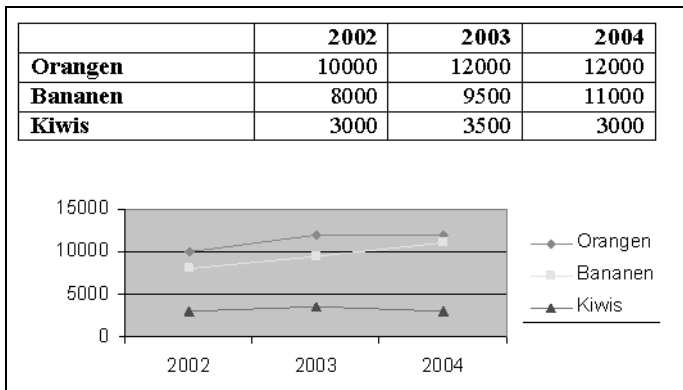
- die im Tabellenblatt hinterlegten Beispieldaten durch die benötigten zu ersetzen und
- das Diagramm nach Wunsch zu formatieren.

Unser folgendes Beispiel zeigt, wie die Daten aus einer im Dokument befindlichen Word-Tabelle in die Excel-Tabelle übernommen werden. Anschließend wird die Datenquelle des Diagramms angepasst sowie der Typ und die Formatierung geändert. Das Resultat sehen Sie in Abbildung 11.6, den Beispielcode in Listing 11.6.

Abbildg. 11.5 Standardmäßig umfasst ein neues Diagramm-Objekt, ein Diagramm- sowie ein Tabellenblatt.



Abbildg. 11.6 Diagramm erstellt auf Basis einer Word-Tabelle. Die Tabellendaten werden in das Excel-Tabellenblatt hinter dem Diagramm eingetragen.



Den Eingangspunkt bildet die Prozedur *DiagrammEinfuegen*. Als erster Schritt wird die erste Tabelle im Dokument einer Objektvariablen zugewiesen. Der Zielbereich für das Diagramm wird in einem darauf folgenden Absatz festgelegt.

Danach wird ein ADO-Recordset angelegt und, zusammen mit der Word-Tabelle, der Prozedur *DatenLaden* übergeben. Hier sehen Sie, wie ein ADO-Recordset von Grund auf erstellt und mit Daten gefüllt wird, ohne Verbindung zu einer Datenbank als Datenquelle. Die erste Tabellenzeile liefert die Feldnamen. Dann wird durch die übrigen Zeilen geschleift, um Datensätze zu bilden.

Nun, da die Daten bereit liegen, wird das Diagramm-Objekt eingefügt und aktiviert. Arbeitsmappe, Tabellenblatt sowie Diagrammblatt werden ihren Objekt-Variablen zugewiesen. Zunächst werden die Beispieldaten aus dem Tabellenblatt gelöscht, dann die Spaltenüberschriften, die als Diagramm-Legende erscheinen, anhand der Feldnamen des ADO-Recordsets angelegt. Die Daten werden, dank Excels *CopyFromRecordset*-Methode, in einem Schritt eingefügt.

Schließlich verpasst die Prozedur *DiagrammFormatieren* dem Diagramm den letzten Schliff. Der Diagrammtyp wird festgelegt. Anhand der Adresseninformation des benutzten Tabellenblattbereichs werden Quellbereiche für Diagramm- sowie X-Achsen-Werte ausgerechnet und zugewiesen. Auch die farbliche und förmliche Gestaltung der Datenserien werden angepasst.

**Listing 11.6** Ein Excel-Diagramm anhand einer Tabelle im Word-Dokument erstellen

```

Sub DiagrammEinfuegen()
    Dim xlDiagramm As Excel.Chart
    Dim xlMappe As Excel.Workbook
    Dim xlBlatt As Excel.Worksheet
    Dim doc As Word.Document
    Dim rng As Word.Range
    Dim ils As Word.InlineShape
    Dim oleF As Word.OLEFormat
    Dim tbl As Word.Table
    Dim rs As ADODB.Recordset
    Dim fld As ADODB.Field
    Dim lZaehler As Long

    Set doc = ActiveDocument
    Set tbl = doc.Tables(1)
    Set rng = tbl.Range
    'Diagramm unter der Tabelle einfügen
    rng.Collapse Direction:=wdCollapseEnd
    rng.InsertAfter vbCrLf
    rng.Collapse Direction:=wdCollapseEnd

    'Tabellendaten in ein ADODB-Recordset lesen
    Set rs = New ADODB.Recordset
    DatenLaden rs, tbl
    'Nach Füllen des Recordsets ist der letzte Datensatz markiert.
    'Deshalb muss der erste Datensatz ausgewählt werden.
    rs.MoveFirst

    'Das Diagramm im Dokument erstellen ...
    Set oleF = doc.InlineShapes.AddOLEObject(ClassType:="Excel.Chart.8", _
        Range:=rng).OLEFormat
    '... und aktivieren.
    oleF.DoVerb

    Der Arbeitsmappe, dem Datenblatt sowie Diagrammblatt Objektvariablen zuweisen
    Set xlMappe = oleF.Object
    Set xlBlatt = xlMappe.Sheets(2)
    Set xlDiagramm = xlMappe.Charts(1)
    'Die vorhandenen Daten löschen.
    xlBlatt.UsedRange.Clear
    'Die Spaltenüberschriften auf Basis der Feldnamen des Recordsets erstellen
    lZaehler = 0
    For Each fld In rs.Fields
        lZaehler = lZaehler + 1
        xlBlatt.Cells(1, lZaehler).Value = fld.Name
    Next fld
    'Die Daten, ab der 2. Tabellenblattzeile, einfügen.
    xlBlatt.Range("A2").CopyFromRecordset rs

    'Das Diagramm formatieren
    DiagrammFormatieren xlDiagramm, xlBlatt

    'Aufräumarbeiten
    rs.Close
    Set rs = Nothing
    Set xlDiagramm = Nothing

```

Listing 11.6 Ein Excel-Diagramm anhand einer Tabelle im Word-Dokument erstellen (Fortsetzung)

```

Set xlBlatt = Nothing
Set xlMappe = Nothing
End Sub

Sub DatenLaden(ByRef rs As ADODB.Recordset, tbl As Word.Table)
    Dim row As Word.Row
    Dim cel As Word.Cell
    Dim rng As Word.Range
    Dim fld As ADODB.Field
    Dim lAnzFelder As Long
    Dim lZaehler As Long
    Dim lFeldTyp As Long
    Dim strZellenInhalt As String

    rs.CursorLocation = adUseClient
    rs.CursorType = adOpenDynamic
    rs.LockType = adLockOptimistic
    lAnzFelder = tbl.Columns.Count
    For Each row In tbl.Rows
        'Die erste Zeile (Spaltenüberschrift) enthält den Feldnamen.
        If row.Index = 1 Then
            'Die Felder des Recordsets definieren
            For Each cel In row.Cells
                Set rng = cel.Range
                'Sicherstellen, dass auch verborgener Text gelesen wird (Die erste Zelle enthält
                'eine Spaltenbezeichnung, weil ein Laufzeitfehler bei leerem Zelleninhalt
                'vorkommt)
                rng.TextRetrievalMode.IncludeHiddenText = True
                strZellenInhalt = TrimZelle(rng.Text) & ""
                'Der Feldtyp ist entweder eine Zeichenkette oder eine Zahl
                lFeldTyp = FeldTypErmitteln(strZellenInhalt)
                rs.Fields.Append Name:=strZellenInhalt, Type:=lFeldTyp, _
                DefinedSize:=30, Attrib:=adFldMayBeNull
                Set rng = Nothing
            Next
            rs.Open
        Else
            'Das Recordset mit Datensätzen füllen...
            rs.AddNew
            '... ein Datensatz pro Zeile
            For lZaehler = 1 To lAnzFelder
                rs.Fields(lZaehler - 1).Value = _
                TrimZelle(row.Cells(lZaehler).Range.Text)
            Next lZaehler
        End If
    Next
End Sub

Sub DiagrammFormatieren(xlDiagramm As Excel.Chart, xlBlatt As Excel.Worksheet)
    Dim xlRange As Excel.Range
    Dim strBereichAdresse As String
    Dim lLetzteSpalte As Long
    Dim lLetzteZeile As Long

    With xlDiagramm
        .ChartType = xlLine
    End With

```

**Listing 11.6** Ein Excel-Diagramm anhand einer Tabelle im Word-Dokument erstellen (*Fortsetzung*)

```

strBereichAdresse = xlBlatt.UsedRange.AddressLocal(ReferenceStyle:=xlR1C1)
lLetzteSpalte = Right(strBereichAdresse, 1)
lLetzteZeile = Mid(strBereichAdresse, 7, 1)
Set xlRange = xlBlatt.Range( _
    xlBlatt.Cells(1, 2), xlBlatt.Cells(1, lLetzteSpalte))
.DatenserieFormatieren Source:=xlBlatt.Range( _
    xlBlatt.Cells(2, 1), xlBlatt.Cells(lLetzteZeile, lLetzteSpalte)), PlotBy:=xlRows

'Orangen sind orange.
DatenserieFormatieren .SeriesCollection(1), xlRange, xlMarkerStyleCircle, _
    5, RGB(255, 153, 0)
'Bananen sind gelb.
DatenserieFormatieren .SeriesCollection(2), xlRange, xlMarkerStyleDiamond, _
    5, RGB(255, 255, 0)
'Kiwis sind grün.
DatenserieFormatieren .SeriesCollection(3), xlRange, xlMarkerStyleSquare, _
    5, RGB(0, 128, 0)
End With
End Sub

Sub DatenserieFormatieren(DatenSerie As Excel.Series, xlRange As Excel.Range, _
    xlDatenPunktStil As Long, xlDatenPunktGrösse As Long, _
    colorDatenPunktFarbe)

    With DatenSerie
        .XValues = xlRange
        .MarkerStyle = xlDatenPunktStil
        .MarkerSize = xlDatenPunktGrösse
        .Border.Color = colorDatenPunktFarbe
        .MarkerBackgroundColor = colorDatenPunktFarbe
        .MarkerForegroundColor = colorDatenPunktFarbe
    End With
End Sub

Function TrimZelle(str)
    str = Mid(str, 1, Len(str) - 2)
    TrimZelle = str
End Function

Function FeldTypErmitteln(str) As Long
    If IsNumeric(str) Then
        FeldTypErmitteln = 3
    Else
        FeldTypErmitteln = 202
    End If
End Function

```



Die Beispieldatei *Bsp11\_02.doc* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap11*.

# MS Graph-Diagramme

Die Funktionalität hinter MS Graph ist die gleiche wie in Excel. Nur die Schnittstelle für die Daten ist anders, zudem bietet MS Graph keine Unterstützung der Farbenpalette. Da MS Graph seine eigene Datentabelle besitzt, muss Excel nicht auf dem Rechner vorhanden sein, um Diagramme in Word oder PowerPoint zu integrieren.

Die Programmierschnittstelle folgt der Benutzerschnittstelle, zu den erwähnten Abweichungen zum Excel-Objektmodell gesellt sich jedoch der Umgang mit grafischen Objekten. MS Graph bietet dafür keine vollständige Automatisierungsschnittstelle. Es ist beispielsweise möglich, ein Textfeld als Kommentar zu erstellen, nicht jedoch, dieses später anzusprechen, um es zu bearbeiten oder zu positionieren.

Der Makrorekorder ergibt folgende Grundsyntax für ein MS Graph-Diagramm (der Klassentyp ist MSGraph.Chart.8):

```
Selection.InlineShapes.AddOLEObject ClassType:="MSGraph.Chart.8", FileName:="", _
    LinkToFile:=False, DisplayAsIcon:=False
```

In diesem Beispiel erstellen wir das gleiche Diagramm wie für Excel, um die Unterschiede und Gemeinsamkeiten zwischen den beiden hervorzuheben. Den Beispielcode entnehmen Sie bitte dem Listing 11.7. Wo die gleichen Prozeduren in beiden Listings gebraucht werden, sind sie nur in Listing 11.6 aufgeführt.

Der erste Teil der Prozedur *DiagrammEinfuegen* ist identisch: die Word-Tabelle wird einer Objektvariablen zugewiesen und der Zielbereich darunter festgelegt. Das Diagramm-Objekt wird auf die gleiche Art eingefügt und aktiviert, nur der Klassentyp ist anders.

Da ein MS Graph-Datenblatt ADO-Recordsets nicht erkennt, werden die Zellinhalte der Word-Tabelle direkt in die Zellen des Datenblatts übernommen. Danach wird das Diagramm formatiert. Die Verbindung des Diagramms mit einem Datenbereich entfällt, weil ein MS Graph-Diagramm fest mit dem einen Datenblatt verbunden ist. Die erste Zeile und die erste Spalte liefern die Legenden- sowie X-Achsen-Einträge, die übrigen Zellen die Daten. Bitte beachten Sie die PlotBy-Eigenschaft des Application-Objekts. In Excel ist diese eine Eigenschaft des Chart-, nicht des Application-Objekts und daher für den erfahrenen Excel-Entwickler entsprechend schwierig zu finden.

Im Gegensatz zum Excel-Beispiel, bei dem das Diagramm-Objekt am Schluss des Codes noch aktiv ist, wird das MS Graph-Diagramm mit der Quit-Methode geschlossen. In MS Graph funktioniert sie auch bei In-place-Aktivierung!

**Listing 11.7** Ein MS Graph-Diagramm einfügen und formatieren

```
Sub DiagrammEinfuegen()
    Dim grphDiagramm As Graph.Chart
    Dim grphDatenblatt As Graph.DataSheet
    Dim grphApp As Graph.Application
    Dim doc As Word.Document
    Dim rng As Word.Range
    Dim ils As Word.InlineShape
    Dim oleF As Word.OLEFormat
    Dim tbl As Word.Table
    Dim cel As Word.Cell
    Dim lZaehler As Long
```

**Listing 11.7** Ein MS Graph-Diagramm einfügen und formatieren (Fortsetzung)

```

Dim lSpaltenZaehler As Long

Set doc = ActiveDocument
Set tbl = doc.Tables(1)
Set rng = tbl.Range
'Diagramm unter der Tabelle einfügen
rng.Collapse Direction:=wdCollapseEnd
rng.InsertAfter vbCrLf
rng.Collapse Direction:=wdCollapseEnd

'Das Diagramm im Dokument erstellen...
Set oleF = doc.InlineShapes.AddOLEObject(ClassType:="MSGraph.Chart.8", _
Range:=rng).OLEFormat
'...und aktivieren
oleF.DoVerb

'Der Arbeitsmappe, dem Datenblatt sowie Diagrammblatt Objektvariablen zuweisen
Set grphDiagramm = oleF.Object
Set grphApp = grphDiagramm.Application
Set grphDatenblatt = grphApp.DataSheet
'Die vorhandenen Daten löschen.
grphDatenblatt.Cells.Clear
'Die Spaltenüberschriften auf Basis der Word-Tabelle erstellen
lSpaltenZaehler = 0
For Each cel In tbl.Rows(1).Cells
    lSpaltenZaehler = lSpaltenZaehler + 1
    grphDatenblatt.Cells(1, lSpaltenZaehler).Value = TrimZelle(cel.Range.Text)
Next cel
'Die Daten, ab der 2. Tabellenblattzeile, einfügen
For lZaehler = 2 To tbl.Rows.Count
    For lSpaltenZaehler = 1 To tbl.Columns.Count
        grphDatenblatt.Cells(lZaehler, lSpaltenZaehler).Value = _
            TrimZelle(tbl.Rows(lZaehler).Cells(lSpaltenZaehler).Range.Text)
    Next lSpaltenZaehler
Next lZaehler

'Das Diagramm formatieren
DiagrammFormatieren grphDiagramm, grphDatenblatt

'Aufräumarbeiten
grphApp.Quit
Set grphDatenblatt = Nothing
Set grphApp = Nothing
Set grphDiagramm = Nothing
End Sub

Sub DiagrammFormatieren(grphDiagramm As Graph.Chart, grphDatenblatt As Graph.DataSheet)
    With grphDiagramm
        .Application.PlotBy = xlRows
        .ChartType = xlLine
        .Width = 450
        'Orangen sind orange.
        DatenserieFormatieren .SeriesCollection(1), xlMarkerStyleCircle, 5, RGB(255, 153, 0)
        'Bananen sind gelb.
        DatenserieFormatieren .SeriesCollection(2), xlMarkerStyleDiamond, 5, RGB(255, 255, 0)
        'Kiwis sind grün.
    End With
End Sub

```



Listing 11.7 Ein MS Graph-Diagramm einfügen und formatieren (Fortsetzung)

```

    DatenserieFormatieren .SeriesCollection(3), xlMarkerStyleSquare, 5, RGB(0, 128, 0)
End With
End Sub

Sub DatenserieFormatieren(DatenSerie As Variant, xlDatenPunktStil As Long,
    xlDatenPunktGrösse As Long, colorDatenPunktFarbe As Variant)

    With DatenSerie
        .MarkerStyle = xlDatenPunktStil
        .MarkerSize = xlDatenPunktGrösse
        .Border.Color = colorDatenPunktFarbe
        .MarkerBackgroundColor = colorDatenPunktFarbe
        .MarkerForegroundColor = colorDatenPunktFarbe
    End With
End Sub

```

Das Erstellen und Formatieren eines MS Graph-Diagramms unterscheidet sich kaum von der Handhabung eines Excel-Diagramms. Deshalb gilt der C#-Code in Listing 11.8 stellvertretend auch für Excel-Diagramme.

Listing 11.8 Ein MS Graph-Diagramm einfügen und formatieren (C#-Code)

```

private void DiagrammEinfuegen_CS()
{
    try
    {
        wd.Application wdApp = (wd.Application)
            wdMarshal.GetActiveObject("Word.Application");
        wd.Document doc = wdApp.ActiveDocument;
        wd.Table tbl = doc.Tables[1];
        wd.Range rng = tbl.Range;
        //Diagramm unter der Tabelle einfügen.
        object objCollapseEnd = wd.WdCollapseDirection.wdCollapseEnd;
        rng.Collapse(ref objCollapseEnd);
        rng.InsertAfter("\n");
        rng.Collapse(ref objCollapseEnd);

        //Das Diagramm im Dokument erstellen...
        object objClass = "MSGraph.Chart.8";
        object objMissing = System.Reflection.Missing.Value;
        object objFalse = false;
        object objRange = (object)rng;
        wd.OLEFormat oleF = doc.InlineShapes.AddOLEObject(ref objClass, ref objMissing,
            ref objFalse, ref objMissing, ref objMissing, ref objMissing,
            ref objRange).OLEFormat;
        //...und aktivieren
        oleF.DoVerb(ref objMissing);

        //Der Arbeitsmappe, dem Datenblatt sowie Diagrammblatt Objektvariablen zuweisen
        grph.Chart grphDiagramm = (grph.Chart) oleF.Object;
        grph.Application grphApp = grphDiagramm.Application;
        grph.DataSheet grphDatenblatt = grphApp.DataSheet;
        //Die vorhandenen Daten löschen.
        grphDatenblatt.Cells.Clear();
    }
}

```

**Listing 11.8** Ein MS Graph-Diagramm einfügen und formatieren (C#-Code) *(Fortsetzung)*

```
//Die Spaltenüberschriften auf Basis der Word-Tabelle erstellen
int spaltenZaehler = 0;
foreach (wd.Cell cel in tbl.Rows[1].Cells)
{
    spaltenZaehler++;
    grph.Range rngZelle = (grph.Range) grphDatenblatt.Cells[1, spaltenZaehler];
    rngZelle.set_Value(objMissing, TrimZelle(cel.Range.Text));
}
//Die Daten, ab der 2. Tabellenblattzeile, einfügen
for (int zaehler = 2; zaehler <= tbl.Rows.Count; zaehler++)
{
    for (spaltenZaehler = 1; spaltenZaehler <= tbl.Columns.Count; spaltenZaehler++)
    {
        grph.Range rngZelle = (grph.Range) grphDatenblatt.Cells[zaehler, spaltenZaehler];
        rngZelle.set_Value(objMissing,
            TrimZelle(tbl.Rows[zaehler].Cells[spaltenZaehler].Range.Text));
    }
}

//Das Diagramm formatieren
DiagrammFormatieren(grphDiagramm, grphDatenblatt);

//Aufräumarbeiten
grphApp.Quit();
grphDatenblatt = null;
grphApp = null;
grphDiagramm = null;
wdMarshal.ReleaseComObject(wdApp);
wdApp = null;
}
catch (System.Exception ex)
{
    MessageBox.Show("Fehlermeldung: " + ex.Message + " in " + ex.Source);
}
}

private void DiagrammFormatieren(grph.Chart grphDiagramm, grph.DataSheet grphDatenblatt)
{
    grphDiagramm.Application.PlotBy = grph.XlRowCol.xlRows;
    grphDiagramm.ChartType = grph.XlChartType.xlLine;
    grphDiagramm.Width = 450;
    //Orangen sind orange.
    grph.Series datenSerie = (grph.Series) grphDiagramm.SeriesCollection(1);
    DatenserieFormatieren(datenSerie,
        grph.XlMarkerStyle.xlMarkerStyleCircle, 5, System.Drawing.Color.MediumPurple);
    // .RGB(255, 153, 0));
    //Bananen sind gelb.
    datenSerie = (grph.Series) grphDiagramm.SeriesCollection(2);
    DatenserieFormatieren(datenSerie,
        grph.XlMarkerStyle.xlMarkerStyleDiamond, 5, System.Drawing.Color.Aqua);
    // RGB(255, 255, 0)
    //Kiwis sind grün.
    datenSerie = (grph.Series) grphDiagramm.SeriesCollection(3);
    DatenserieFormatieren(datenSerie,
        grph.XlMarkerStyle.xlMarkerStyleSquare, 5, System.Drawing.Color.Green);
}
```

Listing 11.8 Ein MS Graph-Diagramm einfügen und formatieren (C#-Code) (Fortsetzung)

```

        // RGB(0, 128, 0)
    }

    private void DatenserieFormatieren(grph.Series DatenSerie, grph.XlMarkerStyle
        xlDatenPunktStil, int xlDatenPunktGrösse, System.Drawing.Color colorDatenPunktFarbe)
    {
        DatenSerie.MarkerStyle = xlDatenPunktStil;
        DatenSerie.MarkerSize = xlDatenPunktGrösse;
        DatenSerie.Border.Color = (int) colorDatenPunktFarbe.ToArgb();
        DatenSerie.MarkerBackgroundColor = (int) colorDatenPunktFarbe.ToArgb();
        DatenSerie.MarkerForegroundColor = (int) colorDatenPunktFarbe.ToArgb();
    }

    private string TrimZelle(string s)
    {
        s = s.Substring(0, s.Length - 2);
        return s;
    }

```



Die Beispieldatei *Bsp11\_03.doc* finden Sie auf der CD-ROM zum Buch im Ordner `\Beispiele\Kap11`.

## WordArt



Im Lieferumfang von Microsoft Office sind mehrere so genannte »Applets« enthalten, die zusätzliche Funktionalität in Form von grafischen Objekten anbieten. Da diese meist nicht als eigenständige Anwendungen betrieben werden können und auch nicht ausdrücklich als OLE-Objekte eingebettet werden, ist auf den ersten Blick nicht erkennbar, dass es sich hier ebenfalls um OLE-Objekte handelt, genauer genommen um ActiveX-Steuerelemente. Hierfür dienen der Formel-Editor und WordArt als Beispiele.

### HINWEIS

Während WordArt eine Automatisierungsschnittstelle zur Verfügung stellt, ist es nicht möglich, den Formel-Editor fernzusteuern. Dies, weil Microsoft den Formel-Editor von einer anderen Firma – MathType – lizenziert hat.

Statt WordArt als eine selbstständige Objektbibliothek zu führen, die allenfalls in das VB-Projekt ausdrücklich eingebunden werden müsste, steht die Automatisierungsschnittstelle als Teil der Office-Zeichnungswerkzeuge zur Verfügung. Aus der Diskussion in Kapitel 6 geht hervor, dass jede Office-Anwendung über die Zeichnungsobjekte verfügt. Ein WordArt-Objekt in PowerPoint oder Excel wird auf gleiche Weise wie in Word erstellt und bearbeitet.

Der Makrorekorder liefert beim Einfügen eines WordArt-Objekts die folgende Codezeile, woraus zu erkennen ist, dass ein WordArt-Objekt durch die Methode `AddTextEffect` des `Shape`-Objekts erstellt wird.

```

ActiveDocument.Shapes.AddTextEffect(msoTextEffect2, "Ihr Text", "Arial Black", 36#, _
    msoFalse, msoFalse, 212.5, 110.4).Select

```

Die Syntax der Methode lautet:

```
AddTextEffect(PresetTextEffect, Text, FontName, FontSize, FontBold, FontItalic, Left, Top, [Anchor])
```

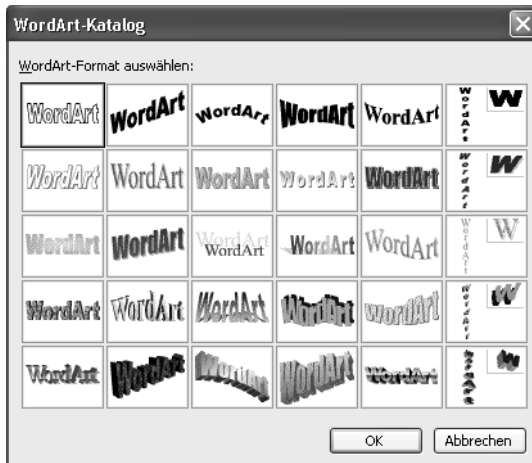
#### HINWEIS

Bekanntlich kann in Word 2002 und 2003 unter *Extras/Optionen* auf der Registerkarte *Bearbeiten* mit dem Eintrag *Bild einfügen als* festgelegt werden, mit welcher Textflussformatierung ein grafisches Objekt standardmäßig eingefügt wird. Wird hierfür *Mit Text in Zeile* ausgewählt, wandelt Word das eingefügte WordArt-Objekt in ein *InlineShape* um. Diese Umwandlung wird vom Makrorekorder nicht berücksichtigt.

Alle Argumente außer *Anchor* sind erforderlich. Das erste Argument – *PresetTextEffect* – bestimmt die Grundform, die im ersten Schritt in der Benutzerschnittstelle ausgewählt wird (siehe Abbildung 11.7). Zudem werden Schriftart, -größe und -schnitt festgelegt sowie die Positionierung des Objekts relativ zum Verankerungsbereich. Wird kein Verankerungsbereich angegeben, erfolgt die Positionierung relativ zur linken, oberen Seitenecke.

*PresetTextEffect* erwartet einen *MsoPreSetTextEffect*-Konstantwert. Dieser ist von 0 bis 29 durchnummeriert, beschreibt den Effekt also nicht. Er entspricht jedoch den Abbildungen im Dialogfeld *WordArt-Katalog*, reihenweise von links oben nach rechts unten.

Abbildg. 11.7 Im ersten Schritt wird die Grundform des WordArt-Objekts aus dem *WordArt-Katalog* gewählt.

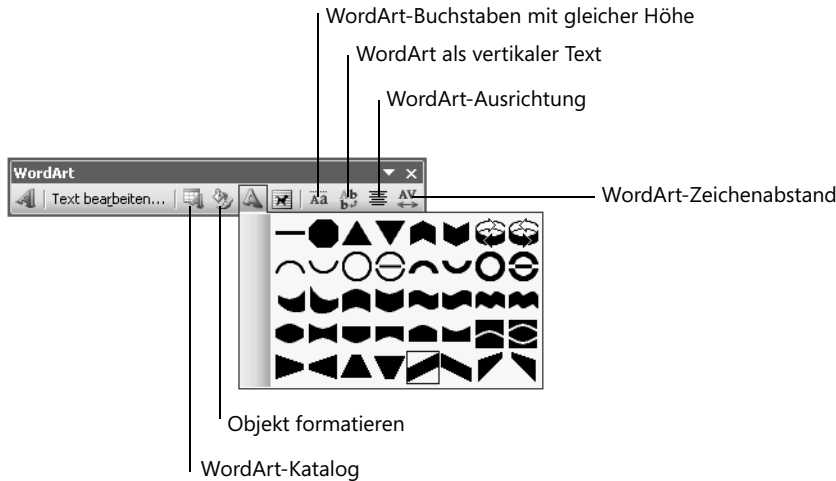


Diese Grundformen sind vordefinierte Zusammenstellungen der erweiterten Formatierungsmöglichkeiten, die die Zeichnungs- und WordArt-Symbolleisten (Abbildung 11.8) bereitstellen. Farben, Linien und Schattierungen werden mit den üblichen grafischen Werkzeugen geregelt. Die Lauflinie des Textes wird bestimmt über die Auswahl im Dropdownfeld *WordArt-Form*. Das Aussehen kann also beliebig angepasst werden.

Im Objektmodell entsprechen die Formen im Dropdownfeld der *PresetShape* Eigenschaft des *Shape*-Objekts. Sie erwartet einen von 40 *MsoPresetTextEffectShape*-Konstantwerten. Die Konstantwertbezeichnungen sind in diesem Fall beschreibend und in der Hilfe sowie dem Objektkatalog aufgelistet.

Die numerischen Werte entsprechen auch hier reihenweise den Abbildungen des Dropdownfeldes von oben links nach unten rechts.

**Abbildg. 11.8** Mit den allgemeinen Zeichnungs- sowie den WordArt-spezifischen Werkzeugen werden WordArt-Objekte angepasst.



Einen Überblick der vordefinierten Grundformen sowie des Dropdownfeld-Inhalts bietet das Resultat aus Listing 11.9. Es wird durch alle Konstantwerte geschleift und für jeden ein WordArt-Objekt ins Dokument gefügt. Bitte beachten Sie, dass die Ausführung der Prozedur sehr lange dauern kann. Anschließend erscheinen die *PresetTextEffect*-Beispiele links, die *PresetTextEffectShape*-Beispiele rechts untereinander aufgelistet (siehe Abbildung 11.9).

**Abbildg. 11.9** Wirkung der *MsoPresetTextEffect*- und *MsoPresetTextEffectShape*-Konstantwerte



**Listing 11.9** Alle Variationen aus *WordArt-Form* in ein Dokument einfügen

```

Sub AlleWordArtTextEffectsAuflisten()
    Dim lEffect As Long
    Dim lEffectShape As Long
    Dim shp As Word.Shape
    Dim doc As Word.Document
    Dim rng As Word.Range

    Set doc = Documents.Add
    Set rng = doc.Content
    rng.ParagraphFormat.LineSpacingRule = wdLineSpaceAtLeast
    rng.ParagraphFormat.LineSpacing = 50
    For lEffect = 0 To 29
        Set rng = doc.Paragraphs(lEffect + 1).Range
        doc.Shapes.AddTextEffect lEffect, "Effect" & CStr(lEffect), "Arial", _
            20, msoFalse, msoFalse, 0, 0, rng
        rng.InsertAfter vbCr
    Next

    'Dafür sorgen, dass genügend Absätze für alle MsoPresetTextEffectShape
    'Variationen vorhanden sind.
    rng.InsertAfter String(40 - doc.Paragraphs.Count, vbCr)
    For lEffectShape = 1 To 40
        Set rng = doc.Paragraphs(lEffectShape).Range
        Set shp = doc.Shapes.AddTextEffect(7, "Shape Effect" & CStr(lEffectShape), _
            "Arial", 20, msoFalse, msoFalse, 200, 0, rng)
        shp.TextEffect.PresetShape = lEffectShape
    Next
End Sub

```

Im folgenden Listing beachten Sie bitte, wie C# auf `Microsoft.Office.Core` verweist, um mit den WordArt-Eigenschaften zu arbeiten. Bemerkenswert ist auch, wie der Ganzzahlwert einer Enumeration in den Enumerationswert konvertiert werden muss, bevor er mit `AddTextEffect` gebraucht werden kann.

**Listing 11.10** Die C#-Version für WordArt

```

private void AlleWordArtTextEffectsAuflisten_CS()
{
    Microsoft.Office.Core.MsoTriState offFalse =
        Microsoft.Office.Core.MsoTriState.msoFalse;
    wd.Application wdApp = (wd.Application) wdMarshal.GetActiveObject("Word.Application");
    wd.Document doc = wdApp.ActiveDocument;
    wd.Range rng = doc.Content;
    rng.ParagraphFormat.LineSpacingRule = wd.WdLineSpacing.wdLineSpaceAtLeast;
    rng.ParagraphFormat.LineSpacing = 50f;
    for (int effect = 0; effect <= 29; effect++)
    {
        rng = doc.Paragraphs[effect + 1].Range;
        object objRange = rng;
        Microsoft.Office.Core.MsoPresetTextEffect objEffect =
            (Microsoft.Office.Core.MsoPresetTextEffect) effect;
        doc.Shapes.AddTextEffect(objEffect, "Effect" + effect.ToString(), "Arial", 20,
            offFalse, offFalse, 0, 0, ref objRange);
        rng.InsertAfter("\n");
    }
}

```

Listing 11.10 Die C#-Version für WordArt (Fortsetzung)

```

    }

    //Dafür sorgen, dass genügend Absätze für alle MsoPresetTextEffectShape-
    //Variationen vorhanden sind.
    string newLine = "\n";
    int i = 40 - doc.Paragraphs.Count;
    char[] cNewLine = newLine.ToCharArray();
    char c = cNewLine[0];
    string s = new String(c, i);
    rng.InsertAfter(s);
    for (int effectShape = 1; effectShape <= 40; effectShape++)
    {
        rng = doc.Paragraphs[effectShape].Range;
        object objRange = rng;
        Microsoft.Office.Core.MsoPresetTextEffect effect7 =
            Microsoft.Office.Core.MsoPresetTextEffect.msoTextEffect7;
        wd.Shape shp = doc.Shapes.AddTextEffect(effect7,
            "Shape Effect" + effectShape.ToString(),
            "Arial", 20, offFalse, offFalse, 200, 0, ref objRange);
        shp.TextEffect.PresetShape =
            (Microsoft.Office.Core.MsoPresetTextEffectShape) effectShape;
    }
}

```

Mit einer Ausnahme entsprechen die weiteren Eigenschaften des TextEffect-Objekts den Schaltflächen der Symbolleiste: Alignment (Ausrichtung), FontBold (Fett), FontItalic (Kursiv), FontName (Schriftart), FontSize (Schriftgröße), KernedPairs (Zeichenpaarkerning), NormalizedHeight (Höhe der Kleinbuchstaben), Text, Tracking (Zeichenabstand). Zudem gibt es die ToggleVerticalText-Methode, die die Buchstaben unter-, anstatt nebeneinander anordnet (oder umgekehrt).



Für die Eigenschaft RotatedChars (Zeichendrehung) gibt es in der Benutzeroberfläche kein Gegenstück mehr. Sie können es jedoch mit der Prozedur in Listing 11.11 anbieten.

Listing 11.11 Die Zeichen in einem WordArt-Objekt um 90° drehen

```

Sub WordArtZeichenDrehen()
    Dim sel As Word.Selection
    Dim shp As Word.Shape
    Dim ils As Word.InlineShape
    Set sel = Selection
    If sel.Type = wdSelectionShape Then
        Set shp = sel.ShapeRange(1)
        If shp.Type = msoTextEffect Then
            shp.TextEffect.RotatedChars = msoTriStateToggle
        End If
    End If

    If Selection.Type = wdSelectionInlineShape Then
        Set ils = sel.InlineShapes(1)
        If ils.Type = 3 Then
            'Nicht alle grafischen Objekte des Typs 3 sind WordArt.
            On Error Resume Next
            ils.TextEffect.RotatedChars = msoTriStateToggle
            On Error GoTo 0
        End If
    End If
End Sub

```

**Listing 11.11** Die Zeichen in einem WordArt-Objekt um 90° drehen (*Fortsetzung*)

```
End If
End If
End Sub
```



Die Beispieldatei *Bsp11\_04.doc* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap11*.

**HINWEIS**

Ein weiteres Code-Beispiel für die Automatisierung von WordArt finden Sie in Kapitel 27.

## ActiveX-Steuerelemente

In Kapitel 6 wurde bereits die in Word 6.0 eingeführte Formular-Funktionalität vorgestellt. Die drei darin zur Verfügung stehenden Formularfeldtypen – Textfeld, Kontrollkästchen und Dropdownliste – sind nützlich, jedoch beschränkt im Vergleich zu der Vielzahl von Steuerelementen, die heute in Gebrauch sind. Für Office 97 wurden ActiveX-Steuerelemente für UserForms entwickelt, die auch in Office-Dokumenten eingebettet werden können. Sie erweitern nicht nur die Palette der Steuerelemente, sondern bieten über eine eigene VBA-Schnittstelle mit Eigenschaften und Ereignissen interessante Möglichkeiten für die Interaktion mit dem Anwender.

**WICHTIG**

Weil ActiveX-Steuerelemente über eine VBA-Schnittstelle verfügen, lösen sie die Makrosicherheitswarnung aus (mehr darüber steht in Kapitel 1 beschrieben).

Wie andere Objekte auch, werden ActiveX-Steuerelemente von Word als grafische Objekte betrachtet. Entsprechend können sie mit Textfluss (Shapes-Auflistung) versehen oder *mit Text in Zeile* (InlineShapes-Auflistung) positioniert sein. ActiveX-Steuerelemente lassen sich in Word mit oder ohne Formularschutz einsetzen, müssen bei aktivem Formularschutz aber *mit Text in Zeile* stehen, um zugänglich zu sein.

Da ActiveX-Steuerelemente nicht explizit für die Word-Dokumentumgebung entwickelt wurden, verhalten sie sich nicht immer optimal. Hier die wichtigsten Punkte, worauf zu achten ist:

- Sie werden von der Macintosh Word-Version nicht unterstützt.
- Word kann sie nicht immer korrekt anzeigen. Vor allem beim Scrollen scheinen sie »in der Luft« zu hängen und bewegen sich mit dem Text erst, wenn das Bildlauffeld wieder losgelassen wird.
- Word-Tastaturkombinationen werden innerhalb eines ActiveX-Steuerelements nicht unterstützt. Wie bei einem Excel-Tabellenobjekt ist das »Innere« eines ActiveX-Steuerelements in Wirklichkeit eine andere Anwendungsumgebung.
- Text in ActiveX-Steuerelementen kann nicht formatiert werden, auch nicht mit Makrounterstützung.
- Im Gegensatz zu einem Formular-Textfeld kann sich ein ActiveX-Steuerelement dem Textfluss nicht anpassen oder als Teil des Textes integriert werden. Ein Steuerelement ist immer ein viereckiger Block.



- ActiveX-Steuerelemente, die sich automatisch dem Inhalt anpassen, erkennen die Dokumentränder nicht und wachsen darüber hinaus.
- Die Seitenansicht ist gesperrt, wenn die Einfügemarke in einem ActiveX-Steuerelement steht. Sind ActiveX-Steuerelemente die einzig zugängigen Stellen in einem Formular, steht die Seitenansicht für dieses Dokument nicht zur Verfügung.
- Das Risiko der Dokumentbeschädigung ist höher, wenn sich ActiveX-Steuerelemente in einem Dokument befinden.
- Inhalte und Einstellungen von ActiveX-Steuerelementen sind in einem als XML, RTF oder HTML gespeicherten Dokument als binärer Code enthalten und daher nicht zugänglich.

Die mit Office gelieferten ActiveX-Steuerelemente befinden sich in der Symbolleiste *Steuerelement-Toolbox* (siehe Abbildung 11.10). Über die letzte Symbolleisten-Schaltfläche *Weitere Steuerelemente* können zusätzliche ActiveX-Steuerelemente in Word verwendet werden. Diese sind jedoch in einem geschützten Formular möglicherweise nicht nutzbar, und es ist nicht garantiert, dass eine mögliche Automatisierungsschnittstelle zur Verfügung steht.

Abbildg. 11.10 Die den Office-Anwendungen zur Verfügung stehenden ActiveX-Steuerelemente



Die erste Schaltfläche der Symbolleiste steuert den Entwurfs-Modus. Werden Steuerelemente eingefügt, verschoben oder deren Eigenschaften geändert, muss dieser Modus aktiv sein. Er ist dagegen auszuschalten, wenn die ActiveX-Steuerelemente als Eingabefelder benutzt werden sollen.

Ein ActiveX-Steuerelement wird von einem Word-Dokument als Teil seiner selbst betrachtet und erscheint somit in der IntelliSense-Liste der Dokument-Eigenschaften, wenn Sie ActiveDocument in einem Code-Fenster eintippen. Die Liste der Ereignisse befindet sich deshalb im Modul *ThisDocument*. Der Einsatz von ActiveX-Steuerelementen in einem Word-Dokument stellt den Entwickler vor einige Herausforderungen.

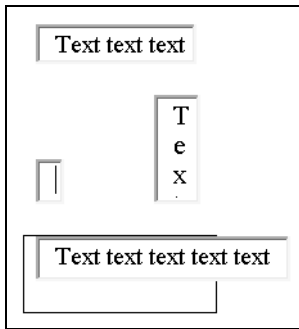



Einpas-  
sung in  
den Text

Ein Textfeld-Steuerelement kann eine genaue Größe haben oder sich dem Inhalt anpassen. Ferner ist es möglich, den Inhalt auf eine einzige Zeile zu begrenzen oder mehrere Zeilen zuzulassen. Solche Einstellungen werden im Eigenschaftenfenster (das gleiche, das auch in einem VBA-Projekt zur Verfügung steht) vorgenommen. Die Eigenschaften *Height* und *Width* legen die Höhe und Breite des Textfeldes fest. Um mehrere Zeilen und/oder eine automatische Größenanpassung zu erlauben, muss *Multiline* bzw. *AutoSize* entsprechend auf *True* gesetzt werden.

Das alles erscheint relativ einfach und logisch; erst in der Ausführung offenbart sich die Falle. Sind *Multiline* sowie *AutoSize* beide auf *True* gesetzt, wird die automatische Größenanpassung unberechenbar und vom Anwenderstandpunkt aus gesehen unkontrollierbar. Die Abbildung 11.11 veranschaulicht das Problem. Oben ist ein statisches Textfeld dargestellt, darunter links ein Textfeld, das in beide Richtungen eine Größenanpassung erlaubt, und daneben das gleiche Feld nach der Eingabe von Text.

Für dieses Problem gibt es keine 100-prozentig zufrieden stellende Lösung, zumal Textfelder die normalen Textbegrenzungen wie Dokumentränder und Tabellenzellen nicht respektieren, wie die unterste Variante in Abbildung 11.11 zeigt.

**Abbildg. 11.11** Begrenzungen von ActiveX-Textfeldern in einem Word-Dokument



Automatisierungscode kann hier etwas Abhilfe schaffen. Im folgenden Beispiel wird von einer statischen Breite ausgegangen (`AutoSize = False`). Lediglich die Höhe wird der eingegebenen Zeilenanzahl angepasst, wobei eine Zeile durch Drücken der -Taste definiert ist. Das Listing 11.12 zeigt, wie die Anzahl der Zeilen ermittelt und mit der Schriftgröße multipliziert wird, um die neue Steuerelementhöhe zu berechnen. Eine brauchbare Zeilenhöhe ergibt sich, wenn die Schriftgröße um einen Faktor erhöht wird, der für die meisten Schriftarten bei etwa 1,2 liegt.

**Listing 11.12** Die Höhe eines Textfeldes der Zeilenanzahl im Feld anpassen

```
Public Function AdjustControlHeight(o_ctl As Object)
    Dim sSchrifthöhe As Single
    Dim strEintrag As String
    Dim lZaehler As Long
    Dim lPos As Long

    o_ctl.SelStart = 0
    sSchrifthöhe = o_ctl.FontSize + (0.2 * o_ctl.FontSize)
    strEintrag = o_ctl.Text
    lZaehler = 1
    For lZaehler = 1 To Len(strEintrag)
        If Mid(strEintrag, lZaehler, 1) = vbCr Then
            lZeilenZaehler = lZeilenZaehler + 1
        End If
    Next
    o_ctl.Height = (lZaehler * sSchrifthöhe) + 1
End Function
```

## Das Navigieren

Stehen die ActiveX-Steuerelemente *mit Text in Zeile* und das Dokument ist als Formular geschützt, kann mit der -Taste wie bei Formularfeldern von einem Element zum nächsten gesprungen werden. Ansonsten lässt sich die Navigation nur begrenzt steuern.

Obwohl es sich um die gleichen ActiveX-Steuerelemente handelt, die in UserForms gebraucht werden, stehen nicht die gleichen Eigenschaften und Ereignisse zur Verfügung. ActiveX-Steuerelemente im Dokument verfügen beispielsweise über keine `TabIndex`- oder `TabStop`-Eigenschaft. Ihnen fehlen

auch die `SetFocus`-Methode sowie die `AfterUpdate`- und `BeforeUpdate`-Ereignisse. Alle diese stammen aus der `UserForm`.

Die einzigen Ereignisse, die in einem Dokument beim Navigieren helfen können, sind `GotFocus` (die Einfügemarke ist in das Steuerelement eingetreten) und `LostFocus` (die Einfügemarke hat das Steuerelement verlassen). Mit diesen kann – ähnlich wie mit den Formularfeld-Eigenschaften `OnExit` und `OnEnter` – festgestellt werden, welches Feld verlassen und welches gerade das aktuelle geworden ist, und entsprechend gehandelt werden.

Um das Prinzip zu veranschaulichen, bleiben wir beim Beispiel der Höhenanpassung. Das Listing 11.12 ist eine Funktion, die ein Steuerelement-Objekt als Argument hat. Woher kommt dieses Objekt? Eine andere Prozedur muss die Funktion aufrufen und ihr das Steuerelement übergeben. In Listing 11.13 finden Sie eine `LostFocus`-Prozedur für das Steuerelement `xTextBox1` und eine `GotFocus`-Prozedur für das Steuerelement `xTextBox2`.

Beim Verlassen von `xTextBox1` und Eintreten in `xTextBox2` wird ein `LostFocus`-Ereignis ausgelöst und die Prozedur ausgeführt. Zuerst wird getestet, ob die Modulebenen-Variable `b_LOSTFOCUS` falsch ist. Ist das der Fall, wird die Funktion `AdjustControlHeight` aufgerufen und ihr dieses Steuerelement übergeben. Die Wirkungsweise der Funktion ist uns schon bekannt, aber hinzugefügt wurde die Codezeile `b_LOSTFOCUS = True`. Damit wird beim folgenden, erwarteten Verlassen von `xTextBox1` diese Funktion nicht mehr aufgerufen.

Parallel dazu wurde beim Eintreten in das Feld `xTextBox2` sein `GotFocus`-Ereignis ausgelöst. Dies weist der Modulebenen-Variable `o_GotFocus` dieses Steuerelement zu.

Nach Anpassung der Steuerelementhöhe wird `xTextBox1` nochmals angesprungen, um die Markierung an den Feldanfang zu setzen. Danach wird `xTextBox2` erneut angesprungen. Diese Handlung löst nochmals das `LostFocus`-Ereignis der `xTextBox1` aus. Weil aber `b_LostFocus` noch `True` ist, wird `AdjustControlHeight` nicht nochmals ausgeführt. Am Schluss wird `b_LostFocus` wieder auf `False` gesetzt.

#### ACHTUNG

Beim Einsatz von `LostFocus` und `GotFocus` für mehrere Steuerelemente müssen Sie sehr sorgfältig vorgehen. Sie werden nicht eines nach dem anderen ausgelöst und ausgeführt, weshalb die Gefahr einer endlosen Schleife besteht.

**Listing 11.13** Navigieren der Steuerelemente mit Hilfe der `LostFocus`- und `GotFocus`-Ereignisse

```
Private b_LOSTFOCUS As Boolean
Private o_GotFocus As Object

Private Sub xTextBox1_LostFocus()
    If Not b_LOSTFOCUS Then
        AdjustControlHeight xTextBox1
        xTextBox1.Select
        xTextBox1.SelStart = 0
        o_GotFocus.Select
    End If
    b_LOSTFOCUS = False
End Sub

Private Sub xTextBox2_GotFocus()
    Set o_GotFocus = xTextBox2
End Sub
```

**Listing 11.13** Navigieren der Steuerelemente mit Hilfe der *LostFocus*- und *GetFocus*-Ereignisse (Fortsetzung)

```
Public Function AdjustControlHeight(o_ctl As Object)
    Dim sSchrifthöhe As Single
    Dim strEintrag As String
    Dim lZaehler As Long
    Dim lZeilenZaehler

    b_LOSTFOCUS = True
    o_ctl.SelStart = 0
    sSchrifthöhe = o_ctl.FontSize + (0.2 * o_ctl.FontSize)
    strEintrag = o_ctl.Text
    lZeilenZaehler = 1
    For lZaehler = 1 To Len(strEintrag)
        If Mid(strEintrag, lZaehler, 1) = vbCr Then
            lZeilenZaehler = lZeilenZaehler + 1
        End If
    Next
    o_ctl.Height = (lZeilenZaehler * sSchrifthöhe) + 1
End Function
```



Die Beispieldatei *Bsp11\_05.doc* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap11*.

## Datengültigkeit prüfen

Das Prinzip für das Navigieren unterliegt dem Vorgang für die Prüfung der Datengültigkeit in einem Formular, das auf ActiveX-Steuerelementen basiert. In Abbildung 11.12 ist ein Beispiel dargestellt, das sich vom Aussehen her zwar nicht von einem Formular mit Formularfeldern (in Kapitel 6 vorgestellt) unterscheidet, jedoch anders verhält. Die Eingabefelder für die Adresse haben beispielsweise eine genaue Breite. Kein Feld kann über die Zeile umbrechen. Die in eckigen Klammern stehenden Eingabeaufforderungen resultieren aus dem Textinhalt von ActiveX-Steuerelementen, die ohne Rahmen formatiert sind. Beim Erstellen eines neuen Dokuments von der Vorlage wird die Prozedur *Document\_New* in Listing 11.14 automatisch ausgeführt. Die Modulebene-Variablen werden belegt und das erste Feld markiert, so dass der Benutzer mit der Eingabe sofort beginnen kann.

**ACHTUNG** Es ist nicht gewährleistet, dass *Document\_New* als erste Prozedur ausgeführt wird. Es ist möglich, dass das *GotFocus*-Ereignis eines Steuerelements vorher ausgelöst wird. Falls in *Document\_New* wie hier globale Variablen initialisiert werden, muss der Stand dieser in *GotFocus* überprüft werden.

Durch Drücken der -Taste oder Klicken mit der Maus wird zum nächsten Feld gesprungen. Der Inhalt des gerade verlassenen Feldes wird zu diesem Zeitpunkt geprüft, und der Benutzer darf, falls der Inhalt nicht gültig ist, seine Arbeit nicht fortsetzen. Nach der Anrede steht ein fortlaufender Abschnittswechsel; der folgende Abschnitt ist nicht geschützt. Dort kann der Benutzer den Brieftext frei redigieren.

Abbildg. 11.12 Ein Formular, das mit ActiveX-Steuerelementen erstellt wurde

Nordwind GmbH  
München

München, den 17. September 2005

[Empfänger eingeben]  
[Adresse eingeben]  
[PLZ und Ort eingeben]  
[Land eingeben oder löschen]

[Betreff eingeben]

Sehr geehrt [Anrede eingeben]

|

Mit freundlichen Grüßen,

In diesem Beispiel wird lediglich geprüft, ob ein Feld einen Eintrag hat. Nur die Angabe für das Bestimmungsland darf fehlen, alle anderen sind erforderlich.

Für jedes ActiveX-Steuerelement befindet sich im *ThisDocument*-Modul der Vorlage eine *GetFocus*- sowie *LostFocus*-Prozedur. Diese sind für nur zwei Felder in Listing 11.14 angegeben: *txtEmpfänger* sowie *txtLand*.

*txtEmpfänger* erhält beim Anlegen oder Öffnen des Dokuments den Fokus. Erfahrungsgemäß wird dessen *GotFocus*-Ereignis vor dem *Document\_New*-Ereignis ausgelöst. Deshalb wird geprüft, ob die globale Variable *o\_CTL* schon initialisiert wurde. Wenn nicht, wird die Prozedur abgebrochen.

Beim Verlassen dieses Feldes wird das *LostFocus*-Ereignis ausgeführt. Diese Prozedur ruft ihrerseits *SetDataValidation* auf, die die Datenprüfung ausführt. In diesem Beispiel prüft sie lediglich, ob das gerade verlassene Feld Text beinhaltet. Wenn ja, wird die boolesche Variable *bDATENGÜLTIG* auf *True* gesetzt, sonst auf *False*.

Parallel dazu wird das *GotFocus*-Ereignis des Feldes ausgeführt, das der Benutzer als Nächstes gewählt hat. Das ActiveX-Steuerelement sowie das Dokument werden an die Prozedur *ValidateData* übergeben, die die globale Variable *bDATENGÜLTIG* prüft. Falls sie den Wert *True* hat, wird die globale Variable *o\_CTL* diesem Steuerelement gleich gesetzt und markiert. Sonst wird *o\_CTL* (das gerade verlassene Feld) wieder angesprungen.

Da ein Eintrag in *txtLand* fakultativ ist, wird in seiner *LostFocus*-Prozedur *bDATENGÜLTIG* einfach auf *True* gesetzt, ohne den Feldinhalt zu prüfen.

**Listing 11.14** Datengültigkeit eines Formulars mit ActiveX-Steuerelementen prüfen

```
'Code im Modul "ThisDocument"
Private Sub Document_New()
    Dim doc As Word.Document

    Set doc = ActiveDocument
    Set o_CTL = doc.InlineShapes(1).OLEFormat.Object
    bDATENGUELTIG = False
    o_CTL.Select
    SelectControlContent o_CTL
    With doc.ActiveWindow.View
        .ShowHiddenText = True
        .ShowAll = False
    End With
End Sub

Private Sub txtEmpfänger_GotFocus()
    Dim doc As Word.Document
    Dim o_Control As Object

    Set doc = ActiveDocument
    'Document_New wurde noch nicht ausgeführt
    If o_CTL Is Nothing Then Exit Sub
    Set o_Control = Selection.InlineShapes(1).OLEFormat.Object
    ValidateData doc, o_Control
End Sub

Private Sub txtEmpfänger_LostFocus()
    Dim doc As Word.Document

    Set doc = ActiveDocument
    SetDataValidation doc, doc.txtEmpfänger
End Sub

Private Sub txtLand_GotFocus()
    Dim doc As Word.Document
    Dim o_Control As Object

    Set doc = ActiveDocument
    Set o_Control = Selection.InlineShapes(1).OLEFormat.Object
    ValidateData doc, o_Control
End Sub

Private Sub txtLand_LostFocus()
    'Dieses Feld darf leer sein
    bDATENGUELTIG = True
End Sub

'Code im normalen Modul
Public bDATENGUELTIG As Boolean
Public o_CTL As Object

Public Function ValidateData(doc As Word.Document, o_Control As Object)
    If bDATENGUELTIG = True Then
        Set o_CTL = o_Control
        SelectControlContent o_Control
    Else
```

Listing 11.14 Datengültigkeit eines Formulars mit ActiveX-Steuerelementen prüfen (Fortsetzung)

```

    o_CTL.Select
End If
End Function

Public Function SetDataValidation(doc As Word.Document, o_Control As Object)
    If Len(o_Control.Text) = 0 Then
        bDATENGUELTIG = False
    Else
        bDATENGUELTIG = True
    End If
End Function

Public Function SelectControlContent(ctl As Object)
    Dim lTextLen As Long

    lTextLen = Len(ctl.Text)
    ctl.SelectStart = 0
    ctl.SelectLength = lTextLen
End Function

```

## Steuerelement-Referenzen in VBA-Code

Es wird Ihnen aufgefallen sein, dass wir in den *GotFocus*- und *LostFocus*-Prozeduren die Steuerelemente auf zwei verschiedene Weisen angesprochen haben, über das *InlineShape*-Objekt:

```
Set o_Control = Selection.InlineShapes(1).OLEFormat.Object
```

sowie über den Objekt-Namen:

```
SetDataValidation doc, doc.txtEmpfänger
```

Es handelt sich hier um Late vs. Early Binding. Die erste Variation kann für alle Steuerelemente kopiert und benutzt werden, ohne die Codezeile anpassen zu müssen. Dafür ist nicht auf einen Blick ersichtlich, um welches Objekt es eigentlich geht. Die zweite Variation ist aufwendiger, dafür selbst dokumentierend und in der Ausführung wahrscheinlich ein klein wenig schneller.

Wie schon erwähnt, integriert Word ein ActiveX-Steuerelement in das Dokument-Objekt. Wenn Sie im *ThisDocument*-Modul arbeiten, erscheint es in der IntelliSense-Liste für *ActiveDocument*, *ThisDocument* und *Me*. Der VB-Editor erkennt sogar den Steuerelementnamen allein. Das Verhalten ist analog dem, wenn in einer UserForm gearbeitet wird. Dies erleichtert die Arbeit, birgt jedoch eine gewisse Gefahr in sich.

Viele Entwickler, vor allem jene, die ihre VBA-Erfahrung in Excel gemacht haben, benutzen *Me* oder *ThisDocument*, ohne sich der Bedeutung bewusst zu sein. Diese weisen auf den *Code-Behälter* hin. Solange mit einem Dokument gearbeitet wird, gibt es kein Problem; das Steuerelement ist im gleichen Dokument-Objekt wie der Code.

Bereiten Sie jedoch eine Dokument-Vorlage vor, wird alles funktionieren, während in der Vorlage getestet wird. Wird jedoch ein neues Dokument von der Vorlage erstellt, treten häufig unerwartete Ergebnisse auf, falls Steuerelemente über das *ThisDocument*-Objekt angesprochen werden. Der Code

bezieht sich dann auf die Steuerelemente *der Vorlage* und nicht auf diejenigen des aktuellen Dokuments.

**WICHTIG**

Verwenden Sie in Word immer `ActiveDocument` anstatt `ThisDocument`, um auf das aktuelle Dokument Bezug zu nehmen. Bedienen Sie sich des Objekts `ThisDocument` nur, wenn Sie ausdrücklich den Code-Behälter ansprechen wollen.



Die Beispieldatei *Bsp11\_06.dot* finden Sie auf der CD-ROM zum Buch im Ordner `\Beispiele\Kap11`.

**HINWEIS**

Wir befassen uns hier nicht mit dem programmatischen Erstellen von ActiveX-Steuerelementen. Die Grundsyntax kann mit dem Makrorekorder ermittelt werden. Wie in einem VBA-Projekt Code dynamisch hinzugefügt wird, steht in Kapitel 19 beschrieben.

## Zusammenfassung

In diesem Kapitel wurde auf OLE-Objekte zugegriffen, die in einem Dokument eingebettet sind. Diese Objekte wurden aus Word heraus ferngesteuert.

- In diesem Kapitel wurde zunächst gezeigt, wie eingebettete Excel-Tabellen (Seite 543) und Excel-Diagramme (Seite 550) bearbeitet werden können.
- Im Weiteren wurde erläutert, wie der Zugriff auf MS Graph-Diagramme (Seite 555) und Word-Art-Objekte (Seite 559) erfolgen kann.
- Ein dritter und letzter Schwerpunkt in diesem Kapitel bildet das Einbinden von ActiveX-Steuerelementen (Seite 564) in ein Dokument.

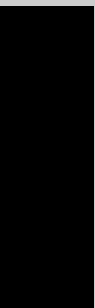


# Teil D

## Optimierung der Benutzerschnittstelle

### In diesem Teil:

<b>Kapitel 12</b>	Anwendungsoptionen	575
<b>Kapitel 13</b>	Speicherort von Anpassungen	599
<b>Kapitel 14</b>	Mit Dialogfeldern arbeiten	607
<b>Kapitel 15</b>	Menüs und Symbolleisten	649
<b>Kapitel 16</b>	Tastaturbelegungen	665
<b>Kapitel 17</b>	Aufgabenbereiche	683
<b>Kapitel 18</b>	Interne Word-Befehle übersteuern	707
<b>Kapitel 19</b>	Zugriff auf den Visual Basic-Editor (VBE)	713



## Kapitel 12

# Anwendungsoptionen

### In diesem Kapitel:

Dokumentvorlage <i>Normal.dot</i> konfigurieren	576
Benutzereinstellungen abspeichern	586
Dokumenteinstellungen abspeichern	593
Zusammenfassung	598

Viele Einstellungen und Optionen beeinflussen ein Dokument sowohl während dessen Erstellung als auch beim Öffnen. Jedes Dokument wird auf der Basis irgendeiner Dokumentvorlage erstellt. Deshalb muss bereits diese Dokumentvorlage entsprechend konfiguriert werden, damit die erwünschten Werte in die neu erstellten Dokumente einfließen. Auch diese Dokumentvorlage hat ihren Ursprung auf einer anderen Dokumentvorlage – oft die globale Standardvorlage *Normal.dot* – und deshalb ist es wichtig, dass die *Normal.dot* als Basis entsprechend angepasst wird. Mehr über die Rolle der *Normal.dot* erfahren Sie in Kapitel 1.

## Dokumentvorlage *Normal.dot* konfigurieren



Normal.dot

In einer Firmenumgebung, bei der Microsoft Word von mehreren Anwendern genutzt wird, müssen Sie sich früher oder später einige Gedanken zur Konfiguration des Programms machen. Bei der Konfiguration steht allerdings weniger die Installation der benötigten Programmmodule im Vordergrund, sondern eher die Konfiguration der Arbeitsumgebung.

Der Anwender soll vom Programm unterstützt werden, so dass seine tägliche Arbeit erleichtert wird. Dazu gehören Punkte wie automatische Trennhilfe, Rechtschreibprüfung usw. Übergeordnet müssen die Vorgaben zur Darstellung von Dokumenten, dem so genannten Corporate Design, berücksichtigt werden. Aus diesen Gründen muss nach erfolgter Programminstallation die Arbeitsumgebung konfiguriert und optimiert werden.

### WICHTIG

Aus Sicht der Autoren bedeutet dies aber nicht, dass innerhalb einer Firmenumgebung nur eine zentrale, gemeinsam genutzte *Normal.dot* im Einsatz steht. Das Gegenteil ist der Fall. Jeder Anwender verfügt über eine eigene Datei und kann seine Arbeitsumgebung entsprechend anpassen.

Die speziell angepasste Standardvorlage wird als so genannte »UrNormal.dot« an einer zentralen Stelle im Netzwerk abgelegt. Dieses Original muss vor dem ersten Programmstart von Word als Kopie einem jeden Anwender zur Verfügung gestellt werden. Ansonsten wird von Word eine vordefinierte Datei angelegt, welche die firmenspezifischen Anpassungen nicht enthält.

Ein Überarbeiten dieser zentralen Datei darf nur sehr selten der Fall sein (Beispiel: Änderungen des Corporate Design), denn das Anpassen des Originals würde unweigerlich eine erneute Verbreitung dieser Datei an alle Anwender bedeuten. Ohne entsprechende Maßnahmen hätte dies zur Folge, dass die Anwender ihre persönlichen Einstellungen verlieren würden.

## Vorlage »UrNormal.dot« erstellen

Damit die neue Standardvorlage die Bezeichnung »UrNormal.dot« wirklich verdient, sollten die nachstehenden Punkte vor dem Verteilen der Dokumentvorlage festgelegt werden:

- Die Einstellungen für *Seitenränder*, *Orientierung*, *Papierformat* sowie *Abstand von Kopf- bzw. Fußzeilen* werden im Abschnitt »Seitenlayout festlegen« erläutert.
- Die Einstellungen zu den *Dokumenteigenschaften* werden im Abschnitt »Dokumenteigenschaften eintragen« behandelt.
- Mögliche Einstellungen zur *Dokumentansicht* und *Zoomfaktor* werden im Abschnitt »Dokumentansicht und Zoomfaktor bestimmen« aufgezeigt.

- Die Liste der vorhandenen *AutoTexte* wird im Abschnitt »AutoTexte entfernen« überarbeitet.
- Möglichkeiten zur Optimierung der *Formatvorlagen* werden im Abschnitt »Formatvorlagen definieren« aufgezeigt.
- Die Einstellungen zur integrierten *Silbentrennung* werden im Abschnitt »Silbentrennung konfigurieren« festgelegt.
- Die Liste der vorhandenen *AutoKorrektur-Einträge* wird im Abschnitt »AutoKorrektur definieren« überarbeitet und ergänzt.
- Als letzter Arbeitsschritt werden die dokumentspezifischen *Optionen* im Abschnitt »Optionen bearbeiten« erläutert.

**WICHTIG**

Gemeinsam genutzte Makros, egal ob aus interner Quelle oder von Anbietern von Programmiererweiterungen, werden auf keinen Fall in der *Normal.dot* abgelegt. Der Programmierer verfügt über genügend andere Möglichkeiten, um diese dem Anwender zur Verfügung zu stellen. Welche Möglichkeiten zur Verfügung stehen erfahren Sie in Kapitel 9 und in Kapitel 13.

Anhand der nachstehenden Zeilen soll Ihnen aufgezeigt werden, wie Sie für Ihre eigene Umgebung eine »UrNormal.dot« konfigurieren können. Es steht Ihnen jedoch frei, einzelne Punkte gemäß Ihren Vorstellungen und betrieblichen Vorgaben anzupassen. Die Zusammenstellung soll Ihnen ein paar Gedankengänge mit auf den Weg geben. Es werden nur jene Einstellungen behandelt, die zum aktuellen Thema von Bedeutung sind.

## Originalvorlage *Normal.dot* durch Word erstellen lassen

Bevor Sie mit dem Konfigurieren und Anpassen der *Normal.dot* beginnen können, muss sichergestellt werden, dass auf der Arbeitsstation eine originale *Normal.dot* zur Verfügung steht, welche noch nicht verändert wurde. Um dies zu erreichen, gehen Sie wie folgt vor:

1. Beenden Sie Word, falls das Programm noch aktiv ist.
2. Starten Sie den Windows-Explorer und suchen Sie alle Dateien mit der Bezeichnung *Normal.dot*.
3. Löschen Sie alle gefundenen Dateien oder benennen Sie diese um.

**HINWEIS**

Beachten Sie, dass beim Löschen der *Normal.dot* Ihre persönlichen Einstellungen für *Formatvorlagen*, *AutoTexte*, *Symbolleisten* sowie *Makroprojektelemente* gleichzeitig verloren gehen, soweit sie in dieser Datei gespeichert sind.

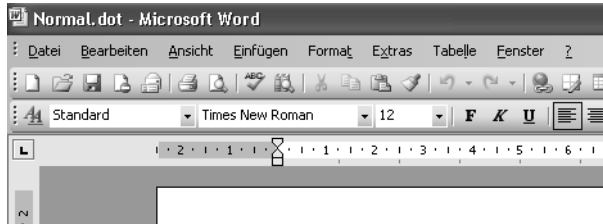
Wird die Datei nur umbenannt und in einem späteren Schritt vom System entfernt, können Sie diese Elemente nachträglich in die neue persönliche *Normal.dot* übertragen. Stellen Sie sicher, dass die *Normal.dot* und nicht die »UrNormal.dot« verwendet wird.

Um einzelne oder gleich alle Elemente von der alten Datei in die neue *Normal.dot* zu übertragen, rufen Sie den Menübefehl *Extras/Vorlagen und Add-Ins* auf und betätigen die Schaltfläche *Organisieren*. Mehr zum Thema Organisieren finden Sie in Kapitel 1.

4. Starten Sie Word und beenden Sie das Programm ohne damit zu arbeiten. In diesem Moment wurde eine neue *Normal.dot* angelegt.
5. Starten Sie erneut eine Suche nach dieser Datei.
6. Klicken Sie jetzt mit der rechten Maustaste auf die gefundene Datei und wählen Sie im Kontextmenü den Eintrag *Öffnen* aus.

7. Stellen Sie sicher, dass Sie wirklich die *Normal.dot* bearbeiten. Beachten Sie dazu, dass der Dateiname wie in Abbildung 12.1 in der Titelleiste erscheint.

**Abbildg. 12.1**

 Bearbeiten der geöffneten *Normal.dot*


**WICHTIG** Es ist enorm wichtig, dass die *Normal.dot* wie vorhin gezeigt erstellt wurde. Das Abspeichern einer beliebigen Datei als Dokumentvorlage mit der Bezeichnung *Normal.dot* macht aus dieser Datei noch keine *Normal.dot*, wie wir sie benötigen.

## Seitenlayout festlegen

In einem ersten Schritt wird das Seitenlayout der Standardvorlage festgelegt. Rufen Sie den Menübefehl *Datei/Seite einrichten* auf, um die drei nachstehenden Punkte zu bearbeiten:

- Auf der Registerkarte *Seitenränder* werden die gewünschten Werte für die *Ränder* und die *Orientierung* eingetragen.
- Auf der Registerkarte *Format* wird das gewünschte *Papierformat* eingetragen.
- Auf der Registerkarte *Layout* wird für die Kopf- und Fußzeilen der *Abstand vom Seitenrand* eingetragen.

## Dokumenteigenschaften eintragen

In einem zweiten Schritt werden die Eigenschaften des Dokuments festgelegt. Rufen Sie dazu den Menübefehl *Datei/Eigenschaften* auf. Da wir uns mit der »UrNormal.dot« beschäftigen, empfehlen wir bei allen Feldern die vorgeschlagenen Werte zu entfernen, damit später keine falschen Werte in die Dokumente übernommen werden.

**ACHTUNG** Im Gegensatz zu allen anderen Dokumentvorlagen verhält sich die *Normal.dot* ein bisschen widerspenstig. So können nur in den Feldern *Titel*, *Thema* und *Stichwörter* Werte hinterlegt werden, die später auch in die neuen Dokumente übernommen werden.

Alle anderen Dokumentvorlagen verhalten sich so, wie dies gewünscht wird. Hier können alle Felder mit einem entsprechenden Wert vorgelegt werden. Im Weiteren ist es möglich, auf der Registerkarte *Anpassen* eigene Dokumenteigenschaften zu erfassen, welche ebenfalls in das Dokument übernommen werden.

## Dokumentansicht und Zoomfaktor bestimmen

Im dritten Arbeitsschritt wird die gewünschte Dokumentansicht festgelegt. Aus unserer Sicht deckt das Seitenlayout die Anforderungen der Benutzer am besten ab. Ein dynamischer Zoomfaktor stellt zudem sicher, dass immer die ganze Breite des Dokuments am Bildschirm sichtbar ist.

- Wählen Sie den Menübefehl *Ansicht/Seitenlayout*, um die vorgeschlagene Ansicht zu aktivieren.
- Rufen Sie den Menübefehl *Ansicht/Zoom* auf und wählen Sie den Wert *Seitenbreite* aus.

## AutoTexte entfernen

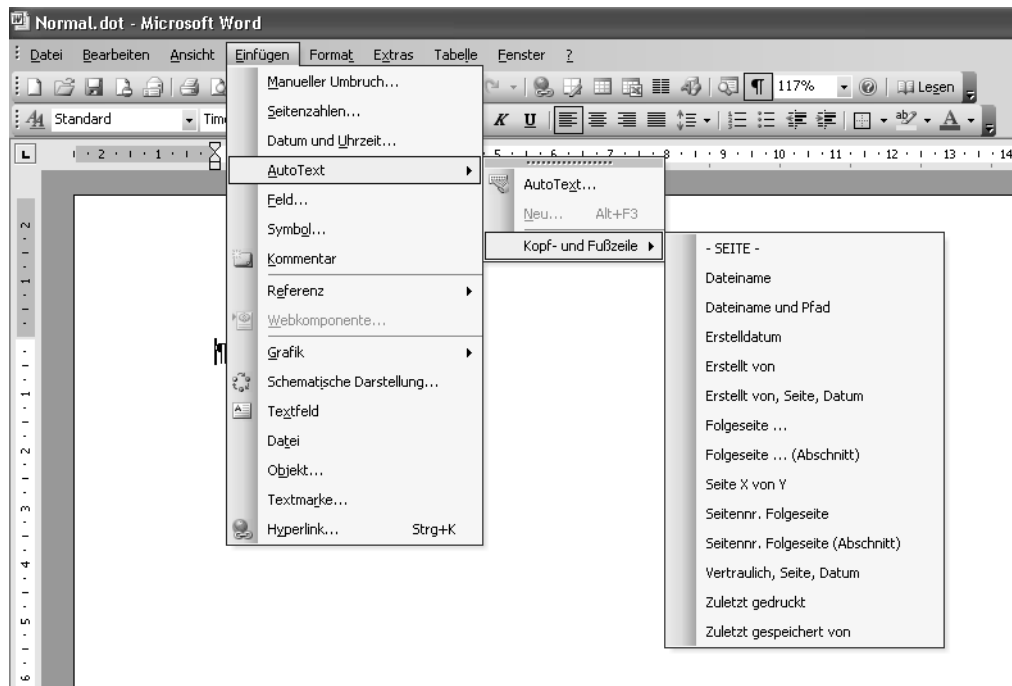
Der vierte Arbeitsschritt widmet sich den vordefinierten AutoTexten. Aus Sicht der Autoren hat der Anwender, mit Ausnahme jener AutoTexte im Bereich *Kopf- und Fußzeile*, keinen direkten Nutzen. Im Gegenteil, die Übersicht über die angebotenen AutoTexte wird erschwert.

Aus diesem Grunde empfehlen wir, alle vordefinierten AutoTexte mit Ausnahme jener der Kategorie *Kopf- und Fußzeile* zu löschen. Gehen Sie dazu wie folgt vor:

1. Rufen Sie den Menübefehl *Einfügen/AutoText/AutoText* auf.
2. Markieren Sie einen der nicht benötigten Einträge und betätigen Sie die Schaltfläche *Löschen*.
3. Wiederholen Sie diesen Vorgang, bis nur noch die gewünschten Einträge zu Auswahl stehen.

Nachdem Sie die Liste der AutoText-Einträge erfolgreich bearbeitet haben, entspricht diese ungefähr der Liste wie in Abbildung 12.2.

Abbildg. 12.2 Liste der nützlichen AutoTexte nach dem Löschen der überzähligen Einträge



### WICHTIG

Gemeinsam genutzte AutoTexte werden auf keinen Fall in der *Normal.dot* hinterlegt. Word verfügt über genügend andere Möglichkeiten, um diese dem Anwender zur Verfügung zu stellen. Welche Möglichkeiten zur Verfügung stehen, erfahren Sie in Kapitel 9 und in Kapitel 13.

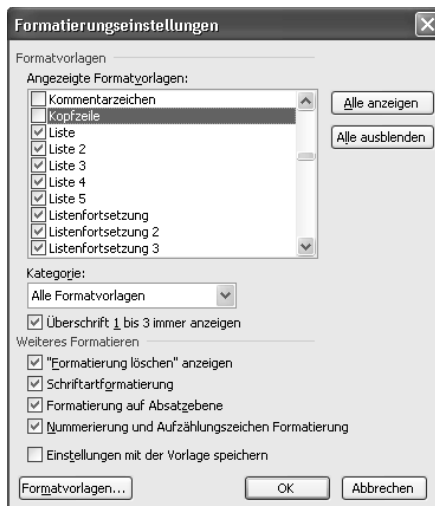
## Formatvorlagen definieren

Im fünften Arbeitsschritt werden die benötigten Formatvorlagen definiert. Sie werden verstehen, dass wir hier keine Anleitung geben können wie Sie die Formatvorlagen aufbauen müssen, da die Anforderung sehr unterschiedlich sind. Grundsätzlich sollten Sie sich jedoch zu den nachstehenden Punkten Gedanken machen:

- Welche der vordefinierten Formatvorlagen müssen auf die eigenen Bedürfnisse hin angepasst werden? Aus unserer Sicht müssen mindestens die Eigenschaften der nachstehenden Formatvorlagen überprüft werden:
  - *Standard* und *Standardeinzug*
  - *Überschrift 1* bis *Überschrift 4* und *Verzeichnis 1* bis *Verzeichnis 3*
  - *Kopfzeilen* und *Fußzeilen*
  - *Normale Tabelle*
  - *Fußnotenzeichen* und *Fußnotentext*
  - *Endnotenzeichen* und *Endnotentext*
  - *Hyperlink* und *BesucherHyperlink*
- Welche Zeichen- und Absatzformate werden den benötigten Formatvorlagen zugewiesen?
- Welche Formatvorlagen müssen in der Liste der vorhandenen Formatvorlagen sichtbar sein? Viele Formatvorlagen werden von Word automatisch zugewiesen und müssen deshalb nicht unbedingt aufgelistet werden (beispielsweise die Formatvorlage *Kopfzeilen* innerhalb von *Kopfzeilen*).

Abbildg. 12.3

Sichtbarkeit der Formatvorlagen im Listenfeld *Angezeigte Formatvorlagen* festlegen



## Silbentrennung konfigurieren

Im sechsten Arbeitsschritt werden die Eigenschaften der Silbentrennung festgelegt. Um die automatische Silbentrennung zu aktivieren, rufen Sie den Menübefehl *Extras/Sprache/Silbentrennung* auf und aktivieren das Kontrollkästchen *Automatische Silbentrennung*.



## AutoKorrektur definieren

Im siebten Arbeitsschritt wird die Liste der vorhandenen AutoKorrektur-Einträge überarbeitet und bei Bedarf mit eigenen Einträgen ergänzt. Eigene Einträge sind dann besonders sinnvoll, wenn innerhalb einer Firmenumgebung sichergestellt werden soll, dass einzelne Wörter von allen Anwendern immer gleich geschrieben und gleich formatiert werden sollen. Als Beispiel können Produktbezeichnungen, chemische Formeln, firmeninterne Terminologien usw. genannt werden.

Um die Liste der AutoKorrektur-Einträge zu bearbeiten, wählen Sie den Menübefehl *Extras/AutoKorrektur-Optionen* an. In der entsprechenden Liste werden überzählige Einträge entfernt und eigene Einträge hinzugefügt.

Abbildg. 12.4 Formatierten AutoKorrektur-Eintrag erfassen



**HINWEIS** Um einen formatierten AutoKorrektur-Eintrag zu erstellen, gehen Sie wie folgt vor:

1. Erfassen Sie den gewünschten Text in einem leeren Dokument.
2. Formatieren Sie den Text gemäß Ihren Vorstellungen.
3. Markieren Sie die entsprechende Textsequenz und stellen Sie sicher, dass die nachfolgende Absatzmarke in der Markierung **nicht** enthalten ist.
4. Rufen Sie den Menübefehl *Extras/AutoKorrektur-Optionen* auf. Aktivieren Sie im Dialogfeld *AutoKorrektur* die Option *Formatierten Text* und tragen Sie den zu ersetzenden Wert im Feld *Ersetzen* ein.

## Optionen bearbeiten

Im achten Schritt werden die Optionen von Word bearbeitet. Um die Einstellungen zu bearbeiten, wählen Sie den Menübefehl *Extras/Optionen* an. Hier werden jetzt Schritt für Schritt die relevanten Registerkarten bearbeitet.

**WICHTIG** Einige der Einstellungen, die durch den Menübefehl *Extras/Optionen* definiert werden können, sind für die gesamte Programmumgebung gültig. Die anderen Einstellungen haben einen direkten Bezug auf das aktuelle Dokument.

An dieser Stelle werden nur jene Einstellungen behandelt, deren Bezug sich auf das aktuelle Dokument, in diesem Falle auf die *Normal.dot*, beziehen.

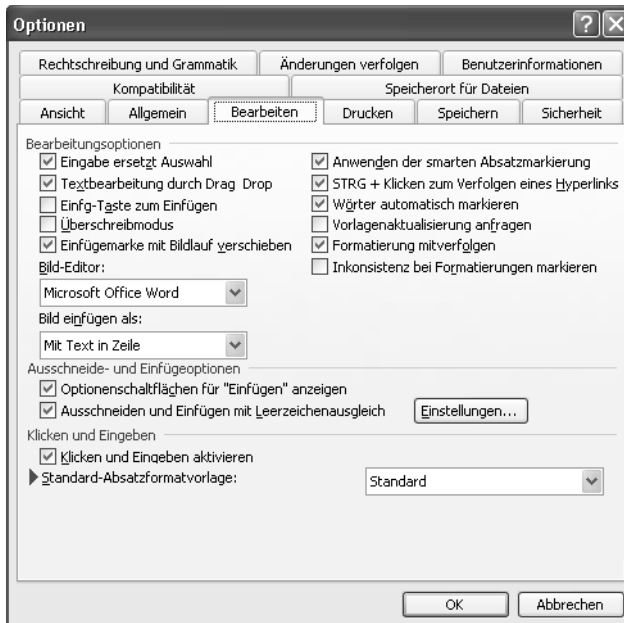
Beachten Sie die nebenstehend gezeigte Markierung in den nachfolgenden Abbildungen!

Wir verzichten darauf, jede einzelne Einstellung explizit mit dem zugehörigen Werten aufzulisten. Stattdessen versuchen wir die wichtigsten Punkte kurz zu erläutern. Alle eingestellten Optionen und die zugehörigen Vorschlagswerte sind in Abbildung 12.5 bis Abbildung 12.10 zusammengefasst. Zum Hervorheben der betroffenen Optionen wurde auf den zugehörigen Abbildungen der Text zusätzlich markiert.

### Registerkarte *Bearbeiten*

Die Möglichkeit von *Klicken und Eingeben* ist für weniger geübte Anwender eine einfache und schnelle Möglichkeit, um Text auf einem Blatt an jeder gewünschten Stelle zu platzieren. Aus diesem Grunde wird dieser Funktion die gewünschte Formatvorlage zugewiesen.

Abbildg. 12.5 Festlegen der nötigen Optionen auf der Registerkarte *Bearbeiten*



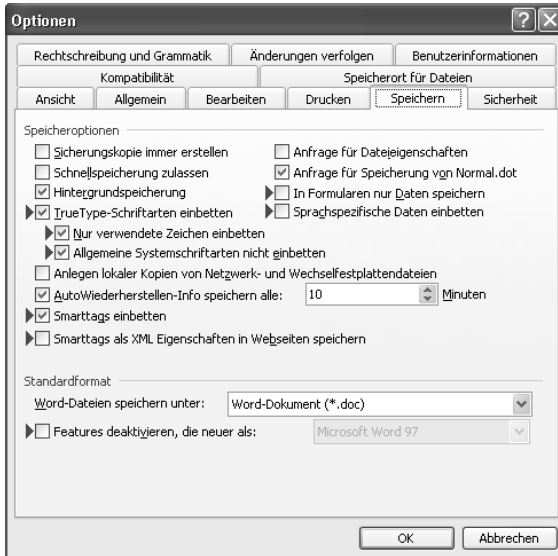
### Registerkarte *Speichern*

Kommt in einer Firmenumgebung eine spezielle Schriftart zum Einsatz, welche auf das Corporate Design abgestimmt ist, ist es mehr als sinnvoll, wenn die entsprechenden Schriftenarten in die einzelnen Dokumente eingebunden werden. So ist sicher gestellt, dass jedes Dokument auch auf firmenfremden Arbeitsgeräten über das gewünschte Layout verfügen.

**HINWEIS** Beachten Sie jedoch, dass das Einbetten der Schriftarten zusätzlichen Speicherplatz benötigt. Dokumente mit eingebetteten Schriften sind entsprechend größer.

Das Aktivieren des Kontrollkästchens *Sprachspezifische Daten einbetten* scheint von der Bezeichnung her sinnvoll zu sein. Ein direkter Nutzen ist jedoch nicht vorhanden, da die entsprechende Schnittstelle (Erkennung von Spracheingabe und Handschrift) in der deutschen Version von Word nicht implementiert ist.

Abbildg. 12.6 Festlegen der nötigen Optionen auf der Registerkarte *Speichern*



## Eigenschaften der TrueType-Schriften beachten

In den Eigenschaften einer TrueType-Schrift legt der Hersteller fest, ob diese überhaupt in ein Dokument eingebunden werden darf. Falls dies der Fall ist, wird weiter festgelegt, wie die Schriftart auf einem anderen Computer verwendet werden darf. Einer TrueType-Schrift wird somit eine der nachstehenden Eigenschaften zugewiesen:

- *Geschützt*, die Schriftart kann nicht eingebettet werden.
- *Drucken erlaubt*, die Schriftart kann in ein Dokument eingebunden werden, das Dokument kann mit der entsprechenden Schrift gedruckt werden.
- *Schreiben erlaubt*, das Dokument kann zusätzlich mit der entsprechenden Schriftart bearbeitet werden.
- *Installieren erlaubt*, die Schriftart kann zusätzlich vom Dokument aus auf dem fremden Arbeitsgerät installiert werden.

Wird innerhalb der Firmenumgebung eine spezielle Schriftart verwendet, sollte bereits bei der Anschaffung derselben eine Lizenz mit der Eigenschaft *Schreiben erlaubt* (Editable embedding allowed) erstanden werden.



TrueType.ttf

Um die Eigenschaften einer TrueType-Schrift einsehen zu können, wird von Microsoft eine Erweiterung zum Download angeboten. Dieses Programm erweitert die Standarddarstellung der Eigenschaften um zusätzliche Registerkarten.

Abbildg. 12.7 Die Eigenschaften der TrueType-Schrift prüfen



Die Datei *tttext.exe* finden Sie im Ordner *\Beilagen\TrueTypeFont Eigenschaften* auf der CD-ROM zum Buch oder im Internet auf der Homepage von Microsoft unter <http://www.microsoft.com/typography/TrueTypeProperty21.mspx>.

### Registerkarte *Sicherheit*

Das Aktivieren des Kontrollkästchens *Schreibschutz empfehlen* ist für Dokumentvorlagen selten sinnvoll. Ansonsten stellt Word bei jedem Anlegen eines neuen Dokuments, welches auf dieser Dokumentvorlage beruht, die Frage, ob die betreffende Datei mit oder ohne Schreibschutz geöffnet werden soll.

Abbildg. 12.8 Festlegen der nötigen Optionen auf der Registerkarte *Sicherheit*



Um eine Dokumentvorlage vor Änderungen zu schützen, sollte die betreffende Datei im Windows-Explorer als schreibgeschützt markiert werden.

Das grundsätzliche Entfernen von persönlichen Daten aus einem Dokument ist ebenfalls nicht empfehlenswert, denn davon sind unter anderem die Namen des Verfassers eines Kommentars betroffen.

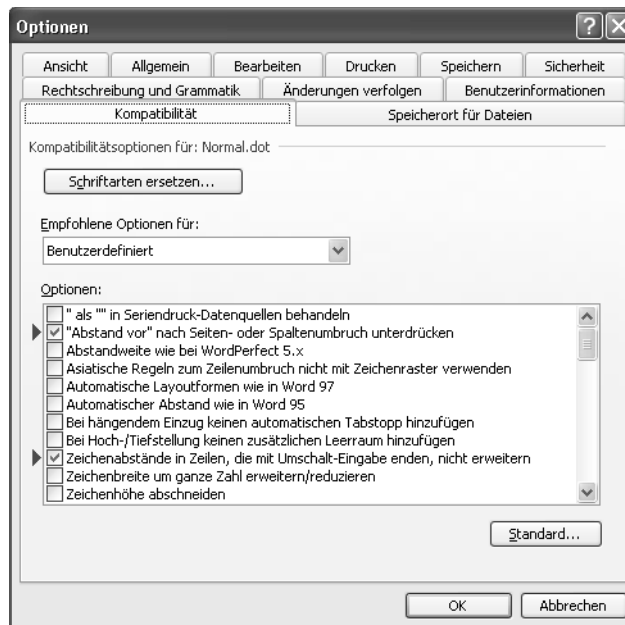
### Registerkarte *Kompatibilität*

Von den möglichen Einstellungen von Word zur Kompatibilität zu älteren Programmversionen wurden zwei Optionen ausgewählt, die den Anwender direkt bei der Arbeit unterstützen.

#### HINWEIS

Möchten Sie erfahren, was die einzelnen Optionen bedeuten und welche Auswirkungen diese auf das Dokument haben, so steht Ihnen in der Microsoft Knowledge Base (<http://support.microsoft.com/search/>) der Artikel 288792 »Beschreibung der Registerkarte "Kompatibilität" im Word 2002-Dialogfeld "Optionen"« zur Verfügung.

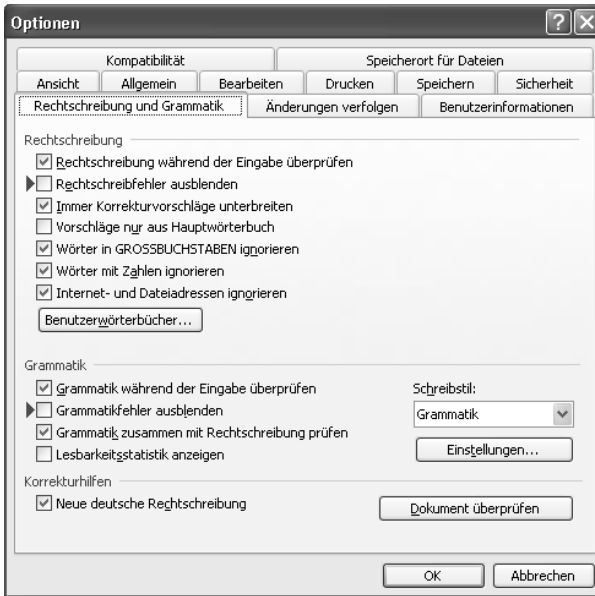
Abbildg. 12.9 Festlegen der notwendigen Optionen auf der Registerkarte *Kompatibilität*



### Registerkarte *Rechtschreibung und Grammatik*

Aus Sicht der Autoren ist es angebracht, wenn dem Anwender die beiden Hilfsmittel als Standard zur Verfügung gestellt werden. Deshalb werden diese beiden Kontrollkästchen nicht aktiviert.

Abbildg. 12.10 Festlegen der nötigen Optionen auf der Registerkarte *Rechtschreibung und Grammatik*



An dieser Stelle möchten wir einmal mehr betonen, dass die vorgeschlagenen Werte für die einzelnen Einstellungen auf der Erfahrung der Autoren aufbauen. Es bleibt also weiterhin Ihnen überlassen, welche der Vorschläge Sie in Ihre »UrNormal.dot« übernehmen möchten.

### **Vorlage *UrNormal.dot* speichern**

Als letzter Arbeitsschritt speichern Sie Ihre fertig konfigurierte »UrNormal.dot« ab. Erstellen Sie mindestens eine Sicherheitskopie dieser wertvollen Datei.

Stellen Sie diese Datei im Netzwerk zur Verfügung, so dass für jeden Anwender vor dem ersten Start von Word eine Kopie im Ordner mit den *Benutzervorlagen* angelegt wird.

## **Benutzereinstellungen abspeichern**

In der konfigurierten *Normal.dot* wurden Einstellungen vorgenommen, die den Anwender generell unterstützen sollen. Neben diesen Einstellungen ist es sinnvoll, wenn zusätzliche Werte über die aktuelle Arbeitsumgebung hinaus abgespeichert werden können, damit diese beim nächsten Programmaufruf wieder zur Verfügung stehen.

Von Word selber werden unzählige Benutzerparameter und -einstellungen abgespeichert. Diese Werte werden entweder in der entsprechenden Dokumentvorlage, meistens ist dies die *Normal.dot*, oder in der Windows-Registrierung abgelegt.

In den meisten Windows-Programmen ist es ebenfalls üblich, dass der Anwender persönliche Einstellungen, die das Verhalten des Programms beeinflussen, abspeichern kann. Oder das Programm speichert ohne das Zutun des Anwenders gewisse Einstellungen ab, um diese zu einem späteren Zeitpunkt erneut verwenden zu können.

Diese Funktionalität kann ebenfalls in die Makros integriert werden und unterstützt den Anwender zusätzlich bei seiner täglichen Arbeit. Besonders bei Makros, die mit einem Dialogfeld arbeiten, ist es oft sinnvoll, wenn der Status der Kontrollkästchen, Optionen usw. zwischengespeichert wird. Um spezielle Optionen des Anwenders zu speichern, sind zwei unterschiedliche Speicherorte möglich:

- Die Windows-Registrierung zum Abspeichern von unterschiedlichsten Daten, bezogen auf die aktuelle Arbeitsstation oder den aktuellen Anwender (Benutzerprofil).
- Das Dateisystem als Speicherort von sämtlichen Dateien, im aktuellen Fall in Form von Konfigurationsdateien.

Für welche der beiden Arten Sie sich entscheiden, bleibt Ihnen überlassen, es ist durchaus sinnvoll im gleichen Makro beide Arten zu verwenden. Als Faustregel können wir Ihnen Folgendes empfehlen:

- Persönliche Einstellungen des Anwenders werden in der Windows-Registrierung abgespeichert (beispielsweise die Position des Dialogfelds).
- Allgemeine Konfigurationsparameter zum Makro werden in einer eigenen Konfigurationsdatei abgespeichert (beispielsweise die Definition eines zugehörigen Datenverzeichnisses).
- Allgemeine Daten zum Makro werden in einer Konfigurationsdatei gespeichert (beispielsweise die Aufstellung aller möglichen Einträge in einem Listenfeld).

Im Befehlsumfang von Visual Basic für Applikationen sind zwei Funktionen und eine Methode integriert, welche das Schreiben und Lesen von Benutzereinstellungen unterstützen.

### Was ist eine Konfigurationsdatei?

In einer Konfigurationsdatei werden die Einstellungen zur Steuerung eines Programms hinterlegt. Die entsprechenden Dateien tragen normalerweise die Erweiterung *.ini* und werden deshalb salopp als *INI-Datei* bezeichnet.

Die Datei ist eine Textdatei und kann mit jedem Editor eingesehen werden. Die gespeicherten Daten sind in Abschnitte unterteilt. Für jeden Parameter steht eine Zeile zur Verfügung. Am Anfang der Zeile steht der Name des Eintrags, gefolgt von einem Gleichheitszeichen. Anschließend ist der zugehörige Wert eingetragen.

Mit den Programmzeilen aus dem Listing 12.4 wird ebenfalls eine ini-Datei aufgebaut. Das Resultat der entsprechenden Programmsequenz ist in Abbildung 12.11 dargestellt.

Abbildg. 12.11 Der Aufbau einer *.ini*-Datei



Die Möglichkeiten der Konfigurationsdateien standen vor allem in Microsoft Windows 3.11 und älteren Versionen im Vordergrund. Seit der Einführung von Microsoft Windows 95 steht die zentrale Windows-Registrierung dem Programmierer zur Verfügung. Viele Einstellungen, die früher in *.ini*-Dateien hinterlegt wurden, werden in den aktuellen Versionen von Windows in der Windows-Registrierung gespeichert.

Der allgemeine Trend in der Softwareentwicklung zeigt jedoch, dass Anwendungsdaten wieder vermehrt außerhalb der Windows-Registrierung abgespeichert werden. Der Grund dazu steht im Zusammenhang mit den Problemen, die entstehen können, wenn immer größere Datenmengen in dieser Datei abgelegt werden. Anstelle von *.ini*-Dateien verwenden Entwickler immer häufiger eine *.xml*-Datei zum Speichern solcher Daten.

## SaveSetting-Anweisung

Die `SaveSetting`-Anweisung dient zum Abspeichern von Einstellungen in der Windows-Registrierung. Sämtliche gespeicherten Daten werden in einer Struktur unterhalb von `HKEY_CURRENT_USER\Software\VB and VBA Program Settings` abgelegt (fehlende Schlüssel werden angelegt).

**Listing 12.1**    Abspeichern der Anwenderdaten mittels *SaveSetting* in der Windows-Registrierung

```
Sub SaveSetting_Demo()
    Dim strVorname As String
    Dim strName As String

    'Daten des Anwenders abfragen
    strVorname = InputBox("Bitte Vorname eingeben.")
    strName = InputBox("Bitte Familienname eingeben.")

    'Werte in Windows-Registrierung abspeichern
    SaveSetting AppName:="Word-Programmierung - Das Handbuch", _
        Section:="Anwenderdaten", _
        Key:="Vorname", _
        Setting:=strVorname

    SaveSetting AppName:="Word-Programmierung - Das Handbuch", _
        Section:="Anwenderdaten", _
        Key:="Name", _
        Setting:=strName
End Sub
```

## GetSetting-Funktion

Die `GetSetting`-Funktion dient zum Einlesen von Einstellungen aus der Windows-Registrierung. Es können nur Werte eingelesen werden, die sich innerhalb der Struktur von `HKEY_CURRENT_USER\Software\VB and VBA Program Settings` befinden. Für fehlende Werte wird eine leere Zeichenkette zurückgegeben, sofern der Funktion kein Standardwert als zusätzlicher Parameter übergeben wurde.



Listing 12.2 Auslesen der Anwenderdaten mittels *GetSetting* aus der Windows-Registrierung

```

Sub GetSetting_Demo()
    Dim strVorname As String
    Dim strName As String
    Dim strOrt As String

    'Werte aus Windows-Registrierung auslesen
    strVorname = GetSetting(AppName:="Word-Programmierung - Das Handbuch", _
        Section:="Anwenderdaten", _
        Key:="Vorname", _
        Default:="unbekannt")

    strName = GetSetting(AppName:="Word-Programmierung - Das Handbuch", _
        Section:="Anwenderdaten", _
        Key:="Name", _
        Default:="unbekannt")

    strOrt = GetSetting(AppName:="Word-Programmierung - Das Handbuch", _
        Section:="Anwenderdaten", _
        Key:="Ort", _
        Default:="unbekannt")

    'Daten des Anwenders ausgeben.
    MsgBox "Vorname" & vbTab & strVorname & vbCr & _
        "Name" & vbTab & strName & vbCr & _
        "Ort" & vbTab & strOrt
End Sub

```

## PrivateProfileString-Eigenschaft

Möchten Sie Werte, ohne die Einschränkung von *SaveSetting*, an irgendeinem Punkt innerhalb der Windows-Registrierung abspeichern oder von dort einlesen, bietet Ihnen die *PrivateProfileString*-Eigenschaft die entsprechenden Möglichkeiten an.

Diese Eigenschaft kann mit Daten aus der Windows-Registrierung umgehen und bietet Ihnen die gleichen Möglichkeiten im Umgang mit Konfigurationsdateien an. Anhand von Listing 12.3 bis Listing 12.6 werden Ihnen die Möglichkeiten aufgezeigt.

Müssen aus der Windows-Registrierung nicht nur einzelne Einträge, sondern ganze Untereinträge gelesen oder geschrieben werden, kann dies nur durch den Einsatz von so genannten API-Funktionen umgesetzt werden. Das Gleiche gilt, wenn beispielsweise ganze Abschnitte aus einer Konfigurationsdatei bearbeitet werden müssen. Wie solche Funktionen in den Programmcode eingebaut werden, wurde in Kapitel 4 mit entsprechenden Beispielen aufgezeigt.

Listing 12.3 Abspeichern der Anwenderdaten mittels *PrivateProfileString* aus der Windows-Registrierung

```

Sub PrivateProfileString_Demo_1()
    Const cstrSECTION As String = "HKEY_CURRENT_USER\Software\" & _
        "Word-Programmierung - Das Handbuch\Anwenderdaten"

    Dim strVorname As String
    Dim strName As String

```

**Listing 12.3** Abspeichern der Anwenderdaten mittels *PrivateProfileString* aus der Windows-Registrierung (*Fortsetzung*)

```
'Daten des Anwenders abfragen
strVorname = InputBox("Bitte Vorname eingeben.")
strName = InputBox("Bitte Familienname eingeben.")

'Werte in Windows-Registrierung abspeichern
System.PrivateProfileString(FileName="", _
    Section:=cstrSECTION, _
    Key:="Vorname") = strVorname

System.PrivateProfileString(FileName="", _
    Section:=cstrSECTION, _
    Key:="Name") = strName
End Sub
```

**HINWEIS**

Die *PrivateProfileString*-Eigenschaft greift automatisch auf die Windows-Registrierung zu, wenn an Stelle eines Dateinamens eine leere Zeichenkette eingetragen wird.

**Listing 12.4** Abspeichern der Anwenderdaten mittels *PrivateProfileString* in eine Konfigurationsdatei

```
Sub PrivateProfileString_Demo_2()
    Const cstrFILENAME As String = "C:\Das Handbuch.ini"

    Dim strVorname As String
    Dim strName As String

    'Daten des Anwenders abfragen
    strVorname = InputBox("Bitte Vorname eingeben.")
    strName = InputBox("Bitte Familienname eingeben.")

    'Werte in Konfigurationsdatei abspeichern
    System.PrivateProfileString(FileName:=cstrFILENAME, _
        Section:="Anwenderdaten", _
        Key:="Vorname") = strVorname

    System.PrivateProfileString(FileName:=cstrFILENAME, _
        Section:="Anwenderdaten", _
        Key:="Name") = strName
End Sub
```

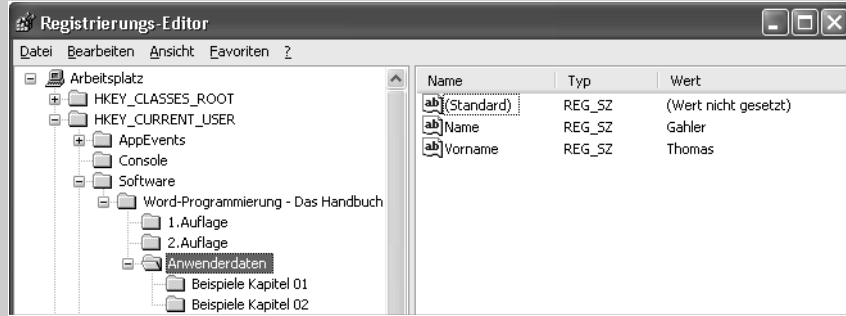
**Daten in der Windows-Registrierung abspeichern**

Werden persönliche Einstellungen des Anwenders in der Windows-Registrierung abgespeichert, so werden diese Werte im Bereich *HKEY\_CURRENT\_USER* abgelegt. Beachten Sie, dass die Daten innerhalb dieses Astes nicht willkürlich angelegt werden, sondern gewissen Spielregeln unterworfen sind. Die Einstellungen zu Programmen werden im Ast *Software* abgelegt.

Die einzelnen Äste werden nach dem Schema *Hersteller\Programm\Version* aufgebaut. Von der Programmversion unabhängige Einstellungen werden direkt im Ast *Programm* abgelegt.

Die gespeicherten Daten können nach Themen strukturiert werden, indem zusätzliche Äste unterhalb des Astes *Programm* bzw. des Astes *Version* angelegt werden. ►

Abbildung. 12.12 Strukturierte Datenablage in der Windows-Registrierung aufbauen



Mit der `PrivateProfileString`-Eigenschaft können die gespeicherten Werte auf die gleiche Art vom Speicherort ausgelesen werden.

Listing 12.5 Auslesen der Anwenderdaten mittels *PrivateProfileString* aus der Windows-Registrierung

```
Sub PrivateProfileString_Demo_3()
    Const cstrSECTION As String = "HKEY_CURRENT_USER\Software\" & _
        "Word-Programmierung - Das Handbuch\Anwenderdaten"

    Dim strVorname As String
    Dim strName As String
    Dim strOrt As String

    'Werte aus Windows-Registrierung auslesen
    strVorname = System.PrivateProfileString(FileName="", _
        Section:=cstrSECTION, _
        Key:="Vorname")

    strName = System.PrivateProfileString(FileName="", _
        Section:=cstrSECTION, _
        Key:="Name")

    strOrt = System.PrivateProfileString(FileName="", _
        Section:=cstrSECTION, _
        Key:="Ort")

    'Daten des Anwenders ausgeben.
    MsgBox "Vorname" & vbTab & strVorname & vbCr & _
        "Name" & vbTab & strName & vbCr & _
        "Ort" & vbTab & strOrt
End Sub
```

Listing 12.6 Auslesen der Anwenderdaten mittels *PrivateProfileString* aus einer Konfigurationsdatei

```
Sub PrivateProfileString_Demo_4()
    Const cstrFILENAME As String = "C:\Das Handbuch.ini"

    Dim strVorname As String
    Dim strName As String
    Dim strOrt As String
```

**Listing 12.6** Auslesen der Anwenderdaten mittels *PrivateProfileString* aus einer Konfigurationsdatei (Fortsetzung)

```
'Werte aus Konfigurationsdatei auslesen
strVorname = System.PrivateProfileString(FileName:=cstrFILENAME, _
    Section:="Anwenderdaten", _
    Key:="Vorname")

strName = System.PrivateProfileString(FileName:=cstrFILENAME, _
    Section:="Anwenderdaten", _
    Key:="Name")

strOrt = System.PrivateProfileString(FileName:=cstrFILENAME, _
    Section:="Anwenderdaten", _
    Key:="Ort")

'Daten des Anwenders ausgeben.
MsgBox "Vorname" & vbTab & strVorname & vbCr & _
    "Name" & vbTab & strName & vbCr & _
    "Ort" & vbTab & strOrt
End Sub
```

Die Eigenschaft *PrivateProfileString* gibt eine leere Zeichenkette zurück, wenn der gesuchte Eintrag nicht gefunden wurde. Leider kann kein Standardwert als zusätzlicher Parameter übergeben werden. Diesem Umstand wurde im Listing 12.7 Rechnung getragen, indem eine eigene, erweiterte Funktion verwendet wird.

Damit nicht nach jedem Aufruf dieser Eigenschaft eine Prüfung der Variable erfolgen muss, ist es sinnvoll, wenn diese regelmäßige Überprüfung innerhalb einer eigenen Funktion erledigt wird.

**Listing 12.7** *PrivateProfileString*-Eigenschaft als eigene Funktion mit Standardrückgabewert erweitert

```
Sub PrivateProfileString_Demo_5()
    Const cstrSECTION As String = "HKEY_CURRENT_USER\Software\" & _
        "Word-Programmierung - Das Handbuch\Anwenderdaten"

    Dim strVorname As String
    Dim strName As String
    Dim strOrt As String

    'Werte aus Windows-Registrierung auslesen
    'indirekt via Funktion 'fktPrivateProfileString()'
    strVorname = fktPrivateProfileString(strFileName="", _
        strSection:=cstrSECTION, _
        strKey:="Vorname", _
        strDefault:="unbekannt")

    strName = fktPrivateProfileString(strFileName="", _
        strSection:=cstrSECTION, _
        strKey:="Name", _
        strDefault:="unbekannt")

    strOrt = fktPrivateProfileString(strFileName="", _
        strSection:=cstrSECTION, _
        strKey:="Ort", _
        strDefault:="unbekannt")
End Sub
```

Listing 12.7 PrivateProfileString-Eigenschaft als eigene Funktion mit Standardrückgabewert erweitert (Fortsetzung)

```

'Daten des Anwenders ausgeben.
MsgBox "Vorname" & vbTab & strVorname & vbCr & _
    "Name" & vbTab & strName & vbCr & _
    "Ort" & vbTab & strOrt
End Sub

Public Function fktPrivateProfileString( _
    ByVal strFileName As String, _
    ByVal strSection As String, _
    ByVal strKey As String, _
    ByVal strDefault As String) As String

    Dim strEintrag As String

    'Eintrag aus Windows-Registrierung oder
    'Konfigurationsdatei auslesen
    strEintrag = System.PrivateProfileString( _
        FileName:=strFileName, _
        Section:=strSection, _
        Key:=strKey)

    'Überprüfen ob ein Eintrag gefunden wurde,
    'Ansonsten wird der Wert aus 'strDefault' verwendet
    If strEintrag = "" Then
        strEintrag = strDefault
    End If

    'Rückgabewert festlegen
    fktPrivateProfileString = strEintrag
End Function

```



Die aufgeführten Codesequenzen finden Sie in der Beispieldatei *Bsp12\_01.doc* auf der Buch-CD im Ordner *\Beispiele\Kap12* (siehe Einleitung).

## Dokumenteinstellungen abspeichern

Unter dem Begriff »Dokumenteinstellungen« verstehen wir statische Werte, die im Zusammenhang mit dem Inhalt des Dokuments stehen. In diesen Einstellungen werden Parameter hinterlegt, die für andere Makros zur Steuerung verwendet werden. Die nachstehenden Beispiele sollen das Ganze ein wenig näher bringen:

- Hinterlegen des Dateinamens für das Firmenlogo, welches beim Drucken auf einen Schwarz-weißdrucker bzw. bei einem Farbdrucker in das Dokument eingebunden wird.
- Hinterlegen der Anzahl, welche als Vorgabewert im Dialogfeld für den Ausdruck vorgeschlagen wird (beispielsweise die Anzahl Personen gemäß Verteilerliste in einem Protokoll).
- Abspeichern des Passworts für geschützte Dokumente, damit zu einem späteren Zeitpunkt der Dokumentschutz aufgehoben bzw. dieser automatisch mit einem Makro bearbeitet werden kann.

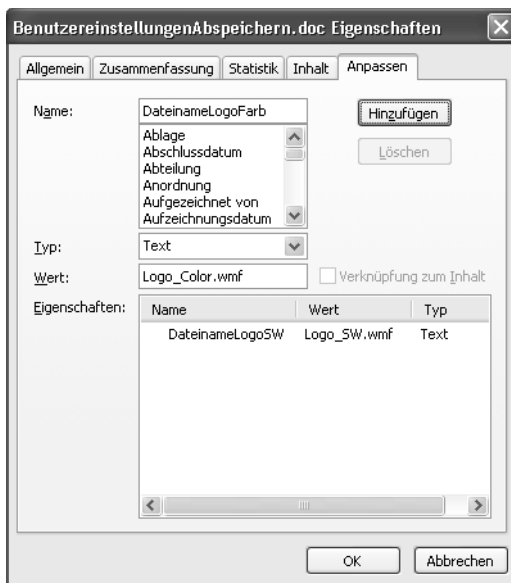
Zum Abspeichern dieser Werte stehen Ihnen zwei verschiedene Möglichkeiten zur Verfügung. Beide haben ihre Vor- und Nachteile.

Die erste Möglichkeit ist die Verwendung der *Dokumenteigenschaften*. Diese können sehr einfach auf die persönlichen Bedürfnisse hin angepasst und erweitert werden:

- Es können verschiedene Datentypen abgespeichert werden.
- Alle gespeicherten Werte können jederzeit eingesehen werden.
- Die Werte können auch von außerhalb, beispielsweise direkt im Windows-Explorer, eingesehen werden.
- Die Werte können von jedem Anwender sehr einfach manipuliert werden.
- Die Werte können manuell oder mit Codeprozeduren bearbeitet werden.

Um eine Dokumenteigenschaft manuell zu bearbeiten, rufen Sie den Menübefehl *Datei/Eigenschaften* auf und wechseln zur Registerkarte *Anpassen*.

**Abbildg. 12.13** Dokumenteigenschaft zum Verwalten des Firmenlogos erfassen



Die zweite Möglichkeit zum Abspeichern von Werten innerhalb eines Dokuments sind die *Dokumentvariablen*:

- Es stehen keine unterschiedlichen Datentypen zur Verfügung. Dokumentvariablen bestehen aus Zeichenketten. Sämtliche Werte müssen zuerst konvertiert werden.
- Die Werte können vom Anwender manuell nicht manipuliert werden.
- Die Werte können nur mittels Codeprozeduren bearbeitet werden.

## Dokumenteigenschaften bearbeiten

Bei den Dokumenteigenschaften gibt es zwei Kategorien. Die erste Kategorie beinhaltet die `BuiltInDocumentProperties`, in welchen die internen Werte von Word (beispielsweise der Name des Autors, das letzte Druckdatum usw.) abgelegt werden. Bei der zweiten Kategorie handelt es sich um die `CustomDocumentProperties`. Hier kann der Anwender seine eigenen Werte hinterlegen.

Die `CustomDocumentProperties` ist eine Auflistung. Diese kann mit den Methoden `Add`, `Delete` usw. bearbeitet werden.

**Listing 12.8** Benutzerdefinierte Dokumenteigenschaften bearbeiten

```
Sub CustomDocumentProperties_Demo()
    Dim docDemo As Word.Document
    Dim prop As DocumentProperty

    Set docDemo = Documents.Add

    'Dokumenteigenschaften anlegen
    docDemo.CustomDocumentProperties.Add _
        Name:="DateinameLogoSW", _
        Value:="Logo_SW.wmf", _
        LinkToContent:=False, _
        Type:=msoPropertyTypeString

    docDemo.CustomDocumentProperties.Add _
        Name:="DateinameLogoFarb", _
        Value:="Logo_Color.wmf", _
        LinkToContent:=False, _
        Type:=msoPropertyTypeString

    docDemo.CustomDocumentProperties.Add _
        Name:="AnzahlKopien", _
        Value:=1, _
        LinkToContent:=False, _
        Type:=msoPropertyTypeNumber

    'Dokumenteigenschaften auflisten
    For Each prop In docDemo.CustomDocumentProperties
        MsgBox "Name" & vbTab & prop.Name & vbCr & _
            "Wert" & vbTab & CStr(prop.Value)
    Next prop

    'Dokumenteigenschaften entfernen
    docDemo.CustomDocumentProperties("AnzahlKopien").Delete

    'Dokument als bearbeitet kennzeichnen
    docDemo.Saved = False
    Set docDemo = Nothing
End Sub
```

**PROFITIPP**

Werden bei einem gespeicherten Dokument die Dateieigenschaften mittels einer Codeprozedur bearbeitet, wird diese Änderung von Word nicht als solche erkannt. Folglich erscheint beim Schließen des Dokuments keine Speicheraufforderung. Die geänderten Dokumenteigenschaften würden also verloren gehen. Das Dokument muss deshalb zwingend innerhalb der Codeprozedur gespeichert oder als bearbeitet gekennzeichnet werden.

Im Listing 12.8 wurde das Dokument als bearbeitet gekennzeichnet. Dies wurde mit der Zeile `docDemo.Saved = False` erreicht.

Wird wie in Listing 12.8 auf ein Element einer Auflistung zugegriffen, müssen Sie sicher sein, dass dieses Element auch tatsächlich vorhanden ist. Ansonsten wird ein Laufzeitfehler erzeugt.

Der Aufruf der Methode `Delete` kann nur deshalb ohne vorherige Prüfung erfolgen, weil die entsprechende Dokumenteigenschaft vorab in der gleichen Prozedur angelegt wurde. Besser wäre jedoch, wenn eine Prüfung, wie in Listing 12.9 vorgestellt, erfolgen würde.

Die meisten Auflistungen stellen keine Methode zur Verfügung, mit welcher geprüft werden kann, ob ein Element innerhalb der Auflistung vorhanden ist. Eine Ausnahme bildet die `Bookmarks`-Auflistung. Somit bleibt nichts anders übrig, als eine entsprechende Funktion selbst zu erstellen. Diese Funktion muss mittels einer Schleife alle Elemente der Auflistung untersuchen.

**Listing 12.9** Funktion zum Überprüfen, ob eine Dokumenteigenschaft vorhanden ist

```
Sub Test()
    MsgBox fktDokumentEigenschaftVorhanden( _
        doc:=ActiveDocument, _
        strEigenschaftName:="AnzahlKopien")
End Sub

Public Function fktDokumentEigenschaftVorhanden( _
    ByVal doc As Word.Document, _
    ByVal strEigenschaftName As String) _
    As Boolean

    Dim prop As DocumentProperty

    'Dokumenteigenschaften untersuchen
    For Each prop In doc.CustomDocumentProperties
        If LCase$(prop.Name) = LCase$(strEigenschaftName) Then
            fktDokumentEigenschaftVorhanden = True
            Exit For
        End If
    Next prop
End Function
```

### Dateieigenschaften mittels *Dsofile.dll* bearbeiten

Die Dateieigenschaften aller Office-Dokumente können bearbeitet werden, ohne dass die entsprechenden Dateien in der zugehörigen Applikation geöffnet werden müssen. Dazu stellt Microsoft die Datei *dsofile.dll* zur Verfügung. ►



Möchten Sie erfahren, wie die Datei in ein VBA-Projekt eingebunden und wie die entsprechenden Methoden aufgerufen werden, steht Ihnen in der Microsoft Knowledge Base (<http://support.microsoft.com/search/>) der Artikel 224351 »Dsofile.dll ermöglicht Bearbeitung der Eigenschaften von Office-Dokumenten aus Visual Basic und ASP« zur Verfügung.



Die Datei *dsofile.exe* finden Sie im Ordner *\Beilagen\Dsofile* auf der CD-ROM zum Buch oder im Internet auf der Homepage von Microsoft unter <http://support.microsoft.com/default.aspx?scid=kb;de;224351>.

## Dokumentvariablen bearbeiten

Bei den Dokumentvariablen handelt es sich um einen Bereich im Dokument, auf den der Anwender keinen direkten Zugriff hat. Die Bearbeitung dieser Werte muss zwingend mit einer Codeprozedur ausgeführt werden.

Bitte beachten Sie, dass der Inhalt der Dokumentvariablen nicht geschützt ist. Der Inhalt dieser Variablen kann auch ohne Kenntnisse in Makroprogrammierung sichtbar gemacht werden, indem das Dokument ins HTML- oder XML-Format exportiert wird.

Die Variables-Auflistung kann mit den Methoden Add, Delete usw. bearbeitet werden.

Listing 12.10 Dokumentvariablen bearbeiten

```
Sub Variables_Demo()
    Dim docDemo As Word.Document
    Dim vrbl As Variable

    Set docDemo = Documents.Add

    'Dokumentvariable anlegen
    docDemo.Variables.Add
        Name:="DateinameLogoSW", _
        Value:="Logo_SW.wmf"

    docDemo.Variables.Add
        Name:="DateinameLogoFarb", _
        Value:="Logo_Color.wmf"

    docDemo.Variables.Add
        Name:="AnzahlKopien", _
        Value:=1

    'Dokumentvariable auflisten
    For Each vrbl In docDemo.Variables
        MsgBox "Name" & vbTab & vrbl.Name & vbCr & _
            "Wert" & vbTab & CStr(vrbl.Value)
    Next vrbl

    'Dokumentvariable entfernen
    docDemo.Variables("AnzahlKopien").Delete

    Set docDemo = Nothing
End Sub
```

Sie können jedoch auch die verkürzte Syntax zum Anlegen bzw. Löschen einer Dokumentvariablen verwenden und auf den Einsatz der beiden Methoden `Add` und `Delete` verzichten.

Anlegen und Löschen der Dokumentvariablen *DateinameLogoSW*:

```
docDemo.Variables("DateinameLogoSW") = "Logo_SW.wmf"  
docDemo.Variables("DateinameLogoSW") = ""
```

Wenn Sie die zweite Zeile, also das Löschen der Dokumentvariablen, genauer betrachten, werden Sie feststellen, dass die Variable gelöscht wird, indem eine leere Zeichenkette gesetzt wird. Dies bedeutet aber auch, dass keine leeren Dokumentvariablen auf Vorrat angelegt werden können.



---

Die aufgeführten Codesequenzen finden Sie in der Beispieldatei *Bsp12\_01.doc* auf der Buch-CD im Ordner *\Beispiele\Kap12* (siehe Einleitung).

---

## Zusammenfassung

In diesem Kapitel wurden die unterschiedlichen Möglichkeiten zum Abspeichern von Anwendungsoptionen aufgezeigt.

- Im ersten Abschnitt des Kapitels wurde aufgezeigt, weshalb es sinnvoll ist die allgemeine Dokumentvorlage *Normal.dot* explizit zu konfigurieren und welche Schritte dazu nötig sind (Seite 576 ff.).
- Im Weiteren wurde aufgezeigt, wie Benutzereinstellungen innerhalb einer Konfigurationsdatei oder der Windows-Registrierung abgespeichert und von dort wieder gelesen werden können (Seite 586).
- Im dritten und letzten Abschnitt wurde erläutert, wie Dokumenteinstellungen abgespeichert und wieder gelesen werden können (Seite 593).

## Kapitel 13

# Speicherort von Anpassungen

### In diesem Kapitel:

Der Speicherort einer Anpassung	600
Auswahl des Speicherorts	603
Kontext für COM-Add-Ins	603
Hierarchie der Anpassungen	604
Zusammenfassung	605

Heutzutage stellen alle Office-Anwendungen ähnliche Benutzerschnittstellen zur Verfügung. Dazu gehören beispielsweise Dialogfelder, Symbolleisten (inkl. Menüs) und Tastaturkürzel. Die Möglichkeiten, diese den eigenen Bedürfnissen anzupassen, sind von Anwendung zu Anwendung verschieden. Historisch betrachtet hat Word den Ruf, extrem anpassungsfähig zu sein. Von allen Office-Anwendungen bietet es dem Anwender sowie dem Entwickler die breiteste Palette von Anpassungsfähigkeiten.

Inzwischen unterstützen alle Module der Office-Familie das Erstellen mehrerer Symbolleisten und deren Bestückung mit Menübefehlen und Makros. Einzig Word stellt *alle* internen Befehle zur direkten Anwendung zur Verfügung. Dazu gehören auch alte Befehle, die nicht mehr in den Menüs zu finden sind (inzwischen unterstützt Word etwa 1.800 einzelne Befehle – kein Wunder, dass nicht alle in den Menülisten Platz haben!). Die Möglichkeiten zum Automatisieren von Symbolleisten werden in Kapitel 15 vorgestellt.

Des Weiteren kann der Entwickler einer Prozedur den Namen eines internen Befehls geben. Der Befehl wird »überdefiniert« und kann auf die eigenen Bedürfnisse hin angepasst werden. Soll beispielsweise ein Dokument nur gedruckt werden, wenn es ausschließlich mit einem bestimmten Satz von Formatvorlagen formatiert wurde, kann eine Prozedur mit der Bezeichnung DateiDrucken dies nachprüfen und den Druck freigeben, sofern es den Anforderungen entspricht. Mehr über das Abfangen von internen Wordbefehlen ist in Kapitel 18 beschrieben.

Zusätzlich können die internen Dialogfelder von Word eingeblendet werden. Viele Einstellungen lassen sich vor dem Aktivieren festlegen und anschließend wieder auslesen. Diese Möglichkeiten werden in Kapitel 14 erläutert.

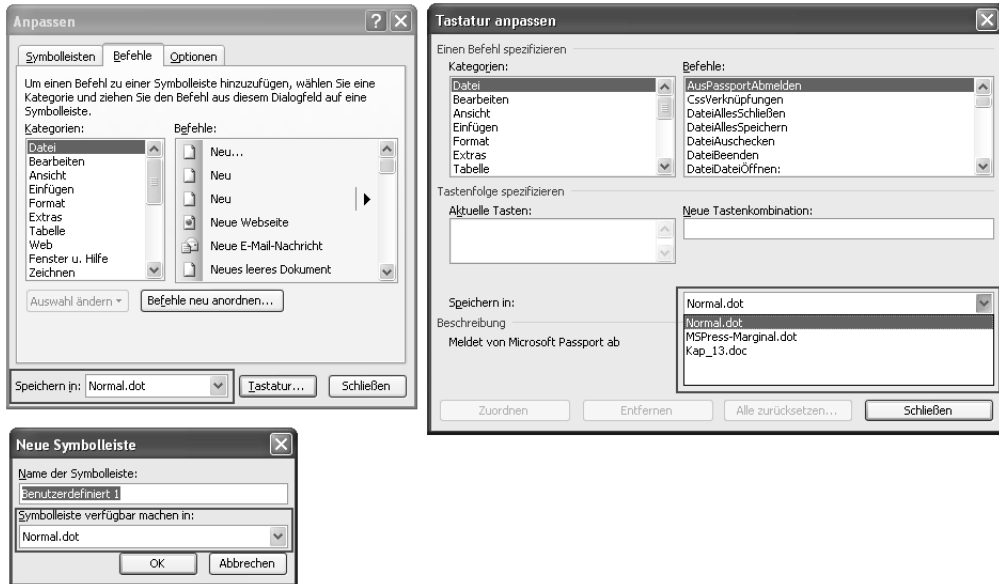
Schließlich kann jeder Menübefehl oder jedes Makro (Public Sub, ohne Argumente) einem beliebigen Tastaturkürzel zugewiesen werden (Menübefehl *Extras/Anpassen/Tastatur* in Word oder über die KeyBindings-Auflistung des Objektmodells, worüber in Kapitel 16 die Rede ist).

Dieses Kapitel zeigt auf, wie diese Vielfalt an Anpassungen zu organisieren ist, um Konflikte zu vermeiden. Damit werden die richtigen Werkzeuge zum gegebenen Zeitpunkt dem Anwender zur Verfügung stehen.

## Der Speicherort einer Anpassung

Im Gegensatz zu Excel oder PowerPoint bietet Word mehrere Speicherorte für Anpassungen, den so genannten *Kontext*. Der Kontext wurde bereits mehrmals im Buch erwähnt, meist in Verbindung mit der Benutzerschnittstelle – so zum Beispiel in Kapitel 1 bei der Diskussion über Speicherorte für Makros. Die Dialogfelder in *Extras/Anpassen* verfügen über eine *Speichern in*-Dropdownliste (Abbildung 13.1), welche alle momentan gültigen Speicherorte enthält. Grundsätzlich stehen die *Normal.dot* sowie das aktuelle Dokument immer zur Verfügung. Weiterhin werden alle geladenen Vorlagen-Add-Ins aufgeführt sowie die Dokumentvorlage des aktuellen Dokuments, sofern es sich bei diesem nicht selbst um eine Dokumentvorlage handelt.

Abbildg. 13.1 Bei Anpassungen sollte die Auswahl für *Speichern in* immer genau kontrolliert werden.



Makros, Symbolleisten und Tastenkombinationen können sowohl in einem Dokument als auch einer Dokumentvorlage gespeichert werden. AutoText-Einträge dagegen lassen sich ausschließlich in Dokumentvorlagen ablegen. Der Speicherort, also die eigentliche Datei, dieser vier Elemente stellt den *Kontext* der Anpassung dar.

Custo-  
mization  
Context

Im Word-Objektmodell wird dieser Speicherort über die Eigenschaft `CustomizationContext` des `Application`-Objekts festgelegt. Vergisst der Entwickler, diese vor der Erstellung, Änderung oder Entfernung eines solchen Elements anzugeben, und überlässt Word die Entscheidung, wird der Speicherort für die Anpassung zur Lotterie. Noch schlimmer: Es ist nicht gesagt, dass alle Änderungen in der gleichen Datei vorgenommen werden. Es kommt beispielsweise vor, dass in der einen Datei eine Symbolleiste erstellt wird, das Löschen derselben jedoch in einer anderen Datei erfolgt (mit der Folge, dass die Symbolleiste gar nicht gelöscht wird, was verständlicherweise sehr verwirrend ist).

Die Eigenschaftssyntax sieht wie folgt aus:

```
Application.CustomizationContext = [Dokument- oder Vorlagenobjekt]
```

Die Syntax für C# oder eine andere Programmierungsumgebung bleibt die gleiche. Jedoch wird eine Variable benötigt, der das `Application`-Objekt zugewiesen wurde:

```
wdApp.CustomizationContext = doc
```

Um die Anpassung in der standardmäßigen globalen Vorlage *Normal.dot* zu speichern:

```
Application.CustomizationContext = NormalTemplate
```

Um sie im aktuellen Dokument zu speichern:

```
Application.CustomizationContext = ActiveDocument
```

Um sie in der Vorlage des aktuellen Dokuments zu speichern:

```
Application.CustomizationContext = ActiveDocument.AttachedTemplate
```

Um sie in einem beliebigen geöffneten Dokument zu speichern:

```
Dim doc as Word.Document  
Dim s as String  
s = "DokName.doc"  
Set doc = Application.Documents(s)  
Application.CustomizationContext = doc
```

Um sie in einer anderen, geladenen globalen Vorlage (Add-In-Vorlage) zu speichern:

```
Dim templ as Word.Template  
Dim s as String  
Dim vKontext As Variant  
'Am Schluss soll der gegenwärtige Kontext wieder hergestellt werden, so dass weitere  
'Anpassungen nicht versehentlich in unserem Kontext ausgeführt werden.  
'Da dieser ein Dokument oder eine Vorlage sein könnte, muss die Objektvariable  
'vom Typ Variant sein.  
Set vKontext = Application.CustomizationContext  
s = "AddinName.dot"  
s = Application.Addins(AddinName).Path & "\" & AddinName  
Set templ = Application.Templates(s)  
Application.CustomizationContext = templ  
'Anpassungen hier vornehmen  
Application.CustomizationContext = vKontext
```

---

**WICHTIG** Wenn die Anpassungen im Speicherort erhalten bleiben sollen, muss diese Datei gespeichert werden. Dies wird, wie üblich, mit der Save-Methode ausgeführt.

Sind die Anpassungen temporärer Natur und sollen beim Schließen des Kontextes wegfallen, kann auf diesen Schritt verzichtet werden. Trotzdem muss gehandelt werden, weil der Anwender sonst aufgefordert wird, die (ihm nicht bekannten) Änderungen zu speichern (»Wollen Sie Änderungen in [Dateiname] speichern?«). Durch Festlegen der Saved-Eigenschaft wird Word angewiesen, alle Änderungen seit dem letzten Speichervorgang bis zum aktuellen Zeitpunkt zu ignorieren:

```
templ.Saved = True
```

---

# Auswahl des Speicherorts

Nachdem gezeigt wurde, welche unterschiedlichen Speicherorte für die verschiedenen Anpassungen verwendet werden können, soll nun geklärt werden, welches der »richtige« Speicherort für die einzelnen Anpassungen ist.

Dies ist in erster Linie davon abhängig, in welchen Bereichen die entsprechende Funktionalität dem Anwender zur Verfügung stehen soll. Eine Anpassung kann entweder global innerhalb der ganzen Word-Umgebung oder in Dokumenten, die auf der gleichen Dokumentvorlage basieren, oder in einem einzelnen Dokument zur Verfügung gestellt werden.

Grundsätzlich gelten die gleichen Spielregeln, wie sie in Kapitel 1 im Abschnitt über den geeigneten Speicherort eines Makros empfohlen wurden:

- Die *Normal.dot* gehört grundsätzlich dem Anwender. Hier haben Sie, als »Fremder«, nichts zu suchen.

---

**HINWEIS** Wenn Sie trotzdem die *Normal.dot* in Betracht ziehen, bedenken Sie, dass die Gefahr einer Dokumentbeschädigung dieser Vorlage relativ groß ist. Wird sie beschädigt, muss sie vernichtet und ersetzt werden, weil Word sonst instabil wird. Alle Anpassungen würden verloren gehen.

---

- Sollen Symbolleisten, Tastaturbelegungen, AutoText oder Makros immer zur Verfügung stehen, gehören sie in ein Vorlagen-Add-In. Add-Ins, die sich im *StartUp*-Ordner befinden, werden automatisch beim Starten von Word geladen.
- Anpassungen, die bei der Arbeit mit einer bestimmten Art von Dokumenten verfügbar sein sollen, gehören in die zugehörige Dokumentvorlage. Sie werden somit in jedem Dokument, das auf der entsprechenden Dokumentvorlage basiert, zur Verfügung stehen. Dies funktioniert jedoch nur, solange die Vorlage auffindbar ist. Sie muss also im gleichen Pfad bleiben oder sich in einem von Word standardmäßig durchsuchten Verzeichnis befinden.

---

**HINWEIS** Dokumente erben nur Formatvorlagen und Text von der Dokumentvorlage; alles andere steht ihnen nur durch eine dynamische Verknüpfung zur Vorlage zur Verfügung.

---

- Änderungen zur Benutzerschnittstelle, die nur in einem einzigen Dokument zur Verfügung stehen sollen, müssen in diesem Dokument gespeichert werden. Wird beispielsweise ein Formular per E-Mail an verschiedene Personen gesandt, bleiben die Anpassungen und Werkzeuge nur dann erhalten, wenn sie in diesem Dokument gespeichert sind.

## Kontext für COM-Add-Ins

Beim Erstellen eines COM-Add-Ins muss der Kontext für die Symbolleisten und Symbolschaltflächen ebenfalls festgelegt werden.

Sollen die Werkzeuge des COM-Add-Ins immer zur Verfügung stehen, ohne dass ein Zugriff auf ein eigenes Vorlagen-Add-In (*.dot*) zur Verfügung steht, müssen die Anpassungen in der *Normal.dot* gespeichert werden. Werden einige Verhaltensregeln eingehalten, ist dies durchaus vertretbar:

- Beim Erzeugen der Anpassungen muss der Kontext explizit auf `NormalTemplate` festgelegt werden.
- Die Änderungen müssen beim Entladen des COM-Add-Ins wieder entfernt werden. In diesem Fall muss der Speicherort – `NormalTemplate` – wiederum explizit festgelegt werden.
- Dieser Vorgang soll für den Benutzer möglichst transparent sein. Das heißt, er soll nicht mit einer Aufforderung, ob die Änderungen zu speichern sind, gestört werden. Dies wird erreicht, indem die Vorlage explizit gespeichert oder die `Saved`-Eigenschaft auf »Wahr« festgelegt wird.

**WICHTIG** Auch wenn durch Verwendung der `Saved`-Eigenschaft die explizite Entfernung der Anpassungen theoretisch entfallen würde, muss dieser Vorgang trotzdem durchgeführt werden. Es ist nicht auszuschließen, dass die *Normal.dot* inzwischen durch den Anwender oder ein anderes Add-In gespeichert wird.

Vergessen Sie bitte nie: Ihr Add-In ist wahrscheinlich nicht das einzige vorhandene. Konflikte sind möglichst zu vermeiden. Es ist nicht möglich, festzulegen, in welcher Reihenfolge Add-Ins geladen werden. Seien Sie bitte höflich und entfernen Sie keine Anpassungen Anderer (inklusive des Benutzers). Bedenken Sie: Ihr Add-In ist ein globales Werkzeug. Wäre es nur für eine Dokument-spezifische Aufgabe vorgesehen, könnte argumentiert werden, es wäre besser, wenn nur diese oder jene Symbolleisten oder Menüpunkte sichtbar wären. Dies ist aber bei einem *globalen* Add-In nicht der Fall.

Falls Sie ein COM-Add-In für eine Dokument-spezifische Aufgabe programmieren und Ihnen keine eigens dafür vorgesehene Vorlage zur Verfügung steht, fügen Sie nur eine Schaltfläche permanent in die *Normal.dot* ein. Diese Schaltfläche wird das COM-Add-In aufrufen, das zu diesem Zeitpunkt die weiteren Schnittstellen und Anpassungen vornimmt. Beim Entladen sind diese wieder zu entfernen.

## Hierarchie der Anpassungen

Mit so vielen Kontexten, die gleichzeitig aktiv sind, ist es wichtig, die Rangordnung für gleichnamige Anpassungen zu verstehen. Prinzipiell gelten die gleichen Regeln wie jene, die in Kapitel 1 für gleichnamige Makros vorgestellt wurden.

Höchste Priorität haben die Anpassungen im aktuellen Dokument. Wurde beispielsweise die Tastenkombination `Alt+I` in der *Normal.dot* sowie im aktuellen Dokument zwei verschiedenen Befehlen zugewiesen, hat die Zuweisung im aktuellen Dokument Vorrang.

An zweiter Stelle stehen die Anpassungen aus der verbundenen Dokumentvorlage des aktuellen Dokuments. Enthalten diese Dokumentvorlage und die *Normal.dot* Symbolleisten gleichen Namens, erscheint diejenige Symbolleiste am Bildschirm, die in der Dokumentvorlage angelegt wurde. Dies gilt nur, solange das auf dieser Dokumentvorlage basierende Dokument auf dem Bildschirm aktiv ist.

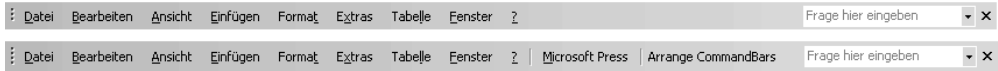
An letzter Stelle folgen die Anpassungen in geladenen Add-In-Vorlagen. Anpassungen der *Normal.dot* kommen nur zur Geltung, wenn in keinem anderen Kontext eine gleichnamige Anpassung vorhanden ist.

Einträge in Menüs der Menüleiste verhalten sich ein wenig anders. Eine Menüleiste wird durch die Menüleiste eines anderen Kontextes nicht ersetzt, sondern ergänzt. Die Abbildung 13.2 veranschaulicht dieses Prinzip. Oben steht die unveränderte Menüleiste der *Normal.dot*. Zum Vergleich befin-



den sich in der darunter stehenden zwei zusätzliche Menüpunkte; der eine stammt aus der angehängten Vorlage des aktuellen Dokuments, der andere aus einer, als Add-In geladenen globalen Vorlage.

**Abbildg. 13.2** Menüeinträge aus angehängten Vorlagen sowie Add-Ins ergänzen die standardmäßige Symbolleiste der *Normal.dot*.



Nur wenn zwei Menüeinträge die gleiche Beschriftung haben, kommt die oben beschriebene Rangordnung zum Zug.

#### WICHTIG

Falls es einmal vorkommt, dass ein Menüeintrag in der Vorlage vorhanden, aber auf einer bestimmten Installation nicht sichtbar ist, könnte ein Konflikt mit einer Anpassung in *Normal.dot* die Ursache sein.

Wurde auf dieser Installation irgendwann eine Symbolschaltfläche mit der gleichen Beschriftung erstellt und danach ausdrücklich gelöscht, erinnert sich die Vorlage möglicherweise daran. Wird einem Menü später, durch einen anderen Kontext, ein Eintrag mit gleicher Beschriftung hinzugefügt, könnte er von der Vorlage deshalb unterdrückt werden. In diesem Fall muss der fehlbare Behälter (Vorlage) neu erstellt oder die Beschriftung des Menüpunkts geändert werden.

## Zusammenfassung

In diesem Kapitel wurde die Problematik des Kontexts, also des Speicherorts für Anpassungen besprochen. Es wurden die folgenden Punkte aufgezeigt:

- Alle Anpassungen müssen gezielt an einem bestimmten Ort gespeichert werden. Dazu ist die CustomizationContext-Eigenschaft zuständig (Seite 600).
- Die bewusste Wahl des Speicherorts legt die Sichtbarkeit der Anpassung fest (Seite 603).
- Bei so genannten COM-Add-Ins muss ein »externer« Speicherort verwendet werden, damit die erzeugten Anpassungen abgelegt werden können. Diese Änderungen sollten nur temporär erzeugt werden (Seite 603).
- Sind Anpassungen aus mehreren Kontexten gleichzeitig vorhanden, wird hierarchisch entschieden, welche sichtbar sind (Seite 604).



## Kapitel 14

# Mit Dialogfeldern arbeiten

### In diesem Kapitel:

Benutzerdefinierte Dialogfelder	608
Interne Dialogfelder	631
<i>FileDialog-Objekt</i>	638
Zusammenfassung	648

Benötigt ein Makro Informationen, die zur Laufzeit durch den Anwender eingegeben oder ausgewählt werden müssen, so muss eine Schnittstelle zwischen dem Programm und dem Anwender definiert werden.

Eine Möglichkeit besteht darin, die beiden von VBA zur Verfügung gestellten Funktionen `InputBox` und `MsgBox` zu verwenden. Die Interaktion mit dem Anwender bei der Nutzung dieser beiden Funktionen ist jedoch sehr beschränkt und eignet sich nur für einfache Ein- bzw. Ausgaben. Dieses Kapitel soll aufzeigen, wie eine komplexere Kommunikation mit dem Anwender aufgebaut werden kann und welche Möglichkeiten dazu zur Verfügung stehen:

- Der Abschnitt »Benutzerdefinierte Dialogfelder« behandelt die Möglichkeiten, um eigene Dialogfelder zu erstellen. Wobei hier alle Anforderungen an die Schnittstelle manuell erstellt werden müssen.
- Wie ein Zugriff auf die internen Dialogfelder von Word erfolgen kann, wird im Abschnitt »Interne Dialogfelder« aufgezeigt.
- Im Abschnitt »FileDialog-Objekt« wird erläutert, welche internen Dialogfelder zusätzlich im Zusammenhang mit dem Dateisystem zur Verfügung stehen.

## Benutzerdefinierte Dialogfelder

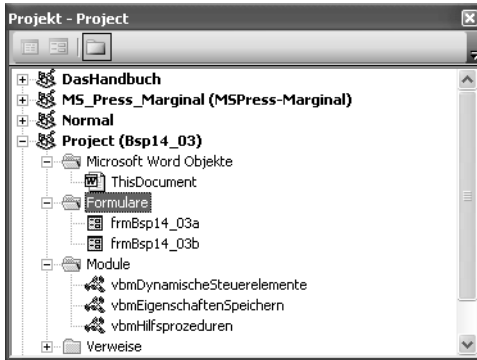


Für benutzerdefinierte Dialogfelder steht die Klasse `UserForm` aus der Bibliothek `MSForms` zur Verfügung. Mit dieser Klasse können Dialogfelder erstellt werden, welche die Schnittstelle zwischen dem eigentlichen Programm und dem Anwender bilden. Alle benötigten Eingabewerte für ein Makro werden in einem solchen Dialogfeld festgelegt und kontrolliert.

Für die Interaktion mit dem Anwender werden standardmäßig die wichtigsten Komponenten zur Verfügung gestellt. Dazu gehören unter anderem:

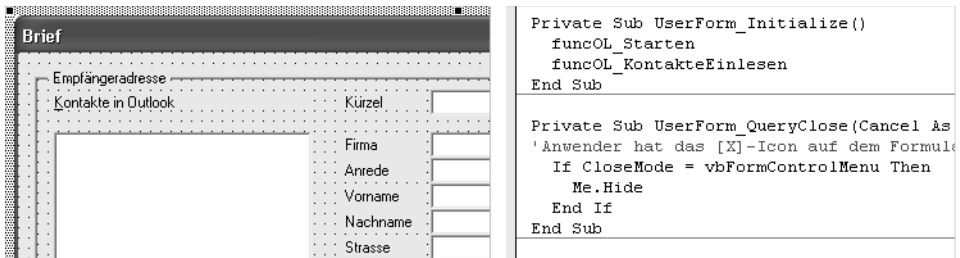
- Das *Textfeld* zur Eingabe von Text über die Tastatur.
- Das *Kontrollkästchen* zum Aktivieren bzw. Deaktivieren einer einzelnen Option.
- Das *Optionsfeld* zum Auswählen einer einzelnen Option aus einer Gruppe von möglichen Werten.
- Das *Listefeld* für die Auswahl von einzelnen oder mehreren Werten aus einer Menge von Werten, die, je nach Anwendung, sogar teilweise dynamisch ermittelt werden.
- Das benutzerdefinierte Dialogfeld wird, wie dies in Abbildung 14.1 ersichtlich ist, wie der eigentliche Programmcode selber, zusammen als Projekt in einem Dokument oder einer Dokumentvorlage abgespeichert.
- Ein `UserForm`-Objekt besteht eigentlich aus zwei Teilen. Der eine Teil ist das eigentliche Objekt. Hier werden wie mit einem Zeichnungsprogramm alle benötigten Komponenten eingefügt. Im so genannten Entwicklungsmodus werden für das `UserForm`-Objekt sowie alle eingefügten Komponenten die Standardeigenschaften gesetzt.
- Der zweite Teil ist das zugehörige Code-Fenster. In diesem Bereich werden alle nötigen Prozeduren und Funktionen hinterlegt, die für das einwandfreie Funktionieren des Dialogfeldes benötigt werden. Dazu gehören unter anderem die Logik und Abhängigkeiten zwischen den einzelnen Komponenten, sowie die Programmzeilen zum Überprüfen der Eingaben.

Abbildg. 14.1 Das *UserForm*-Objekt wird im Zweig *Formulare* zusammen mit dem Projekt gespeichert.



- Prozeduren und Funktionen, die in verschiedenen *UserForm*-Objekten zum Einsatz kommen, sollten in ein normales Modul ausgelagert und mittels Ein- und Ausgabeargumenten so aufgebaut werden, dass diese allgemein verwendet werden können. Dies bedeutet, dass eigentlich nur die allernötigsten Programmzeilen innerhalb des Klassenmoduls des *UserForm*-Objekts hinterlegt werden.

Abbildg. 14.2 Das *UserForm*-Objekt im Entwicklungsmodus (links) und das zugehörige Code-Fenster (rechts)



## UserForm in verschiedenen Projekten nutzen

- Das Erstellen eines benutzerdefiniertes Dialogfelds ist mit einem nicht zu unterschätzenden zeitlichen Aufwand verbunden. Denn das eigentliche Design der Bildschirmmaske ist der kleinere Teil. Komplexer hingegen wird das Umsetzen der entsprechenden Logik zwischen den verschiedenen Komponenten und das Überprüfen der Eingaben, denn diese Schritte müssen, wie bereits erwähnt, manuell erstellt werden.

### Export und Import

- Damit nicht in jedem neuen Projekt ein bestehendes *UserForm*-Objekt komplett neu erstellt werden muss, kann dieses exportiert und im neuen Projekt importiert werden. Das Exportieren und Importieren eines *UserForm*-Objekts wurde bereits in Kapitel 2 beschrieben.
- Durch das Exportieren des Formulars werden zwei Dateien auf der Festplatte der Arbeitsstation erstellt. Bei der *.frm*-Datei handelt es sich um eine Textdatei, welche die einzelnen Programmzei-

len enthält. Die zweite Datei trägt die Erweiterung *.frx*. In dieser Datei werden binäre Informationen zu den eingebunden Komponenten (OLE-Objekte) hinterlegt.

**HINWEIS** Ein UserForm-Objekt kann nur dann erfolgreich in ein Projekt importiert werden, wenn die beiden zusammengehörigen Dateien im gleichen Ordner vorhanden sind.

### Speichern in gemeinsamer Programmbibliothek

- Das benutzerdefinierte Dialogfeld kann auch an einer zentralen Stelle abgespeichert werden. In Kapitel 9 wurde gezeigt, wie Prozeduren und Funktionen in einem gemeinsam genutzten Add-In angelegt werden können. In einem solchen Add-In können auch UserForm-Objekte abgespeichert werden. Wie mit diesen gemeinsamen genutzten Objekten gearbeitet wird, wurde ebenfalls in Kapitel 9 beschreiben.

## Das *UserForm*-Objekt

Das UserForm-Objekt verfügt über verschiedene Eigenschaften, Methoden und Ereignisse. Einige wichtige davon sollen in einer kurzen Übersicht aufgezeigt und besprochen werden.

### Eigenschaften

Alle Eigenschaften können während der Entwurfszeit direkt im Eigenschaftenfenster des Dialogfelds eingetragen oder während der Laufzeit innerhalb des Programms dynamisch gesetzt werden.

**Abbildg. 14.3** Tragen Sie die Eigenschaften des UserForms zur Entwurfszeit im Eigenschaftenfenster ein.



**Name** Jedes benutzerdefinierte Dialogfeld muss innerhalb des VBA-Projekts eindeutig bezeichnet werden. Diese Bezeichnung wird in der Name-Eigenschaft eingetragen. Beim Anlegen eines neuen UserForm-Objekts wird als Vorgabewert der Name *UserForm1* eingetragen, wobei die einzelnen UserForm-Objekte der Reihe nach durchnummeriert werden.

Normalerweise wird der Anwender an keiner Stelle des Programms mit der Name-Eigenschaft konfrontiert. Aus diesem Grunde kann dieser Wert auch eine technische Bezeichnung oder eine Abkürzung enthalten (beispielsweise »frmBriefErfassen«).

Als Präfix zum eigentlichen Namen des Dialogfelds wird normalerweise »frm« verwendet.

**HINWEIS** Wie bereits in Kapitel 3 erwähnt, sollte bei der Benennung von Variablen eine spezielle Namenskonvention eingehalten werden. Als logische Konsequenz macht es durchaus Sinn, wenn diese Namensgebung auch auf die UserForm-Objekte und deren zugehörigen Komponenten ausgeweitet wird. Einen unverbindlichen Vorschlag einer solchen Namenskonvention finden Sie im Anhang A.

**Caption** In der Caption-Eigenschaft wird die Bezeichnung des UserForm-Objekts hinterlegt, welcher in der Titelleiste des Dialogfeldes dargestellt wird. Der Eintrag dient dem Anwender zur Erkennung der einzelnen Dialogfelder und sollte somit mit einem aussagekräftigen Titel versehen werden.

**Left** Anhand der Left-, Top, Height- und Width-Eigenschaften kann die eigentliche Größe und Position des UserForm-Objekts festgelegt werden. Diese Masse sind statisch und können zur Laufzeit nur durch eine explizite Programmanweisung geändert werden.

**Top**

**Height**

**Width**

**HINWEIS** Ein benutzerdefiniertes Dialogfeld verfügt über kein Systemmenü. Dies hat zur Folge, dass ein Dialogfeld weder minimiert noch maximiert werden kann. Ebenso besteht keine Möglichkeit, die Ausmaße am Bildschirm durch Verschieben der Ränder mit der Maus anzupassen.

**StartUp-Position** Mit der StartUpPosition-Eigenschaft kann festgelegt werden, an welcher Position eine neue Instanz des UserForm-Objekts am Bildschirm angezeigt wird.

Tabelle 14.1 Zusammenstellung der möglichen Startposition des benutzerdefinierten Dialogfelds

Wert	Beschreibung
0 (Manuell)	Die Position entspricht den eingetragenen Werten der <b>Top</b> - und <b>Left</b> -Eigenschaft.
1 (Fenstermitte)	Das Dialogfeld wird auf dem Element zentriert, von wo aus das <b>UserForm</b> -Objekt aufgerufen wurde. In diesem Fall handelt es sich immer um Word selbst.
2 (Bildschirmmitte)	Das Dialogfeld wird innerhalb des ganzen Bildschirms zentriert.
3 (Windows.-Standard)	Die Position befindet sich in der oberen linken Ecke des Bildschirms.

**HINWEIS** Wird die StartUpPosition-Eigenschaft zur Laufzeit festgelegt, müssen die Werte aus Tabelle 14.1 direkt eingetragen werden, da in der zugehörigen Programmbibliothek keine entsprechenden Konstanten deklariert wurden.

**Tag** Die Tag-Eigenschaft dient zum Abspeichern bzw. Zwischenspeichern zusätzlicher Informationen zum aktuellen Dialogfeld, ohne die Einstellungen oder Attribute anderer Eigenschaften zu verändern. Der Inhalt dieser Eigenschaft kann zur Laufzeit ausgewertet oder geändert werden.

Methoden

Alle Methoden werden explizit aus dem Programm heraus aufgerufen. Die Steuerung der Methode erfolgt durch Übergabe von entsprechenden Werten an die zugehörigen Argumente.

**Load** Um ein UserForm am Bildschirm darzustellen, muss diese vorab in den Speicher geladen werden.

**UnLoad** Dieser Schritt kann (wie in Listing 14.1 ersichtlich) durch den Aufruf der Load-Methode erfolgen.

Als Gegenstück dazu dient die `Unload`-Methode. Sie gibt den durch das Dialogfeld belegten Speicher wieder frei. Das Entladen des `UserForm`-Objekts erfolgt, nachdem alle Eingaben verarbeitet und die entsprechenden Werte nicht mehr benötigt werden. Ist ein Dialogfeld während des Aufrufs der `Unload`-Methode immer noch sichtbar, wird dieses automatisch ausgeblendet.

Show  
Hide

Um ein benutzerdefiniertes Dialogfeld am Bildschirm darzustellen, muss dieses mit der `Show`-Methode eingeblendet werden. Wurde das `UserForm`-Objekt noch nicht explizit in den Speicher geladen, wird dies durch den Aufruf der `Show`-Methode automatisch nachgeholt.

**WICHTIG** Ein Dialogfeld kann auf zwei verschiedene Arten am Bildschirm dargestellt werden. Dies lässt sich anhand des Parameters `Modal` beim Aufruf der `Show`-Methode festlegen:

- Das Dialogfeld kann als gebundenes Objekt (`vbModal`) am Bildschirm eingeblendet werden (beispielsweise das Dialogfeld *Optionen*). Solange das Dialogfeld aktiv ist, steht das Programm still und wird nicht weiter verarbeitet. Das Dokument im Hintergrund kann nicht bearbeitet werden:

```
UserForm1.Show vbModal
```

- Die zweite Möglichkeit besteht darin, das Dialogfeld als ungebundenes Objekt (`vbModeless`) am Bildschirm einzublenken (beispielsweise das Dialogfeld *Suchen und Ersetzen*). Unabhängig davon, zu welchem Zeitpunkt das Dialogfeld geschlossen wird, das Programm, und somit auch das Makro, wird weiterverarbeitet. Das parallele Arbeiten mit Word ist ebenfalls möglich:

```
UserForm1.Show vbModeless
```

Als Gegenstück zur `Show`-Methode dient die `Hide`-Methode, welche zum Verbergen eines dargestellten Dialogfelds dient. Nachdem das `UserForm`-Objekt ausgeblendet wurde, verbleibt dieses jedoch im Arbeitsspeicher. Somit können weiterhin alle Eigenschaften des Dialogfelds bzw. dessen Komponenten angesprochen werden.

**Listing 14.1** Laden des *UserForm*-Objekts in den Speicher, darstellen Desselben und anschließendes Entladen

```
Sub Demo_UserformAnzeigen_1()  
    Load frmBsp_01  
    frmBsp_01.Show vbModal  
    Unload frmBsp_01  
End Sub
```

In Listing 14.1 wird das `UserForm`-Objekt *frmBsp\_01* durch den direkten Aufruf der `Load`-Methode in den Speicher geladen und zu einem späteren Zeitpunkt durch den Aufruf der `Unload`-Methode wieder aus dem Speicher entfernt.

**PROFITIPP**

In sämtlichen Kapiteln wurde darauf hingewiesen, dass beim Erfassen der Programmzeilen stets mit Objekten gearbeitet werden soll. Dieser Hinweis hat auch im Zusammenhang mit dem `UserForm`-Objekt seine Gültigkeit. Der Grund dazu ist folgender: Wird das gleiche Dialogfeld aus mehreren Dokumenten heraus aufgerufen, muss für jeden Aufruf eine eigene Instanz des Objekts erzeugt werden, damit die erfassten Daten eindeutig einem Dokument zugeordnet werden können. Deshalb wurden die Programmzeilen in Listing 14.2 entsprechend angepasst.



Listing 14.2 Erzeugen eines eigenen Objekts zum Laden und Entladen des UserForms

```

Sub Demo_UserformAnzeigen_2()
    Dim frm As frmBsp_01

    Set frm = New frmBsp_01
    frm.Show vbModal
    Set frm = Nothing
End Sub

```

Das Laden des Objekts in den Arbeitsspeicher erfolgt beim Anlegen der neuen Instanz auf das betreffende UserForm-Objekt:

```
Set frm = New frmBsp_01
```

Das Entladen des Objekts erfolgt beim Freigeben der entsprechenden Instanz auf das betreffende UserForm-Objekt:

```
Set frm = Nothing
```

**Move** Anhand der Move-Methode kann das Dialogfeld an eine andere Bildschirmposition verschoben werden. Beim Verschieben des Objekts können mit einer Anweisung alle vier Ausdehnungen in einem Schritt modifiziert werden (Links, Oben, Breite und Höhe).

**HINWEIS** Obwohl die Parameter zur Move-Methode optional sind, werden keine benannten Argumente unterstützt. Dies bedingt, dass der Reihe nach alle Werte übergeben werden müssen, auch wenn einzelne Ausdehnungen des Dialogfelds nicht modifiziert werden müssen.

Die nachstehende Zeile bewirkt somit, dass die Left-Eigenschaft auf den Wert 100 und die Width-Eigenschaft auf den Wert 500 gesetzt wird. Da keine benannten Argumente unterstützt werden, muss die Top-Eigenschaft trotzdem übergeben werden. Als »neuer« Wert wird der aktuelle Wert (frm.Top) eingetragen:

```
frm.Move 100, frm.Top, 500
```

**Repaint** Durch den Aufruf der Repaint-Methode wird das benutzerdefinierte Dialogfeld am Bildschirm neu gezeichnet. Dieser Schritt wird dann benötigt, wenn zur Laufzeit ein am Bildschirm aktives Dialogfeld in der Darstellung modifiziert wird (beispielsweise die Caption-Eigenschaft eines Label-Objekts wird geändert).

**HINWEIS** Wird die Repaint-Methode innerhalb einer Schleife und in kurzen Abständen mehrmals aufgerufen, beginnt das UserForm am Bildschirm zu flackern. Dieses Flackern steht in direkten Zusammenhang mit dem mehrmaligen Neuzeichnen des Dialogfeldes am Bildschirm.

Diese störende Auswirkung am Bildschirm kann nicht behoben werden. Es besteht jedoch die Möglichkeit, nur einen Teil des Dialogfeldes neu zu zeichnen, sofern dieses mittels eines Rahmens (Frame) unterteilt wurde. So kann anstelle von UserForm1.Repaint nur der Rahmen mittels Frame1.Repaint neu gezeichnet werden. Dies würde das Flackern auf den entsprechenden Teil des Dialogfelds einschränken.

**Abbildg. 14.4** Flackerndes Dialogfeld, »eingefangen« während des *Repaint*-Vorgangs (links flackert das ganze Dialogfeld, rechts nur der Bereich des Rahmens)



Ein Beispiel, welches das Problem mit dem flackernden Bildschirm aufzeigt, finden Sie in der Beispieldatei *Bsp14\_01.doc* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap14*.

## Ereignisse

Die einzelnen Programmschritte, die beim Auftreten eines Ereignisses ausgeführt werden, müssen in jeden Fall manuell beim entsprechenden Ereignis hinterlegt werden.

Initiali-  
ze  
Termini-  
ng

Jede Klasse verfügt über eine Initialisierungs- bzw. Terminierungssequenz, dies ist auch bei jedem UserForm-Objekt der Fall. Das Initialize-Ereignis tritt ein, wenn eine neue Instanz der Klasse erzeugt wird.

Während der Initialisierung werden in erster Linie Programmsequenzen abgearbeitet, die das benutzerdefinierte Dialogfeld abschließend konfigurieren, denn nicht alle benötigten Eigenschaften können zur Entwurfszeit abschließend gesetzt werden. Dazu gehören unter anderem:

- Eintragen von abgespeicherten Werten der einzelnen Komponenten, um den Zustand des Dialogfelds wie beim letzten Aufruf wieder herzustellen (beispielsweise Position am Bildschirm, Status der Kontrollkästchen usw.). Ein entsprechendes Beispiel ist im Abschnitt »Setzen von Eigenschaften« in diesem Kapitel aufgeführt.
- Befüllen von Listefeldern mit Werten aus externen Datenquellen (beispielsweise aus einer Textdatei). Das zugehörige Beispiel ist im Abschnitt »

### HINWEIS

In Abbildung 14.14 ist ersichtlich, wie in der *Ini*-Datei *Bsp14\_03.ini* die beiden Einträge *Label3.Caption* und *Label3.Accelerator* auskommentiert wurden. Dies hat zur Folge, dass im Dialogfeld die Standardwerte, die während der Entwicklungszeit gesetzt wurden, dargestellt werden.

- Eigenschaften in *.txt*-Dateien auslagern« in diesem Kapitel beschrieben.
- Aktivieren bzw. deaktivieren von Komponenten in Abhängigkeit zu anderen Komponenten.

Das Terminate-Ereignis entspricht dem Gegenteil. Dieses Ereignis tritt ein, wenn die entsprechende Instanz der Klasse zerstört wird. Während der Terminierung werden vor allem Programmzeilen zum Zwischenspeichern des Status des UserForm-Objekts eingefügt.

Activate  
Deacti-  
vate  
Layout

Im Zusammenhang mit der Bildschirmdarstellung treten drei Ereignisse in den Vordergrund. Das Activate-Ereignis tritt ein, wenn das Dialogfeld aktiviert wird und den so genannten Fokus bekommt. Dies ist bei dessen erster Anzeige sowie beim Wechseln zwischen zwei benutzerdefinierten Dialogfeldern der Fall.

Wird zwischen zwei eigenständigen UserForm-Objekten hin und her gewechselt, so tritt im ersten UserForm-Objekt (dasjenige, das den Fokus abgibt) das Deactivate-Ereignis ein.

Wird das Dialogfeld am Bildschirm an eine neue Position verschoben, muss dieses neu gezeichnet werden. Dies bedeutet, dass das Layout-Ereignis eintritt.

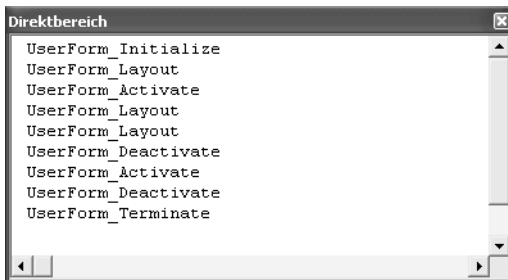
**HINWEIS** Die einfachste Art, um herauszufinden, in welcher Reihenfolge die einzelnen Ereignisse ausgeführt werden, ist in Listing 14.3 dargestellt. Hier wurde jedem Ereignis, das von Interesse war, eine kleine Programmsequenz hinterlegt. Die `Debug.Print`-Anweisung protokolliert jedes Auftreten des Ereignisses im Direktbereich der Programmierungsumgebung, wie dies in Abbildung 14.5 ersichtlich ist.

Um den Direktbereich als Fenster innerhalb der Programmierungsumgebung anzuzeigen, rufen Sie den Menübefehl *Ansicht/Direktfenster* auf.

**Listing 14.3** Hinterlegen einer Programmsequenz, die das Auftreten der Ereignisse protokolliert

```
Private Sub UserForm_Activate()
    Debug.Print "UserForm_Activate"
End Sub
Private Sub UserForm_Deactivate()
    Debug.Print "UserForm_Deactivate"
End Sub
Private Sub UserForm_Layout()
    Debug.Print "UserForm_Layout"
End Sub
Private Sub UserForm_Initialize()
    Debug.Print "UserForm_Initialize"
End Sub
Private Sub UserForm_Terminate()
    Debug.Print "UserForm_Terminate"
End Sub
```

**Abbildg. 14.5** Das Resultat der Protokollierung kann im Direktbereich analysiert werden.



Query-  
Close

Das `QueryClose`-Ereignis tritt auf, bevor das Dialogfeld geschlossen wird. Dieses Ereignis dient zum Überprüfen der erfassten Daten im Dialogfeld. Wurden die Daten noch nicht gespeichert, kann beispielsweise das Schließen des Dialogfelds verhindert werden.

Dem Ereignis sind zwei Argumente zugeordnet. Wird das erste Argument (`Cancel`) auf `True` gesetzt, so wird das `QueryClose`-Ereignis für alle geladenen UserForm-Objekte unterbrochen.

Anhand des Werts des `CloseMode`-Arguments kann ausgewertet werden, aus welchem Grund das Ereignis überhaupt eingetreten ist. Die möglichen Werte, die dieser Parameter annehmen kann, und deren Bedeutung sind in Tabelle 14.2 zusammengestellt.

**Tabelle 14.2** Zusammenstellung der möglichen Werte des `CloseMode`-Arguments

Bezeichnung	Wert	Bedeutung
<code>vbFormControlMenu</code>	0	Der Benutzer hat auf dem UserForm im Systemmenü den Befehl <i>Schließen</i> gewählt.
<code>vbFormCode</code>	1	Innerhalb des Programms wurde die <b>Un</b> load-Anweisung aufgerufen.
<code>vbAppWindows</code>	2	Die aktuelle Windows-Umgebung wird beendet.
<code>vbAppTaskManager</code>	3	Die Anwendung wird vom Windows Task-Manager geschlossen.

In Listing 14.4 wird als mögliches Beispiel gezeigt, wie verhindert werden kann, dass der Anwender durch Betätigen des in der Titelleiste vorhandenen Schließen-Symbols das benutzerdefinierte Dialogfeld geschlossen wird.

**Listing 14.4** Das Dialogfeld kann nicht mehr über das Systemmenü verlassen werden.

```
Private Sub UserForm_QueryClose(Cancel As Integer, CloseMode As Integer)
    'Anwender hat das [X]-Symbol auf dem Formular betätigt.
    If CloseMode = vbFormControlMenu Then
        MsgBox "Schließen via [x]-Icon ist verboten."
        Cancel = True
    End If
End Sub
```



Das dargestellte Beispiel finden Sie in der Beispieldatei *Bsp14\_01.doc* auf der CD-ROM zum Buch im Ordner `\Beispiele\Kap14`.

### Die *UserForm*-Auflistung

Die *UserForms*-Auflistung enthält alle Instanzen von *UserForm*-Objekten. Dies bedeutet, dass alle geladenen Formulare darin enthalten sind und nicht alle Dialogfelder, die einem Projekt zur Verfügung stehen. Somit können mittels einer Schleife alle sich im Arbeitsspeicher befindliche *UserForm*-Objekte angesprochen werden.

Anhand des Beispiels in Listing 14.5 werden alle geladenen benutzerdefinierten Dialogfelder angesprochen und entladen. Dabei spielt es keine Rolle, ob die einzelnen Dialogfelder am Bildschirm sichtbar oder mittels der `Hide`-Methode versteckt wurden.

**Listing 14.5** Entladen aller *UserForm*-Objekte aus dem Arbeitsspeicher

```
Sub Demo_UserformAlleEntladen()
    Dim intZähler As Integer

    For intZähler = UserForms.Count - 1 To 0 Step -1
```

Listing 14.5 Entladen aller *UserForm*-Objekte aus dem Arbeitsspeicher (Fortsetzung)

```

Unload UserForms(intZähler)
Next intZähler
End Sub

```

**HINWEIS** Damit die Schleife mindestens einmal durchlaufen wird, muss sich mindestens ein *UserForm*-Objekt im Arbeitsspeicher befinden. Dies kann anhand des kleinen Makros *Demo\_UserformAnzeigen*, das sich ebenfalls im Modul *vbmUserforms\_Auflistung* befindet, erreicht werden.

### *UserForm*-Objekt als Argument an eine Prozedur übergeben

Wie bereits im Abschnitt »Methoden« in diesem Kapitel aufgezeigt, können von einem *UserForm*-Objekt mehrere Instanzen im Arbeitsspeicher vorhanden sein. Wird eine allgemeine Prozedur entwickelt, die auf das *UserForm*-Objekt zugreift, muss der Programmsequenz die entsprechende Instanz des Dialogfelds als Argument übergeben werden.

In Listing 14.6 ist ein Fragment aus dem allgemeinen Programm und die Prozedur zur Bearbeitung einer bestimmten Instanz des *UserForm*-Objekts dargestellt.

Listing 14.6 *UserForm* als Parameter an eine Prozedur übertragen

```

frmA.Show vbModeless
frmB.Show vbModeless
Demo_UserformAnpassen frmA, "Das ist frmA", 100, 100
Demo_UserformAnpassen frmB, "Das ist frmB", 120, 400

Sub Demo_UserformAnpassen( _
    ByVal frm As frmEreignisse2, _
    ByVal txtCaption As String, _
    ByVal lngTop As Long, _
    ByVal lngLeft As Long)

    With frm
        .Caption = txtCaption
        .Top = lngTop
        .Left = lngLeft
    End With
End Sub

```

**HINWEIS** Mit der Prozedur in Listing 14.6 können nur Dialogfelder der Klasse *frmEreignisse2* bearbeitet werden. Sollte die Prozedur weitere Klassen unterstützen, muss das entsprechende Argument vom Datentyp *Object* definiert werden. Der Datentyp *Object* wurde bereits in Kapitel 3 vorgestellt.

```

Sub Demo_UserformAnpassen(ByVal frm As Object, ByVal txtCaption As String, _
    ByVal lngTop As Long, ByVal lngLeft As Long)

```



Das dargestellte Beispiel finden Sie in der Beispieldatei *Bsp14\_01.doc* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap14*.

## Steuerelemente einbinden

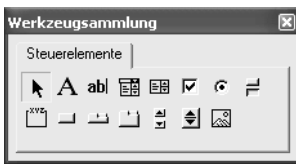
Das UserForm-Objekt ist ein so genanntes Container-Objekt. Dies bedeutet, dass es weitere Objekte aufnehmen und darstellen kann. Diese Objekte werden Steuerelemente genannt, denn damit kann der Anwender das Programm über ein Dialogfeld steuern.

Zum Umfang von VBA gehört die »Microsoft Forms 2.0 Object Library«, welche die wichtigsten Steuerelemente zur Verfügung stellt. Grundsätzlich ist es jedoch möglich, weitere Steuerelemente einzubinden, sofern diese als ActiveX-Komponente zur Verfügung steht.

### Microsoft Forms 2.0 Object Library

Wie in Abbildung 14.6 ersichtlich, werden von der Microsoft Forms 2.0 Object Library vierzehn unterschiedliche Steuerelemente zur Verfügung gestellt. Diese sollen kurz der Reihe nach erläutert werden.

Abbildg. 14.6 Die Standardsteuerelemente aus der Microsoft Forms 2.0 Object Library



**A**

Das *Bezeichnungsfeld* (Label) dient zum Beschriften von einzelnen Steuerelementen, die über keine eigene Bezeichnung (Caption) verfügen, oder zur Ausgabe allgemeiner Informationen auf dem Dialogfeld.

**abl**


Das *Textfeld* (TextBox) stellt dem Anwender eine Schnittstelle zur Verfügung, welche das Erfassen von freien Texten ermöglicht. Das Textfeld kann als einzeiliges Eingabefeld oder als mehrzeiliges Feld definiert werden.


Im *Kombinationsfeld* (ComboBox) kann aus einer vorgegeben Menge von Werten ein einzelner Wert ausgewählt werden. Es kann definiert werden, ob neben den vorgegebenen Werten auch ein zusätzlicher Wert manuell eingetragen werden kann. Nach erfolgter Auswahl ist nur der gewählte Wert am Bildschirm sichtbar.


Im *Listenfeld* (ListBox) kann aus einer vorgegeben Menge von Werten ein oder mehrere Einträge ausgewählt werden. Es kann definiert werden, ob nur ein Wert oder mehrere ausgewählt werden können. Nach der erfolgten Auswahl bleibt die Liste weiterhin im Dialogfeld sichtbar.


Das *Kontrollkästchen* (CheckBox) dient zum Aktivieren bzw. Deaktivieren einer einzelnen Option. Es kann drei mögliche Zustände annehmen (aktiviert, deaktiviert, unbekannt).


Mit dem *Optionsfeld* (OptionButton) wird eine Auswahl aus einer Gruppe von Optionen zur Verfügung gestellt. Innerhalb der gleichen Gruppe kann nur ein Mitglied gleichzeitig aktiv sein.


 Das *Umschaltfeld* (ToggleButton) wird ähnlich wie in den Symbolleisten zur Darstellung der aktuellen Auswahl verwendet. Es wird meistens in kleinen Gruppen zusammen eingesetzt. Es kann jedoch auch als einzelne Schaltfläche eingesetzt werden. In den meisten Fällen wird zugunsten einer kleinen Grafik auf eine Beschriftung verzichtet.


 Der *Rahmen* (Frame) dient zum Zusammenfassen einer Gruppe von Steuerelementen, die thematisch zusammengehören. Er dient in erster Linie der optischen Unterstützung des Anwenders beim Bearbeiten des Dialogfelds. Dieses Steuerelement ist ebenfalls ein Container-Objekt und kann wiederum weitere Steuerelemente aufnehmen.


 Der *Befehlsschaltfläche* (CommandButton) wird eine Aktion hinterlegt. Durch das Betätigen derselben löst der Anwender die betreffende Aktion aus. Je nach Verwendung wird beim Betätigen der Schaltfläche das zugehörige Dialogfeld gleichzeitig ausgeblendet.

 Das *Register* (TabStrip) ist ein Container-Objekt. Es dient zum Darstellen gleicher Steuerelemente mit unterschiedlichem Inhalt. Der Inhalt der betreffenden Steuerelemente muss beim Wechseln der einzelnen Registerkarten aktualisiert werden.

 Die *Multiseiten* (MultiPage) werden eingesetzt, wenn viele Steuerelemente auf einem Dialogfeld vorhanden sind. Die einzelnen Seiten entsprechen jeweils einer Kategorie und beherbergen die entsprechenden Steuerelemente. Bei jeder einzelnen Seite des Steuerelements handelt es sich um ein Container-Objekt.

 Die *Bildlaufleiste* (ScrollBar) wird für die Wahl eines Wertes aus einer Menge von Werten verwendet. Mit dem Schieberegler kann der ungefähre Wert bestimmt werden. Der genaue Wert lässt sich mittels Einzelschritt (Pfeiltasten) festlegen.

 Das *Drehfeld* (SpinButton) wird normalerweise in Kombination mit einem zweiten Steuerelement eingesetzt. Es wird benötigt, um einen Wert aus einer bestimmten Menge auszuwählen. Die Synchronisation mit dem zugehörigen Steuerelement muss programmiert werden.

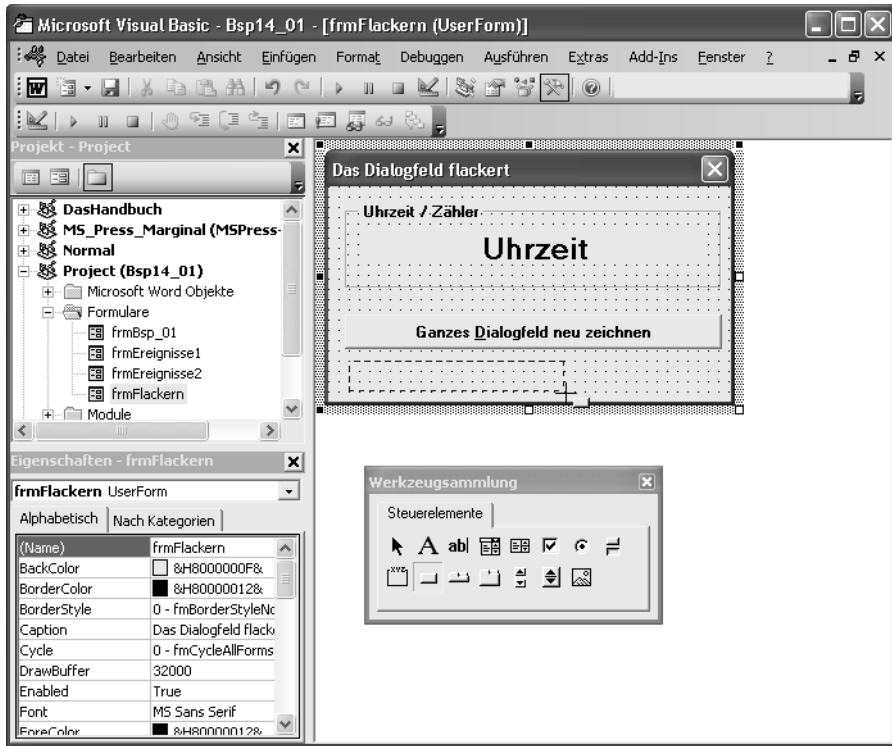
 Die *Anzeige* (Image) dient zum Darstellen eines Bildes bzw. einer Grafik auf dem Dialogfeld. Stimmt das Größenverhältnis der Grafik nicht mit jenem des Steuerelements überein, kann die Grafik in der Originalgröße, im gleichen Seitenverhältnis oder auch gestaucht dargestellt werden.

### Steuerelement einfügen

Um ein Steuerelement auf dem UserForm-Objekt zu platzieren, muss dieses (wie in Abbildung 14.7 dargestellt) am Bildschirm aktiv sein. Das gewünschte Element wird in der Werkzeugsammlung ausgewählt und in einem zweiten Schritte auf dem UserForm-Objekt platziert. Während das Element auf dem Dialogfeld platziert wird, ändert der Mauszeiger seine Form. Das Fadenkreuz dient zum genauen Positionieren des neuen Elements. Die Werkzeugsammlung kann innerhalb des Visual Basic-Editors eingeblendet werden, indem der Menübefehl *Ansicht/Werkzeugsammlung* aufgerufen wird.

Nachdem das Steuerelement eingefügt wurde, können die entsprechend zugehörigen Eigenschaften im Eigenschaftenfenster festgelegt werden. Das Eigenschaftenfenster lässt sich über das Menü *Ansicht* aktivieren.

Abbildg. 14.7 Einfügen eines *CommandButton*-Steuerelements auf dem *UserForm*-Objekt



Grundsätzlich kann ein UserForm-Objekt (oder einzelne Steuerelemente) während der Laufzeit des Makros erzeugt werden. Wie dies funktioniert, ist in Kapitel 19 detailliert beschrieben.

## Zusätzliche Steuerelemente einbinden

Neben den bereits vorgestellten Steuerelementen aus der »Microsoft Forms 2.0 Object Library« können weitere Steuerelemente aus anderen Programmbibliotheken dem UserForm-Objekt hinzugefügt werden.

Das eigentliche Hinzufügen von zusätzlichen Steuerelementen wurde bereits in Kapitel 8 besprochen.

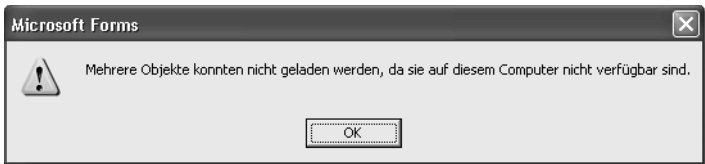
### HINWEIS

Werden zusätzliche Steuerelemente auf einem UserForm-Objekt eingebunden, muss sichergestellt werden, dass die verwendeten Programmbibliotheken zusammen mit dem Projekt zur Verfügung gestellt werden.

Die entsprechenden Dateien müssen auf der Arbeitsstation ordnungsgemäß installiert und registriert werden. Ansonsten lassen sich die Makros innerhalb des Projekts nicht nutzen. Eine entsprechende Fehlermeldung, wie in Abbildung 14.8 dargestellt, weist auf dieses Problem hin.



**Abbildg. 14.8** Im Projekt wurden zusätzliche Steuerelemente eingebunden, die auf der aktuellen Arbeitsstation nicht zur Verfügung stehen.



**WICHTIG**

Werden zusätzliche Steuerelemente eingebunden und die entsprechenden Programmbibliotheken zusammen mit dem Projekt ausgeliefert, müssen die zugehörigen Lizenzbestimmungen und Urheberrechte berücksichtigt werden.

# Steuerelemente und ihre Besonderheiten

Das Verhalten von einzelnen Steuerelementen aus der »Microsoft Forms 2.0 Object Library« unterscheidet sich teilweise vom bekannten Verhalten aus anderen Entwicklungsumgebungen. Aus diesem Grunde werden eine paar dieser Besonderheiten hier erläutert.

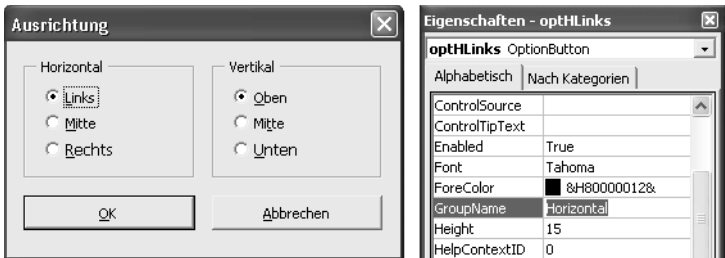
## Option (OptionButton)

Auf einem UserForm-Objekt können mehrere *OptionButtons* eingefügt werden. Zusammengehörende Optionen werden, wie in Abbildung 14.9 ersichtlich, oftmals in einen Rahmen gesetzt, damit dieser Umstand für den Anwender sofort ersichtlich ist.

Group-  
Name

Für das Programm hingegen muss während der Entwicklung des benutzerdefinierten Dialogfeldes festgelegt werden, welche Optionen zusammen in eine Gruppe gehören. Um eine Reihe von Optionen zu einer Gruppe zusammenzufassen, muss deren *GroupName*-Eigenschaft mit einer gemeinsamen und eindeutigen Bezeichnung belegt werden.

**Abbildg. 14.9** Zusammengehörende Optionen werden in Rahmen zusammen gefasst.



Die Auswertung der aktivierten Optionen gestaltet sich schon schwieriger. Die *OptionButton*-Steuerelemente können nicht in eine gemeinsame Auflistung aufgenommen werden. Diese bedeutet, dass sie über keinen gemeinsamen Index verfügen und somit auch keine Eigenschaft zur Verfügung steht, welche die gewählte Option als Resultat beinhaltet.

Aus diesem Grunde muss, wie in Listing 14.7 dargestellt, die Auswertung der gewählten Option einzeln erfolgen und kann beispielsweise mit einer *Select Case*-Anweisung erfolgen.

**Listing 14.7** Auswerten der gewählten Ausrichtung in den beiden *OptionButton*-Gruppen

```

Sub Demo_OptionButton()
    Dim strHorizontal As String
    Dim strVertikal As String

    Dim frm As frmAusrichtung

    'UserForm anzeigen
    Set frm = New frmAusrichtung
    With frm
        .Show

        If bButton Then
            'Horizontale Ausrichtung auswerten
            Select Case True
                Case .optHLinks.Value
                    strHorizontal = "Links"
                Case .optHMitte.Value
                    strHorizontal = "Mitte"
                Case .optHRechts.Value
                    strHorizontal = "Rechts"
            End Select

            'Vertikale Ausrichtung auswerten
            Select Case True
                Case .optVOben.Value
                    strVertikal = "Oben"
                Case .optVMitte.Value
                    strVertikal = "Mitte"
                Case .optVUnten.Value
                    strVertikal = "Unten"
            End Select

            MsgBox "Gewählte Position:" & vbCrLf &
                "Horizontal" & vbCrLf & strHorizontal & vbCrLf & _
                "Vertikal" & vbCrLf & strVertikal, _
                vbInformation & vbOKOnly
        End If

    End With
    Set frm = Nothing
End Sub

```

## Listenfeld (ListBox bzw. ComboBox)

Im Listenfeld ist es nicht ganz einfach, dieses mit Werten zu befüllen bzw. den gewählten Wert zu ermitteln. Stehen zudem mehrere Spalten zur Verfügung, wird die ganze Angelegenheit noch ein bisschen komplexer.

AddItem

Verfügt das Listenfeld nur über eine einzelne Spalte, kann der zugehörige Wert als Argument der AddItem-Methode übergeben werden:

```
With ListBox1
    .AddItem "1. Eintrag"
    .AddItem "2. Eintrag"
End With
```

**AddItem** Verfügt das Listenfeld jedoch über mehrere Spalten, kann zwar der AddItem-Methode weiterhin ein Argument übergeben werden. Die restlichen Spalten müssen jedoch nachträglich manuell in der List-Eigenschaft gesetzt werden:

```
With ListBox1
    .ColumnCount = 2
    .AddItem "1. Eintrag, 1. Spalte"
    .List(ListBox1.ListCount - 1, 1) = "1. Eintrag, 2. Spalte"
    .AddItem
    .List(ListBox1.ListCount - 1, 0) = "2. Eintrag, 1. Spalte"
    .List(ListBox1.ListCount - 1, 1) = "2. Eintrag, 2. Spalte"
End With
```

**List** Eine dritte Möglichkeit bietet die direkte Zuweisung eines Datenfelds (Array) an die List-Eigenschaft des Listenfelds. Dies ist sicher der schnellste und einfachste Weg, wenn die entsprechenden Daten bereits vorhanden sind:

```
Dim strListenWerte(0 To 1, 0 To 1) As String
strListenWerte(0, 0) = "1. Eintrag, 1. Spalte"
strListenWerte(0, 1) = "1. Eintrag, 2. Spalte"
strListenWerte(1, 0) = "2. Eintrag, 1. Spalte"
strListenWerte(1, 1) = "2. Eintrag, 2. Spalte"
With ListBox1
    .ColumnCount = 2
    .List = strListenWerte
End With
```

Zum Auslesen des gewählten Werts sind in Abbildung 14.10 die verschiedenen Eigenschaften und deren Abhängigkeiten dargestellt. Grundsätzlich stehen drei Eigenschaften zur Verfügung um den gewählten Eintrag im Listenfeld zu ermitteln.

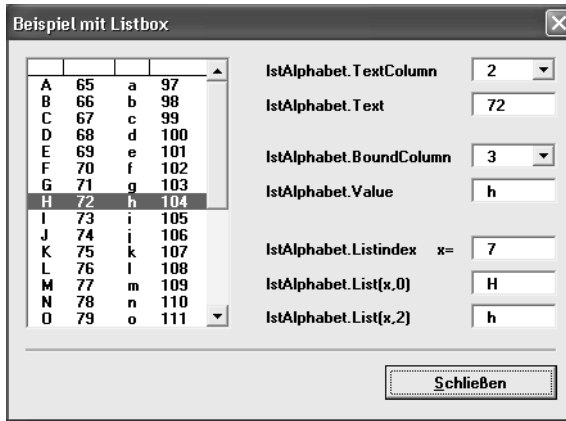
**Text** In der Text-Eigenschaft wird der Wert aus der in der TextColumn-Eigenschaft eingetragenen Spalte ausgegeben. Die Nummerierung der Spalten beginnt bei 1. Wird in der TextColumn-Eigenschaft der Wert 0 eingetragen, werden keine Werte ausgegeben, sondern die Position innerhalb der Liste (ListIndex). Wird ein Wert von -1 eingetragen, so wird der Wert aus der ersten sichtbaren Spalte (ColumnWidth größer 0) ausgegeben.

**Value** In der Value-Eigenschaft wird der Wert aus der in der BoundColumn-Eigenschaft eingetragenen Spalte ausgegeben. Die Nummerierung der Spalten beginnt bei 1. Wird in der BoundColumn-Eigenschaft der Wert 0 eingetragen, werden keine Werte ausgegeben, sondern die Position innerhalb der Liste (ListIndex).

**List** Als dritte Möglichkeit steht die List-Eigenschaft zur Verfügung. Die Abfrage des gewählten Eintrags erfolgt stets zusammen mit der ListIndex-Eigenschaft:

```
txtListboxList0.Text = lstAlphabet.List(lstAlphabet.ListIndex, eBuchstabeGrossZeichen)
```

**Abbildg. 14.10** Die verschiedenen Eigenschaften, um den aktuellen Wert des Listenfelds zu ermitteln



Column-  
Heads

In einem Listenfeld besteht die Möglichkeit, eine Zeile für Spaltenüberschriften zu aktivieren. Leider ist es nicht möglich, den Spaltenüberschriften auch entsprechende Werte zu hinterlegen. Diese zusätzliche Eigenschaft hatten die Entwickler des Steuerelements zu implementieren vergessen (vgl. Abbildung 14.10).



Die dargestellten Beispiele finden Sie in der Beispieldatei *Bsp14\_02.doc* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap14*.




## Anforderungen an ein Dialogfeld

An ein benutzerdefiniertes Dialogfeld stellt der Anwender grundsätzlich die gleichen Anforderungen wie an die internen Dialogfelder von Word oder die ganze grafische Benutzerschnittstelle von Windows.

Neben den Programmsequenzen, die die eigentliche Logik innerhalb der verschiedenen Steuerelemente steuern, sollten die allgemeinen Spielregeln zur Gestaltung von Benutzerschnittstellen eingehalten werden.

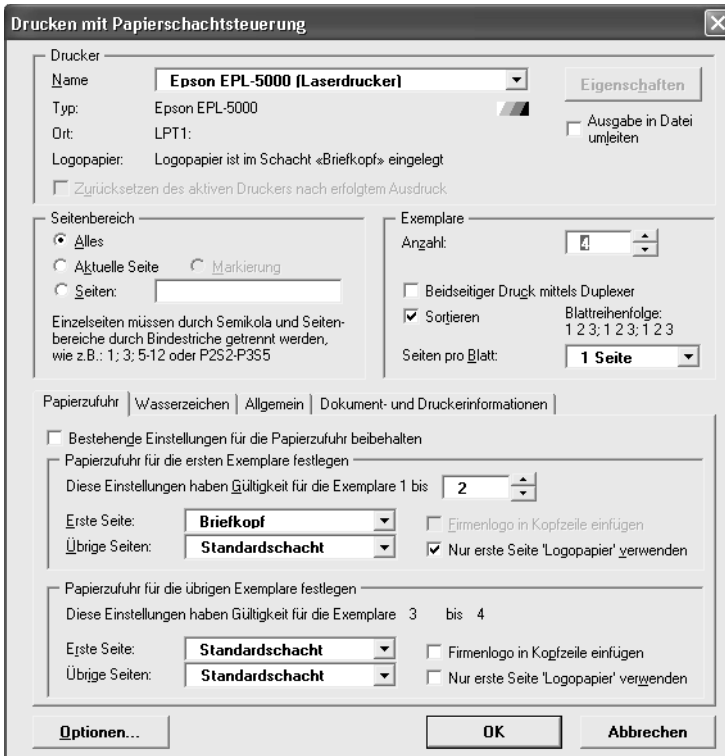
Anhand der nachstehenden Checkliste kann jedes neu entwickelte Dialogfeld geprüft werden:

- Aussagekräftige Bezeichnung in der Titelleiste des UserForm-Objekts (Caption-Eigenschaft).
- Einheitliche Schriftart und -größe bei allen verwendeten Steuerelementen (Font-Eigenschaft).
- Gleiche Maße für zusammengehörende Steuerelemente (Height- und Width-Eigenschaft). Gleichmäßige Ausrichtung der Steuerelemente (Left- und Top-Eigenschaft).
- Schaltflächen werden normalerweise auf der rechten Seite des Dialogfelds untereinander angeordnet. Eine zweite Möglichkeit bietet das Positionieren der Schaltflächen nebeneinander am unteren Rand des Dialogfeldes.
- Thematisch zusammengehörende Steuerelemente werden innerhalb eines Rahmens gruppiert. Damit das Dialogfeld übersichtlich bleibt, kann dieses anhand eines Multiseiten-Steuerelements aufgeteilt werden.

- Die Bedienung der einzelnen Steuerelemente sollte ohne die Verwendung der Maus nur über die Tastatur erfolgen können:
- Alle Steuerelemente sollten mittels der -Taste erreicht werden können (TabStop-Eigenschaft). Die Reihenfolge der »angesprungenen« Steuerelemente sollte mit der dargestellten Reihenfolge übereinstimmen (TabIndex-Eigenschaft oder über den Menübefehl *Ansicht/ Aktivierreihenfolge* festlegen).
- Jedes Steuerelement sollte mit einer eindeutigen Tastaturkombination ()+Buchstabe) direkt erreicht werden können (Accelerator-Eigenschaft).
- Die Schaltfläche OK kann durch Betätigen der -Taste ausgelöst werden (Default-Eigenschaft). Die Schaltfläche *Abbrechen* oder *Schließen* kann durch Betätigen der Esc-Taste ausgelöst werden (Cancel-Eigenschaft).
- Die verwendete Terminologie stimmt mit jener des entsprechenden Basisprogramms überein.
- Wenn immer möglich, werden für die gleichen Bezeichnungen in verschiedenen Dialogfeldern die gleichen Tastaturkombination zugewiesen. Diese sollten möglichst mit jenen aus den internen Dialogfeldern korrespondieren.

Abbildg. 14.11

Komplexes benutzerdefiniertes Dialogfeld, das die Anforderungen an ein Dialogfeld abdeckt



**Drucken mit Papierschatzsteuerung**

**Drucker**

Name: **Epson EPL-5000 (Laserdrucker)** Eigenschaften

Typ: Epson EPL-5000

Ort: LPT1: ☐ Ausgabe in Datei umleiten

Logopapier: Logopapier ist im Schacht »Briefkopf« eingelegt

☐ Zurücksetzen des aktiven Druckers nach erfolgreichem Ausdruck

**Seitenbereich**

☒ Alles

☐ Aktuelle Seite ☐ Markierung

☐ Seiten:

Einzelseiten müssen durch Semikola und Seitenbereiche durch Bindestriche getrennt werden, wie z.B.: 1; 3; 5-12 oder P252-P355

**Exemplare**

Anzahl:

☐ Beidseitiger Druck mittels Duplexer

☒ Sortieren Blattreihenfolge: 1 2 3; 1 2 3; 1 2 3

Seiten pro Blatt: **1 Seite**

Papierzufuhr | Wasserzeichen | Allgemein | Dokument- und Druckerinformationen

☐ Bestehende Einstellungen für die Papierzufuhr beibehalten

Papierzufuhr für die ersten Exemplare festlegen

Diese Einstellungen haben Gültigkeit für die Exemplare 1 bis

Erste Seite: **Briefkopf** ☐ Firmenlogo in Kopfzeile einfügen

Übrige Seiten: **Standardschacht** ☒ Nur erste Seite »Logopapier« verwenden

Papierzufuhr für die übrigen Exemplare festlegen

Diese Einstellungen haben Gültigkeit für die Exemplare 3 bis 4

Erste Seite: **Standardschacht** ☐ Firmenlogo in Kopfzeile einfügen

Übrige Seiten: **Standardschacht** ☐ Nur erste Seite »Logopapier« verwenden

**Optionen...** **OK** **Abbrechen**

### **Benutzerdefiniertes Dialogfeld effizient erstellen**

Um ein benutzerdefiniertes Dialogfeld effizient zu erstellen, sollte stets mit dem gleichen Konzept gearbeitet werden. Wir Autoren haben Ihnen ein mögliches Szenario zusammengestellt, das Sie bei der Entwicklung eines UserForm-Objekts unterstützen soll.

Wir empfehlen Ihnen ein schrittweises Vorgehen, damit die Übersicht gewährleistet bleibt. Dies gilt in erster Linie für die gegenseitigen Abhängigkeiten zwischen den einzelnen Steuerelementen:

- Erstellen Sie eine kleine Skizze, um festzuhalten, wie das neue Dialogfeld ungefähr aussehen wird. Tragen Sie die Abhängigkeiten zwischen den einzelnen Steuerelementen ein.
- Fügen Sie ein neues UserForm-Objekt in das Projekt ein. Weisen Sie dem Objekt im Eigenschaftsfenster den einzelnen Eigenschaften alle benötigten Standardwerte zu.
- Fügen Sie ein erstes Steuerelement dem Dialogfeld hinzu. Weisen Sie dem Steuerelement im entsprechenden Eigenschaftsfenster den einzelnen Eigenschaften alle benötigten Standardwerte zu. Schreiben Sie die zugehörigen Prozeduren für die Bildschirmlogik und Abhängigkeiten des Steuerelements. Testen Sie die Funktionsweise der erzeugten Programmsequenzen.
- Fügen Sie das nächste Steuerelement dem Dialogfeld hinzu. Weisen Sie wiederum den einzelnen Eigenschaften alle benötigten Standardwerte zu. Schreiben bzw. erweitern Sie die zugehörigen Prozeduren für die Bildschirmlogik und Abhängigkeiten der Steuerelemente und testen Sie deren Funktionsweise. Wiederholen Sie diesen Schritt, bis alle benötigten Steuerelemente hingefügt wurden.
- Erstellen Sie die benötigten Prozeduren für die Initialisierung (UserForm\_Initialize) und Terminierung (UserForm\_Terminate) der Instanz des UserForm-Objekts.
- Binden Sie den Aufruf des Dialogfelds in das Makro ein. Erstellen Sie die benötigten Prozeduren, die nach dem Schließen des Dialogfelds abgearbeitet werden müssen.
- Testen Sie das Makro und das Dialogfeld ausgiebig und berücksichtigen Sie dabei auch alle bekannten Ausnahmen und Sonderfälle.

Weitere interessante Hinweise sind in der Online-Hilfe zu VBA unter dem Thema »Erstellen eines benutzerdefinierten Dialogfeldes« zusammengefasst.

## **Dialogfelder zur Laufzeit beeinflussen**

Der Aufbau eines benutzerdefinierten Dialogfelds wird zur Entwicklungszeit erstellt. Den Steuerelementen werden dabei die Standardeigenschaften zugewiesen. Bis zu diesem Zeitpunkt ist das ganze UserForm-Objekt statisch aufgebaut. Sollen einzelne Bereiche dynamisch beeinflusst werden, muss dies zur Laufzeit des Programms erfolgen.

### **Eigenschaften der Dialogfelder zwischenspeichern**

Sollen die gewählten Einstellungen eines benutzerdefinierten Dialogfelds beim erneuten Aufruf des zugehörigen UserForm-Objekts dem Anwender wieder zur Verfügung stehen, müssen diese Werte an einer zentralen Stelle zwischengespeichert werden.

Als Speicherort für diese Einstellung kann die Windows-Registrierung, eine Konfigurationsdatei oder eine Datenbank verwendet werden. In den nachstehenden Beispielen werden die Daten jeweils in der Windows-Registrierung abgespeichert. In Kapitel 12 wurde aufgezeigt, wie einzelne Werte in die Windows-Registrierung abgespeichert und von dort wieder ausgelesen werden können.

### Abspeichern von Eigenschaften

Wird die Instanz eines UserForm-Objekts zerstört, wird automatisch das Terminate-Ereignis ausgelöst. Spätestens zu diesem Zeitpunkt müssen die gewünschten Werte abgespeichert werden, ansonsten gehen sie verloren.

In Listing 14.8 wird gezeigt, wie alle Optionen und die Position des Dialogfelds in der Windows-Registrierung abgespeichert werden. Die beiden Werte zur Position des UserForm-Objekts werden in einem eigenen Schlüssel abgespeichert (vgl. Abbildung 14.12).

**Listing 14.8** Abspeichern der Dialogfeld-Eigenschaften während der Terminierung des *UserForm*-Objekts

```
Const REG_APP As String = "Word-Programmierung - Das Handbuch"
Const REG_SEC As String = "Bsp14_03"

Private Sub UserForm_Terminate()
'Werte in Windows-Registrierung abspeichern
SaveSetting REG_APP, REG_SEC & "\" & Me.Name, "Top", Me.Top
SaveSetting REG_APP, REG_SEC & "\" & Me.Name, "Links", Me.Left

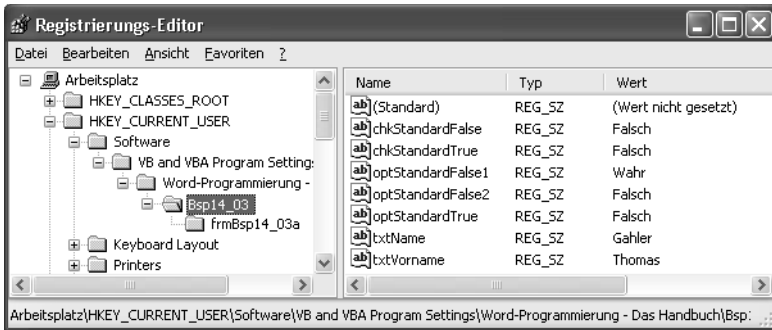
SaveSetting REG_APP, REG_SEC, "chkStandardTrue", chkStandardTrue.Value
SaveSetting REG_APP, REG_SEC, "chkStandardFalse", chkStandardFalse.Value

SaveSetting REG_APP, REG_SEC, "optStandardTrue", optStandardTrue.Value
SaveSetting REG_APP, REG_SEC, "optStandardFalse1", optStandardFalse1.Value
SaveSetting REG_APP, REG_SEC, "optStandardFalse2", optStandardFalse2.Value

SaveSetting REG_APP, REG_SEC, "txtName", txtName.Text
SaveSetting REG_APP, REG_SEC, "txtVorname", txtVorname.Text
End Sub
```

#### HINWEIS

Um den Status von Optionen (OptionButton) abzuspeichern, müssen die Werte aller zur Gruppe gehörenden Steuerelemente abgespeichert werden. Dies ist deshalb notwendig, weil innerhalb von VBA (im Gegensatz zu Visual Basic) das aktive Steuerelement nicht über einen Index ermittelt werden kann.

**Abbildg. 14.12** Abgespeicherte Werte und Eigenschaften in der Windows-Registrierung


### Setzen von Eigenschaften

Wurden die Eigenschaften und Werte aus dem UserForm-Objekt erst einmal an einem zentralen Ort abgespeichert, können diese bei der nächsten Verwendung des Dialogfelds wieder eingetragen werden.

Wird eine neue Instanz des UserForm-Objekts angelegt, wird automatisch das Initialize-Ereignis ausgelöst. Frühestes zu diesem Zeitpunkt können die gewünschten Werte eingetragen werden.

In Listing 14.9 wird gezeigt, wie alle Optionen und die Position des Dialogfelds aus der Windows-Registrierung ausgelesen und den einzelnen Steuerelementen wieder zugewiesen werden. Dies hat zur Folge, dass das Dialogfeld wieder genauso wie beim letzten Aufruf dargestellt wird. Die während der Entwurfszeit eingetragenen Standardwerte werden übersteuert.

**Listing 14.9** Einlesen der Dialogfeld-Eigenschaften während der Initialisierung des *UserForm*-Objekts

```
Const REG_APP As String = "Word-Programmierung - Das Handbuch"
Const REG_SEC As String = "Bsp14_03"

Private Sub UserForm_Initialize()
    'Werte aus Windows-Registrierung auslesen
    Me.Top = CInt(GetSetting(REG_APP, REG_SEC & "\" & Me.Name, "Top", 100))
    Me.Left = CInt(GetSetting(REG_APP, REG_SEC & "\" & Me.Name, "Links", 100))

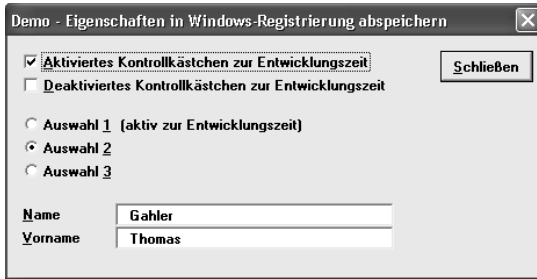
    chkStandardTrue.Value = CBool(GetSetting(REG_APP, REG_SEC, "chkStandardTrue", True))
    chkStandardFalse.Value = CBool(GetSetting(REG_APP, REG_SEC, "chkStandardFalse", False))

    optStandardTrue.Value = CBool(GetSetting(REG_APP, REG_SEC, "optStandardTrue", _
        True))
    optStandardFalse1.Value = CBool(GetSetting(REG_APP, REG_SEC, "optStandardFalse1", _
        False))
    optStandardFalse2.Value = CBool(GetSetting(REG_APP, REG_SEC, "optStandardFalse2", _
        False))

    txtName.Text = GetSetting(REG_APP, REG_SEC, "txtName", "")
    txtVorname.Text = GetSetting(REG_APP, REG_SEC, "txtVorname", "")
End Sub
```



Abbildg. 14.13 Die Werte des Dialogfeldes wurden aus der Windows-Registrierung übernommen.



## Eigenschaften der Dialogfelder in Dateien auslagern

Die meisten Eigenschaften eines benutzerdefinierten Dialogfeldes oder der zugehörigen Steuerelemente werden beim Erstellen desselben festgelegt und müssen zur Laufzeit des Programms auch nicht angepasst werden.

Dennoch besteht manchmal das Bedürfnis, einzelne Eigenschaften von Steuerfeldern dynamisch zu gestalten. Dabei kann es sich beispielsweise um eine sprachspezifische Bezeichnung der Steuerelemente oder um einen geänderten Inhalt eines Listenfelds oder ähnliches handeln.

In beiden Fällen müsste der Programmcode angepasst werden, um diese Änderungen vorzunehmen. Es besteht jedoch die Möglichkeit, diese Werte vom eigentlichen Programm zu trennen und in eine externe Datenquelle auszulagern.

Je nach Anforderung kann als Datenquelle eine Konfigurationsdatei oder eine Datenbank verwendet werden. In den nachstehenden Beispielen werden die Daten jeweils aus unterschiedlichen Konfigurationsdateien eingelesen. Dabei handelt es sich um so genannte *.ini*- oder *.txt*-Dateien. In Kapitel 12 wurde bereits erläutert, wie einzelne Werte in Konfigurationsdateien abgespeichert und von dort wieder eingelesen werden können.

### Eigenschaften in *.ini*-Dateien hinterlegen

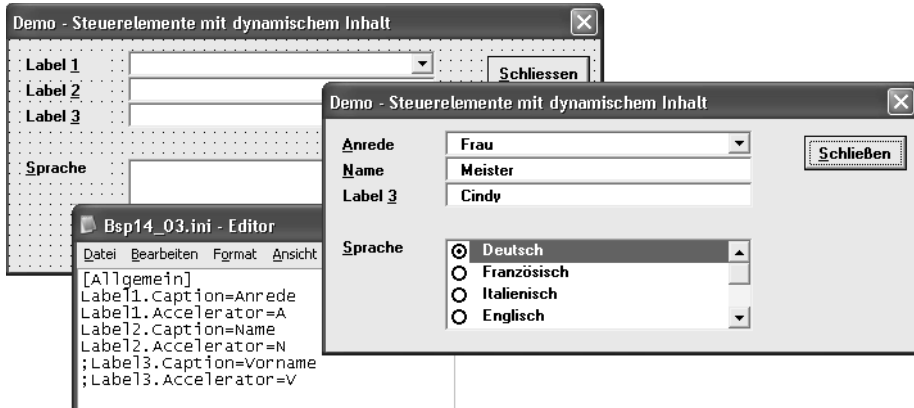
In Listing 14.10 ist eine Programmsequenz aus dem Initialize-Ereignis des UserForm-Objekts aufgeführt. Hier werden die Bezeichnungen der Steuerelemente aus einer *.ini*-Datei eingelesen und auf das Dialogfeld übertragen.

Listing 14.10 Einlesen der Steuerelement-Bezeichnungen aus der Konfigurationsdatei

```
'Werte aus .ini-Datei auslesen
strDatei = ThisDocument.Path & "\" & FILE INI
Label1.Caption = fktPrivateProfileString(strDatei, "Allgemein", _
    "Label1.Caption", Label1.Caption)
Label1.Accelerator = fktPrivateProfileString(strDatei, "Allgemein", _
    "Label1.Accelerator", Label1.Accelerator)
Label2.Caption = fktPrivateProfileString(strDatei, "Allgemein", _
    "Label2.Caption", Label2.Caption)
Label2.Accelerator = fktPrivateProfileString(strDatei, "Allgemein", _
    "Label2.Accelerator", Label2.Accelerator)
Label3.Caption = fktPrivateProfileString(strDatei, "Allgemein", _
    "Label3.Caption", Label3.Caption)
Label3.Accelerator = fktPrivateProfileString(strDatei, "Allgemein", _
    "Label3.Accelerator", Label3.Accelerator)
```

**HINWEIS** Anstelle der `PrivateProfileString`-Methode des System-Objekts wird die benutzerdefinierte Funktion `fktPrivateProfileString` verwendet. Diese Funktion wurde bereits in Kapitel 12 vorgestellt.

Abbildg. 14.14 Benutzerdefiniertes Dialogfeld zur Entwicklungszeit und zur Laufzeit mit zugehöriger `.ini`-Datei



**HINWEIS** In Abbildung 14.14 ist ersichtlich, wie in der `Ini`-Datei `Bsp14_03.ini` die beiden Einträge `Label3.Caption` und `Label3.Accelerator` auskommentiert wurden. Dies hat zur Folge, dass im Dialogfeld die Standardwerte, die während der Entwicklungszeit gesetzt wurden, dargestellt werden.

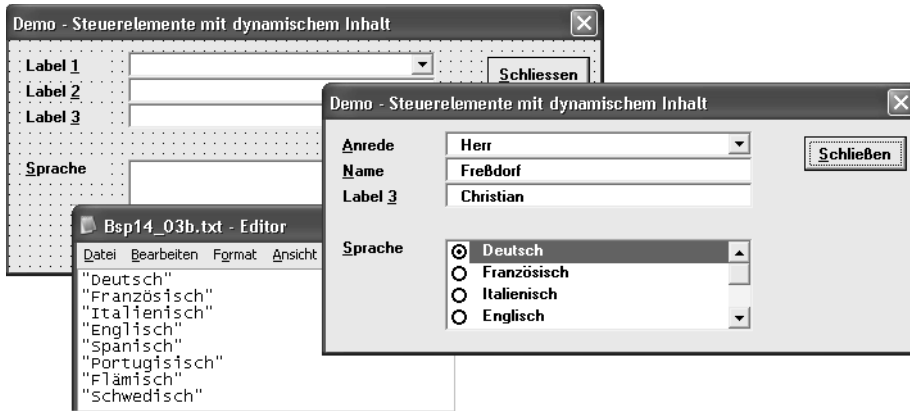
### Eigenschaften in `.txt`-Dateien auslagern

In Listing 14.11 ist eine weitere Programmsequenz aus dem gleichen `Initialize`-Ereignis des `UserForm`-Objekts aufgeführt. In diesem Fall wird der Inhalt eines Listenfelds aus einer `.txt`-Datei eingelesen und in das Listenfeld übertragen.

Listing 14.11 Einlesen des Inhalts des Listenfeldes aus der Textdatei

```
'Werte aus .txt-Datei auslesen
strDatei = ThisDocument.Path & "\" & FILE_TXT_B
If fktExistiertDatei(strDatei) Then
    Open strDatei For Input As #1
    Do While Not EOF(1)
        Input #1, strEintrag
        lstSprache.AddItem strEintrag
    Loop
    Close #1
Else
    lstSprache.AddItem "Deutsch"
End If
```

Abbildg. 14.15 Benutzerdefiniertes Dialogfeld zur Entwicklungszeit und zur Laufzeit mit zugehöriger .txt-Datei



Das dargestellte Beispiel finden Sie in der Beispieldatei *Bsp14\_03.doc* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap14*.

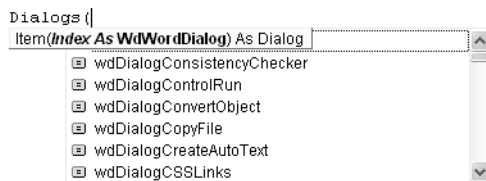
## Interne Dialogfelder

Bei der täglichen Arbeit mit Word begegnen Ihnen an verschiedenen Stellen Dialogfelder, in denen Sie Einstellungen oder Änderungen vornehmen oder Informationen auslesen können. Bei diesen Dialogfeldern handelt es sich um Word-interne (integrierte) Dialogfelder, die sich meist hinter Menübefehlen verbergen und die Interaktion mit dem Anwender übernehmen.

Dialogs-  
Auflis-  
tung

In der VBA-Umgebung haben Sie Zugriff auf diese Dialogfelder über das jeweilige Dialog-Objekt der Dialogs-Auflistung, da jedes Dialog-Objekt ein eigenes Dialogfeld in Word darstellt. Zur Festlegung des gewünschten Dialog-Objektes müssen Sie entweder den Index oder den Namen der dem Dialogfeld zugeordneten `wdWordDialog`-Konstante angeben. Laut Count-Eigenschaft der Auflistung stehen in Word 2003 insgesamt 230 Dialogfelder zur Verfügung.

Alle benannten Konstanten werden Ihnen im Visual Basic-Editor nach Eingabe der ersten Klammer »(« in einer Auswahlliste angezeigt.

Abbildg. 14.16 Übersicht über die benannten *WdWordDialog*-Konstanten

## Dialogfelder »anzeigen«, »anzeigen und ausführen« oder »ausführen«

Nach Auswahl eines Dialogfeldes stehen Ihnen nur wenige allgemeine Methoden des Dialogs-Objektes direkt zur Verfügung bzw. werden Ihnen angeboten.

Dazu gehören die Methoden `Show` und `Display`, mit denen Sie das Dialogfeld anzeigen können. Diese beiden Methoden unterscheiden sich durch die Behandlung der mit dem jeweiligen Dialogfeld verknüpften Aktionen bzw. durch die Übernahme eventueller Einstellungen.

`Show`

Die `Show`-Methode versucht die Einstellungen zu übernehmen bzw. eine mit dem Dialogfeld verknüpfte Aktion auszuführen. Gelingt dies nicht, wird eine Fehlermeldung ausgegeben.

`Display`

Die `Display`-Methode hingegen zeigt ein Dialogfeld nur an, ohne eine evtl. verknüpfte Aktion auszuführen oder Einstellungen zu übernehmen. In diesem Fall müssen Sie selbst ggf. für die Ausführung der Aktion oder die Übernahme sorgen.

`TimeOut`

Zur Verdeutlichung lassen Sie das Dialogfeld zum Speichern eines Dokuments unter einem anderen Namen einmal mit der `Show`-Methode anzeigen:

```
Dialogs(wdDialogFileSaveAs).Show
```

und einmal mit der `Display`-Methode:

```
Dialogs(wdDialogFileSaveAs).Display
```

Wenn Sie einen Dateinamen angeben und dies durch Anklicken der »Speichern«-Schaltfläche bestätigen, wird nur bei Verwendung der `Show`-Methode die Aktion des Speicherns ausgeführt. Bei der `Display`-Methode hingegen wird das Dialogfeld ohne Aktion wieder geschlossen.

Über die Rückgabewerte beider Methoden können Sie gezielt auf die Anwenderaktion reagieren. Dies ist dann wichtig, wenn Sie die Dialogfelder nur anzeigen lassen und anschließend selbst eine Aktion ausführen möchten.

Die Dialogfelder liefern die in Tabelle 14.3 aufgeführten Rückgabewerte.

**Tabelle 14.3** Übersicht über die Rückgabewerte der Dialogfelder

Rückgabewert	Beschreibung
-2	Wenn Sie in einem Dialogfeld Änderungen vorgenommen und es anschließend über die Schaltfläche <i>Schließen</i> beendet haben (z.B. <code>wdDialogFilePrint</code> ).
-1	Das Dialogfeld wurde über die Schaltfläche <i>OK</i> bzw. die jeweilige Aktion des Dialogfeldes beendet (z.B. <code>wdDialogFilePrint</code> ). Bei Verwendung der <code>Show</code> -Methode wird die Aktion ausgeführt.
0	Das Dialogfeld wurde über die Schaltfläche <i>Abbrechen</i> , über das Schließen-Symbol in der Titelleiste oder über die Tastenkombination <code>[Alt] + [F4]</code> beendet.
> 0	Wenn Sie in einem Dialogfeld mit mehreren weiteren Schaltflächen das Dialogfeld über eine dieser Schaltflächen verlassen haben. Dies ist bei selbst erstellten Dialogfeldern der Fall.

Über den optionalen Parameter `TimeOut` können Sie die Zeitspanne festlegen, nach der das Dialogfeld automatisch geschlossen wird, wenn keine Eingabe oder Auswahl erfolgt, also bei Inaktivität:

```
Dialogs(wdDialogFilePrint).Display 5000 '~5 Sekunden
```

Ohne diesen Parameter muss das Dialogfeld manuell geschlossen werden.

Execute

Neben diesen beiden Möglichkeiten der Anzeige können Sie Dialogfelder auch direkt mit den Einstellungen ausführen, ohne dass eine Interaktion des Anwenders notwendig ist. Dazu steht Ihnen die `Execute`-Methode zur Verfügung:

```
Dialogs(wdDialogFileNew).Execute
```

Wenn Sie das Dialogfeld `wdDialogFileNew` über die `Execute`-Methode aufrufen, wird direkt ein neues Dokument erstellt, ohne dass das Dialogfeld angezeigt wird. Beim Aufruf über die Methoden `Display` oder `Show` wird hingegen das Auswahlfenster für neue Dokumente angezeigt.

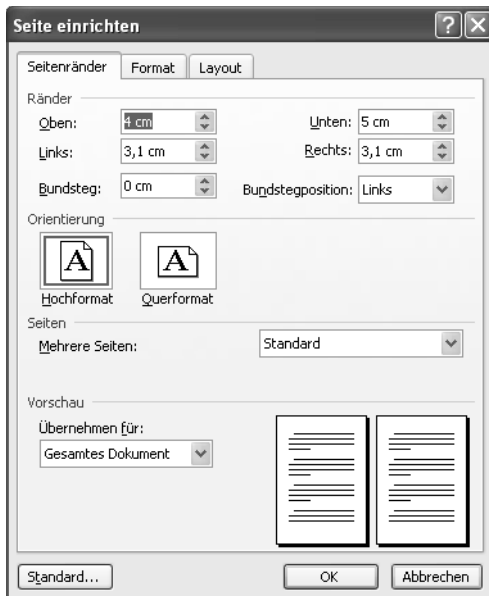
Update

Eine Besonderheit der Dialogfeldanzeige ist die `Update`-Methode im Zusammenhang mit Objektverweisen auf ein Dialogfeld. Mit dieser Methode können Sie nach dem Setzen eines Objektverweises auf ein Dialogfeld Änderungen, die sich auf das Dialogfeld selbst beziehen, in diesem Objektverweis aktualisieren, bevor Sie das Dialogfeld anzeigen.

Als Beispiel dient das Dialogfeld `wdDialogFileDocumentLayout` zur Seiteneinrichtung im Menü *Datei/Seite einrichten*, auf das Sie einen Objektverweis gesetzt haben:

```
Set dlg = Dialogs(wdDialogFileDocumentLayout)  
dlg.Display
```

Abbildg. 14.17 Anzeige des Dialogfeldes zur Seiteneinrichtung `wdDialogFileDocumentLayout`



Wenn die Dokumentseite wie in Abbildung 14.17 im Hochformat eingerichtet ist, und Sie anschließend die Seitenorientierung ändern, führt die erneute Anzeige dieses Dialogfeldes mittels `Display` über den Objektverweis zu einem Fehler: »Laufzeitfehler '4608': Wert nicht im Definitionsbereich.« bzw. »Laufzeitfehler '4605': Dieser Befehl ist nicht verfügbar.«

```
Dim dlg As Dialog
Set dlg = Dialogs(wdDialogFileDocumentLayout)
dlg.Display
ActiveDocument.PageSetup.Orientation = wdOrientLandscape
dlg.Display
```

Der Fehler tritt deshalb auf, weil im Objektverweis auch die Einstellung zur Seitenorientierung enthalten ist und die Änderung an der Seite bei der erneuten Anzeige nicht in den Objektverweis aufgenommen werden kann, bzw. nicht mehr mit der tatsächlichen Einstellung der Seite übereinstimmt.

Um alle Änderungen in dem Dialogfeld auch in die erneute Anzeige zu übernehmen, müssen Sie die `Update`-Methode anwenden.

**Listing 14.12** Aktualisieren eines per Objektverweis referenzierten Dialogfeldes

```
Sub subUpdateDialog()
    Dim dlg As Dialog
    Set dlg = Dialogs(wdDialogFileDocumentLayout)
    dlg.Display
    ActiveDocument.PageSetup.Orientation = wdOrientLandscape
    dlg.Update
    dlg.Display
End Sub
```

Anschließend wird die geänderte Orientierung berücksichtigt und in diesem Dialogfeld korrekt angezeigt.

## Dialogfelder konfigurieren, vorbelegen und auswerten

Jedes Dialogfeld besitzt seine eigenen kontextbezogenen Einstellmöglichkeiten. Über die Dialog-Objekte der `Dialogs`-Auflistung oder den Objektkatalog lassen sich diese Einstellmöglichkeiten (Argumente) jedoch nicht erkennen und in der Online-Hilfe findet sich über den Suchbegriff »Argumente für integrierte Dialoge« zwar eine Liste der Argumente, jedoch keine nähere Erklärung zu den aufgeführten Argumenten und ihre Anwendung.

Ärgerlich ist jedoch, dass auch in Word 2003 nicht alle Argumente unterstützt werden und diese nicht gekennzeichnet sind. So läuft das Ermitteln und Anwenden der Argumente mit seinen (benannten) Konstanten häufig auf ein Ausprobieren hinaus.

### HINWEIS

Glücklicherweise stimmen die meisten Argumente für die internen Dialogfelder mit den alten WordBasic-Anweisungen überein. Aus diesem Grunde ist die alte WordBasic-Hilfe eine hilfreiche Ressource, wenn mit der `Dialogs`-Auflistung gearbeitet wird. Die Datei *Wrdbasic.hlp* befindet sich auf der CD-ROM zum Buch im Ordner `\Beilagen\Word95 Wordbasic Hilfe`.

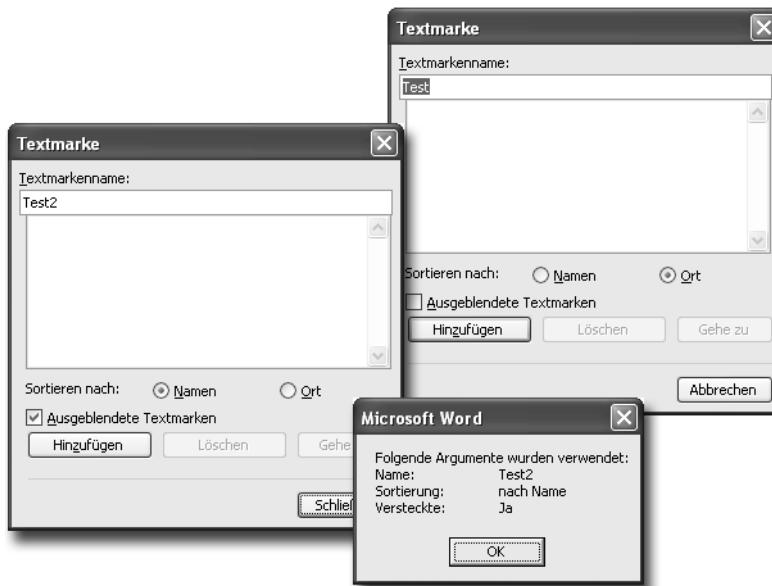
In Listing 14.13 wird das Dialogfeld zum Einfügen von Textmarken so eingestellt, dass ein Name für die Textmarke vorgegeben, die Anzeige versteckter Textmarken ausgeschaltet und die Sortierung nach Ort eingeschaltet wird. Da Sie diese Einstellungen aber im Dialogfeld ändern können, müssen Sie anschließend die Argumente wieder auswerten, um die verwendeten Einstellungen zu berücksichtigen.

Deshalb werden die letztendlich im Dialogfeld vorgenommenen Einstellungen in einer Übersicht noch einmal angezeigt.

**Listing 14.13** Konfigurieren des Dialogfeldes *wdDialogInsertBookmark* mittels Argumente und Anzeige der verwendeten Argumente

```
Sub subDialogArguments()
    Dim dlg As Dialog
    Dim strMSG As String
    Set dlg = Dialogs(wdDialogInsertBookmark)
    With dlg
        .Name = "Test"
        ' .SortBy = wdSortByName '0
        .SortBy = wdSortByLocation '1
        .Hidden = 0
        .Display
        strMSG = "Folgende Argumente wurden verwendet:" & vbCrLf
        strMSG = strMSG & "Name:" & vbTab & vbTab & .Name & vbCrLf
        strMSG = strMSG & "Sortierung:" & vbTab & IIf(.SortBy = wdSortByName, _
            "nach Name", "nach Ort") & vbCrLf
        strMSG = strMSG & "Versteckte:" & vbTab & IIf(.Hidden = 1, "Ja", "Nein") & vbCrLf
    End With
    MsgBox strMSGEnd Sub
```

**Abbildg. 14.18** Voreinstellen der Dialogfelder mittels Argumenten



Bei Dialogfeldern, die Einstellungen auf mehreren Registerkarten verteilt anbieten, können Sie mit dem Namen oder dem Index-Wert der Registerkarte diese aktivieren und somit in den Vordergrund bringen. Dazu steht Ihnen die `DefaultTab`-Eigenschaft zur Verfügung. Diese werden Ihnen im Visual Basic-Editor in einer Auswahlliste angeboten, wenn Sie diese Eigenschaft verwenden. Allerdings ist aus dieser Auswahlliste die Zuordnung dieser `WdWordDialogTab`-Konstanten zu den Dialogfelder nur über die Namensähnlichkeit und nicht kontextabhängig erkennbar.

## Übersicht über die in Word enthaltenen Dialogfelder

Die meisten Dialogfelder lassen sich über ihre benannten `WdWordDialog`-Konstanten (bzw. den Index) aufrufen und anzeigen. Allerdings ist diese Zuordnung zwischen Dialogfeld und Index in der Online-Hilfe nicht dokumentiert.

In Listing 14.14 werden alle Dialogfelder, die Sie über die Auswahlliste auswählen können, mit ihrem deutschen Namen und ihrem Index in einem Dokument ausgegeben.

**ACHTUNG** Beim Ausführen des Makros *subListDialogs* werden mitunter Fehlermeldungen oder Hinweismeldungen bzgl. Outlook-Zugriffe ausgegeben, die von den verwendeten Dialogfeld-Verweisen in der `For Each...Next`-Anweisung stammen und sich nicht vermeiden lassen.

**Listing 14.14** Ausgabe aller benannten Dialogfelder mit Index in ein neues Dokument

```
Sub subListDialogs()
    Dim doc As Document
    Dim dlg As Dialog
    Set doc = Documents.Add
    On Error Resume Next
    Application.DisplayAlerts = wdAlertsNone
    doc.Range.InsertAfter "Index" & vbTab & "Dialog-Name" & vbTab & "Register" & vbCrLf
    For Each dlg In Application.Dialogs
        doc.Range.InsertAfter dlg & vbTab & dlg.CommandName & vbTab & dlg.DefaultTab & vbCrLf
    Next dlg
    doc.Range.ConvertToTable vbTab, 4
    Application.DisplayAlerts = wdAlertsAll
End Sub
```

Allerdings gibt es neben diesen Dialogfeldern auch solche, die keine benannte `WdWordDialog`-Konstante besitzen und sich ausschließlich über ihren Dialogfeld-Index ansprechen lassen.

So lässt sich z.B. das Dialogfeld *Dateieigenschaft* im Menü *Datei/Eigenschaften* nur über den Index aufrufen:

```
Dialogs(750).Show
```

Die Ermittlung aller über den Index erreichbaren Dialogfelder kann nur anhand einer Schleife erfolgen, die alle möglichen Index-Werte ausprobiert und den Namen des Dialogfeldes zurückliefert.





In der Datei *Bsp14\_04.doc* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap14* finden Sie das Makro *subFindDialogs*, das Ihnen eine Liste aller Dialogfelder bis zum Index-Wert 5000 in einem neuen Dokument auflistet. Dabei werden mitunter Fehlermeldungen oder Hinweismeldungen bzgl. Outlook-Zugriffe ausgegeben, die von den verwendeten Dialogfeld-Verweisen stammen und sich nicht vermeiden lassen.

Sie finden eine vollständige Übersicht auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap14\wdDialogsÜbersicht\_Word2003.doc*. Eine Übersicht über die *wdWordDialog*-Konstanten finden Sie in der Datei *\Beispiele\Kap14\wdDialogsKonstanten-Word2003.doc*.

Die internen Dialogfelder von Word sind, neben vielen zusätzlichen Informationen, auch in der Datei *Interne WordBefehle.doc* aufgeführt. Diese Datei befindet sich im Ordner *\Beilagen\Interne Word-Befehle*.

Die Gesamtheit aller Dialogfelder (Dialog-Objekte) in Word lässt sich neben dieser Unterscheidung grob in folgende Bereiche unterscheiden:

- Dialog-Objekte, die sich anzeigen lassen
- Dialog-Objekte, die direkt ausgeführt werden und kein Dialogfeld anzeigen

Die Dialog-Objekte, die angezeigt werden können, lassen sich auch über die benannten *WdWordDialog*-Konstanten der *Dialogs*-Auflistung sowie über die Auswahlliste im Visual Basic-Editor auswählen und aufrufen.

Die Dialog-Objekte, die direkt ausgeführt werden, anstatt ein Dialogfeld anzuzeigen, besitzen keine *WdWordDialog*-Konstanten und können nur über ihren Index aufgerufen werden. Dazu gehört auch das Dialogfeld »Überschreiben« (Index 13).

Wenn Sie dieses Dialogfeld anzeigen lassen wollen, bewirkt der Aufruf direkt ein Umschalten des Überschreibmodus:

```
Dialogs(13).Show
```

Ein weiteres Dialogfeld ohne Anzeige ist das Dialogfeld »FelderAktualisieren« (Index 32), mit dem alle Felder im Haupttext eines Dokumentes aktualisiert werden, was der Taste F9 entspricht.



Die dargestellten Beispiele finden Sie in der Beispieldatei *Bsp14\_04.doc* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap14*.

Im gleichen Ordner befindet sich auch die Datei *wdDialogsÜbersicht\_Word2003.doc*. In dieser Datei sind alle *WdWordDialog*-Konstanten mit deutscher und englischen Bezeichnung, deren Index und die zugehörigen Argumente aufgelistet. Im Weiteren sind jene Dialogfelder aufgelistet, die keine benannten *WdWordDialog*-Konstanten besitzen.

# FileDialog-Objekt

Neben den in obigem Abschnitt »Interne Dialogfelder« aufgeführten Dialogfeldern besitzen alle Microsoft Office-Programme ab der Version Office 2002 (Office XP) ein weiteres interessantes Objekt: Das `FileDialog`-Objekt.

Mit diesem Dialogfeld-Objekt haben Sie in allen Office-Programmen Zugriff auf spezielle Dialogfelder, die denen der integrierten Dialogfelder *Öffnen* und *Speichern unter* entsprechen, aber flexibler in der Anwendung und Konfiguration sind. Zusätzlich stehen Ihnen zwei weitere Dialogfelder zur Datei- und Ordnerauswahl zur Verfügung.

## Übersicht über die verschiedenen *FileDialog*-Typen

Dialog-  
Type

Über die `DialogType`-Eigenschaft dieses Objektes können Sie den Typ und somit die spezifischen Funktionen dieses Objektes auslesen und begrenzt auch festlegen. Der jeweilige Typ des `FileDialog`-Objektes kann durch eine der folgenden benannten `MsoFileDialogType`-Konstanten angesprochen werden kann:

- `msoFileDialogOpen`: Öffnen-Dialogfeld
- `msoFileDialogSaveAs`: Speichern unter-Dialogfeld
- `msoFileDialogFilePicker`: Dateiauswahl-Dialogfeld
- `msoFileDialogFolderPicker`: Ordnerauswahl-Dialogfeld

Über die Eigenschaften des jeweiligen `FileDialog`-Typs können Sie das Erscheinungsbild der Dialogfelder deutlich besser Ihren Wünschen und Anforderungen anpassen als es mit den entsprechenden integrierten Dialogfeldern `wdDialogFileOpen` und `wdDialogFileSaveAs` möglich ist.

In Tabelle 14.4 sind die verfügbaren Eigenschaften aufgeführt.

Tabelle 14.4 Übersicht über die Eigenschaften der *FileDialog*-Objekte

Eigenschaft	Parameter	Zugriff	Beschreibung
<code>AllowMultiSelect</code>	<code>True/False</code>	Lesen Schreiben	Legt fest, ob mehrere Dateien ausgewählt werden können.
<code>Application</code>		Lesen	Gibt die Containeranwendung, in der das <b>FileDialog</b> -Objekt aufgerufen wurde, zurück.
<code>ButtonName</code>	»Bezeichnung«	Lesen Schreiben	Legt den Beschriftungstext der auslösenden Aktionsschaltfläche fest oder gibt ihn zurück.
<code>Creator</code>		Lesen	Eine 32-Bit-Zahl, die die Containeranwendung repräsentiert
<code>DialogType</code>	<code>msoFileDialogOpen</code> <code>msoFileDialogSaveAs</code> <code>msoFileDialogFilePicker</code> <code>msoFileDialogFolderPicker</code>	Lesen Schreiben	Der durch die <b>MsoFileDialogType</b> -Konstante festgelegte <b>FileDialog</b> -Typ

Tabelle 14.4 Übersicht über die Eigenschaften der *FileDialog*-Objekte (Fortsetzung)

Eigenschaft	Parameter	Zugriff	Beschreibung
Filters	FileDialogFilters-Objekt		Gibt eine <b>FileDialogFilters</b> -Auflistung zurück, über die Sie Dateiauswahlfilter definieren können (siehe »Definieren von Dateiauswahlfilter«).
FilterIndex	»Index«	Lesen Schreiben	Legt den zu verwendenden Dateifilter der <b>FileDialogFilters</b> -Auflistung fest oder liefert ihn zurück.
InitialFileName		Lesen Schreiben	Legt den vorgegebenen Dateinamen inkl. Pfad im Dialogfeld fest oder liefert ihn zurück. Dabei können Sie im Dateinamen die Platzhalter »*« und »?« verwenden.
InitialView	msoFileDialogViewDetails msoFileDialogViewLargeIcons msoFileDialogViewList msoFileDialogViewPreview msoFileDialogViewProperties msoFileDialogViewSmallIcons msoFileDialogViewThumbnail	Lesen Schreiben	Legt über die <b>MsoFileDialogView</b> -Konstante die Anzeigoption der Dateien und Ordner im Dialogfeld fest oder liefert sie zurück.
Item		Lesen	Repräsentiert den aktuellen <b>FileDialog</b> -Typ und liefert die Bezeichnung zurück.
SelectedItems		Lesen	Gibt eine <b>FileDialogSelectedItems</b> -Auflistung zurück, in der alle im Dialogfeld markierten Dateien und Ordner enthalten sind, wenn das Dialogfeld mit der <b>Show</b> -Methode angezeigt wird.
Title		Lesen Schreiben	Legt den Titel des Dialogfeldes fest oder liefert ihn zurück.

**ACHTUNG** Im Gegensatz zu den integrierten Dialogfeldern (siehe den Abschnitt »Interne Dialogfelder« in diesem Kapitel) wird die mit dem Dialogfeld verbundene Aktion ausschließlich über die **Execute**-Methode ausgeführt. Die **Show**-Methode zeigt das Dialogfeld an, führt aber keine Aktion aus und entspricht daher eher der **Display**-Methode der integrierten Dialogfelder.

## Dialogfelder »anzeigen« oder »anzeigen und ausführen«

Show  
Execute

Zur Anzeige der `FileDialog`-Dialogfelder stehen Ihnen nur die beiden Methoden `Show` und `Execute` zur Verfügung. Dabei zeigt die `Show`-Methode das Dialogfeld nur an, während die `Execute`-Methode auch die jeweilige Aktion ausführt, die mit dem Dialogfeld verknüpft ist.

In beiden Fällen können Sie über den Rückgabewert beider Methoden gezielt auf die Anwenderaktion reagieren. Dies ist dann wichtig, wenn Sie ein Dialogfeld nur anzeigen lassen und anschließend selbst eine Aktion mit den ausgewählten Dateien oder Ordnern ausführen möchten.

Die Methoden `Show` und `Execute` der Dialogfelder liefern die in Tabelle 14.5 aufgeführten Rückgabewerte.

**Tabelle 14.5** Übersicht über die Rückgabewerte der `FileDialog`-Dialogfelder

Rückgabewert	Beschreibung
-1	Das Dialogfeld wurde über die Aktionsschaltfläche beendet. Bei Verwendung der <code>Show</code> -Methode wird die Aktion dabei nicht ausgeführt.
0	Das Dialogfeld wurde über die Schaltfläche <i>Abbrechen</i> , über das Schließen-Symbol in der Titelleiste oder über die Tastenkombination <code>[Alt] + [F4]</code> beendet.

## Definieren von Dateiauswahlfilter

Filters  
File-  
Dialog-  
Filters

Über die `Filters`-Eigenschaft können Sie für das jeweilige Dialogfeld eigene Dateiauswahlfilter definieren oder auf die vorhandenen Dateiauswahlfilter zugreifen. Diese Eigenschaft liefert in einer `FileDialogFilters`-Auflistung die Eigenschaften für die einzelnen `FileDialogFilters`-Objekte zurück.

### ACHTUNG

Leider lassen sich diese Filter nicht für die Dialogfelder `msoFileDialogSaveAs` und `msoFileDialogFolderPicker` ändern und an eigene Bedürfnisse anpassen.

Filters

Zugriff auf die einzelnen Filter erhalten Sie über die `Filters`-Eigenschaft der Auflistung. Weitere Informationen und Einstellmöglichkeiten erhalten Sie dann über die Eigenschaften des zurückgegebenen `FileDialogFilters`-Objektes.

**Listing 14.15** Auflistung aller aktiven Dateiauswahlfilter

```
Sub subListFileDialogOpenFilters()
    Dim strMSG As String
    Dim dlg As FileDialog
    Dim dlgFlt As FileDialogFilter
    Set dlg = Application.FileDialog(msoFileDialogOpen)
    For Each dlgFlt In dlg.Filters
        strMSG = strMSG & dlgFlt.Description & vbCrLf & "(" & dlgFlt.Extensions & ")" & _
            vbCrLf
    Next dlgFlt
    MsgBox strMSG, vbInformation, "Übersicht über die verfügbaren Dateiauswahlfilter " & _
        vbCrLf & "des 'FileDialogOpen'-Dialogfeldes"
End Sub
```

Abbildg. 14.19 Übersicht über die Dateiauswahlfilter des *FileDialogOpen*-Objektes

**Clear** Über die `Clear`-Methode können Sie die Dateiauswahlliste löschen, um sie z.B. nur mit eigenen Filtern neu zu erstellen. Mit der `Delete`-Methode können Sie einzelne Einträge aus der Dateiauswahlliste über ihren Index entfernen.

**ACHTUNG** Wenn Sie die Liste der Dateiauswahlfilter löschen oder einzelne Einträge aus dieser Liste entfernen, stehen Ihnen diese nicht mehr unmittelbar zur Verfügung, da das `FileDialog`-Objekt an die aktuelle Word-Instanz (Application-Objekt) gebunden ist. Entweder speichern Sie die Filter in einem Array zwischen oder Sie erstellen alle Filter wieder mit der `Add`-Methode.

Erst nach einem Neustart von Word stehen Ihnen die standardmäßig vorgelegten Filter wieder zur Verfügung.

In Listing 14.16 werden zuerst alle Filter des `msoFileDialogOpen`-Dialogfeldes in einem Array gespeichert, bevor die Dateiauswahlliste mittels `Clear` gelöscht und um einen einzelnen Filter ergänzt wird. Nach der Anzeige der anschließend nur noch verfügbaren Filter werden alle ursprünglich vorhandenen Filter wieder der Auflistung hinzugefügt und die Filter erneut angezeigt.

Listing 14.16 Speichern und wiederherstellen der Standard-Dateifilter

```
Sub subResetFileDialogOpenFilter()
    Dim strMSG As String
    Dim strFilters() As String, intIndex As Integer
    intIndex = 0
    Dim dlg As FileDialog
    Dim dlgFlt As FileDialogFilter
    Set dlg = Application.FileDialog(msoFileDialogOpen)
    ReDim strFilters(dlg.Filters.Count, 1)
    For Each dlgFlt In dlg.Filters
        strFilters(intIndex, 0) = dlgFlt.Description
        strFilters(intIndex, 1) = dlgFlt.Extensions
        intIndex = intIndex + 1
    Next dlgFlt
    With dlg.Filters
        .Clear
        .Add "Eigene Word-Dateien", "*.doc", 1
    End With
    For Each dlgFlt In dlg.Filters
        strMSG = strMSG & dlgFlt.Description & vbTab & "(" & dlgFlt.Extensions & ")" & vbCrLf
    Next dlgFlt
    MsgBox strMSG, vbInformation, "Übersicht über die verfügbaren Dateiauswahlfilter " & _
        vbCrLf & "des 'FileDialogOpen'-Dialogfeldes"
    dlg.Filters.Clear
End Sub
```

**Listing 14.16** Speichern und wiederherstellen der Standard-Dateifilter (*Fortsetzung*)

```

For intIndex = LBound(strFilters(), 1) To UBound(strFilters(), 1) - 1
    dlg.Filters.Add strFilters(intIndex, 0), strFilters(intIndex, 1)
Next intIndex
For Each dlgFlt In dlg.Filters
    strMSG = strMSG & dlgFlt.Description & vbTab & "(" & dlgFlt.Extensions & ")" & vbCrLf
Next dlgFlt
MsgBox strMSG, vbInformation, "Übersicht über die verfügbaren Dateiauswahlfilter " & _
    vbCrLf & "des 'FileDialogOpen'-Dialogfeldes"
End Sub

```

## Anwendung des *FileDialog*-Typs *msoFileDialogOpen*

In diesem Abschnitt wird beschrieben, wie Sie das Dialogfeld `msoFileDialogOpen` auf eine bestimmte Erweiterung eingrenzen und die ausgewählten Dateinamen anzeigen lassen können.

Zuerst müssen Sie beim Einsatz dieses Dialogfeldes überlegen, ob Sie eine Mehrfachauswahl von Dateien erlauben möchten und auch weiterverarbeiten können.

Im Beispiel in Listing 14.17 wird die Mehrfachauswahl durch die Angabe der Eigenschaft erlaubt:

Allow-  
Multi-  
Select

```
.AllowMultiSelect = True
```

Als nächster Punkt müssen Sie die notwendigen Filter definieren, wenn Sie nicht auf die Standardfilter zurückgreifen möchten. Das Beispiel verwendet drei Filter für die Auswahl von INI-Dateien, Text-Dateien und zur Auswahl beider Dateitypen in einer Auswahl. Alle weiteren Filter werden vorher durch die Funktion *fstStoreFilters* in einem Array gespeichert und anschließend über die Funktion *fstRestoreFilters* wieder hergestellt.

Initial-  
View

Die weiteren Eigenschaften des Dialogfeldes legen den Titel und den Anzeigetyp der Dateiinformationen fest und ändern den Namen der ausführenden Aktionsschaltfläche *Öffnen* in *Dateien öffnen*.

Anschließend wird das Dialogfeld angezeigt. Nur wenn das Dialogfeld über die Aktionsschaltfläche beendet wird, der Rückgabewert des Dialogfeldes »-1« ist, werden alle markierten Dateinamen inkl. Pfad gesammelt und ausgegeben.

**Listing 14.17** Konfiguration des *msoFileDialogOpen*-Dialogfeldes zur Auswahl von INI- und Text-Dateien

```

Sub subFileDialogOpen()
' Beispiel, in dem die Dateiauswahl auf Text- und INI-Dateien eingeschränkt wird
' und die ausgewählten Dateien am Ende nur angezeigt werden
Dim dlg As FileDialog
Set dlg = Application.FileDialog(msoFileDialogOpen)
fstStoreFilters dlg
With dlg
    .ButtonName = "Dateien öffnen"
    .Title = "Dateiauswahl für INI-Dateien"
    .InitialView = msoFileDialogViewDetails
    .AllowMultiSelect = True

```

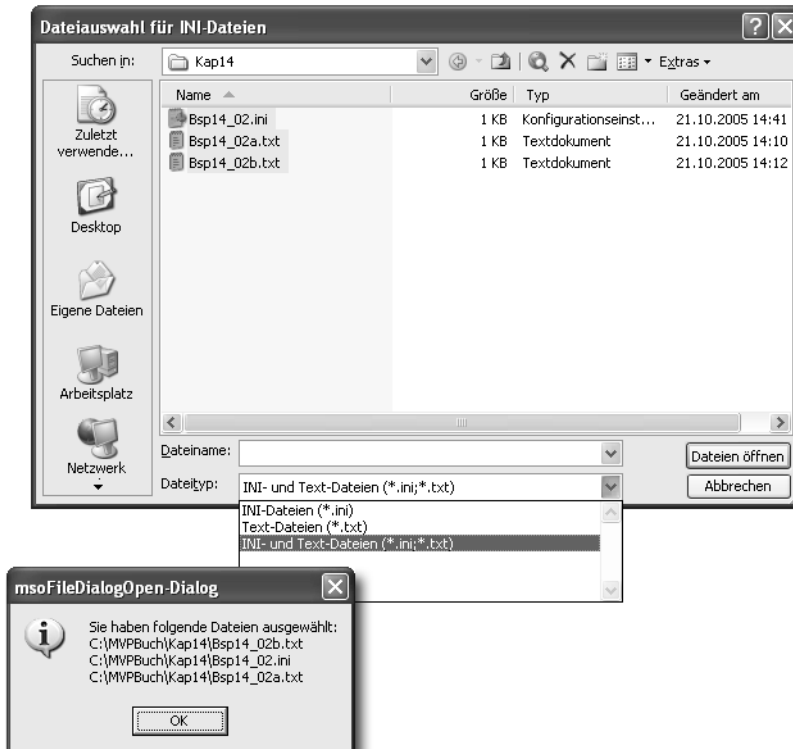
Listing 14.17 Konfiguration des `msoFileDialogOpen`-Dialogfeldes zur Auswahl von INI- und Text-Dateien (Fortsetzung)

```

.InitialFileName = "C:\Beispiele\Kap14\"
.Filters.Clear
.Filters.Add "INI-Dateien", "*.ini"
.Filters.Add "Text-Dateien", "*.txt"
.Filters.Add "INI- und Text-Dateien", "*.ini;*.txt"
.FilterIndex = 3
If .Show = -1 Then
    For intFile = 1 To .SelectedItems.Count
        strMSG = strMSG & .SelectedItems(intFile) & vbCrLf
    Next
    MsgBox "Sie haben folgende Dateien ausgewählt: " & vbCrLf & strMSG, _
        vbInformation, "msoFileDialogOpen-Dialog"
End If
End With
fktRestoreFilters dlg
End Sub

```

Abbildg. 14.20 Auswahl mehrerer Dateien und anschließende Anzeige der ausgewählten Dateipfade



## Anwendung des *FileDialog*-Typs *msoFileDialogSaveAs*

Dieser Abschnitt beschreibt, wie Sie das Dialogfeld `msoFileDialogSaveAs` konfigurieren und verwenden können, um die aktuelle Datei nach Rückfrage im XML-Dateiformat abzuspeichern.

Für dieses Dialogfeld können Sie keine Mehrfachauswahl festlegen, da diese Eigenschaft zum Speichern einer Datei nicht sinnvoll ist.

In Listing 14.18 wird neben der Festlegung des Titels und dem Ansichtstyp die Aktionsschaltfläche in *Datei speichern* geändert.

Damit der Dateiname des aktuellen Dokumentes mit der korrekten Erweiterung für XML-Dateien ».xml« gespeichert wird, muss zuerst überprüft werden, ob die Dateierweiterung angezeigt wird. Ohne auf APIs (siehe Kapitel 4) zurückgreifen zu müssen, können Sie dies in einer `If...Then...Else`-Anweisung oder auch mit der `IIf`-Funktion überprüfen:

```
strFilename = IIf(Right(ActiveDocument.Name, 4) = ".doc", _
    Left(ActiveDocument.Name, Len(ActiveDocument.Name) - 4) & ".xml", _
    ActiveDocument.Name & ".xml")
```

Mit dieser Abfrage wird überprüft, ob die Dateierweiterung ».doc« im Dateinamen vorkommt. Ist dies der Fall, wird diese durch die Endung ».xml« ersetzt, andernfalls wird die Endung an den Dateinamen angehängt.

Im nächsten Schritt werden die vorhandenen Dateifilter dahingehend überprüft, ob ein geeigneter Filter für XML-Dateien vorhanden ist. Ist dies der Fall, wird er als verwendeter Dateityp festgelegt, andernfalls wird der Speicher-Vorgang beendet.

Wenn ein entsprechender Dateifilter existiert, wird das Dialogfeld mit den Einstellungen angezeigt. Wird eine Datei angegeben, was auch in einem anderen als dem vorgegebenen Pfad möglich ist, und über die Aktionsschaltfläche *Datei speichern* die Angabe bestätigt, wird ein Bestätigungsdialog angezeigt. In diesem werden Sie gefragt, ob die Datei tatsächlich in dem angegebenen Verzeichnis unter dem angegebenen Dateinamen gespeichert werden soll. Erst nach Bestätigung dieser Abfrage wird das aktuelle Dokument mittels der `SaveAs`-Eigenschaft als XML-Datei unter dem angegebenen Dateinamen gespeichert.

**Listing 14.18** Speichern der aktuellen Datei im XML-Format

```
Sub subFileDialogSaveAs()
' Beispiel, in dem das aktuelle Dokument nach Rückfrage als XML-Datei gespeichert wird
Dim dlg As FileDialog
Dim intFlt As Integer
Dim ret As Integer
Dim strFilename As String
Set dlg = Application.FileDialog(msoFileDialogSaveAs)
With dlg
    .ButtonName = "Datei speichern"
    .Title = "Speichern von XML-Dateien"
    .InitialView = msoFileDialogViewDetails
    .AllowMultiSelect = False
    strFilename = IIf(InStr(1, ActiveDocument.Name, ".doc"), _
        Replace(ActiveDocument.Name, ".doc", ".xml"), ActiveDocument.Name & ".xml")
```



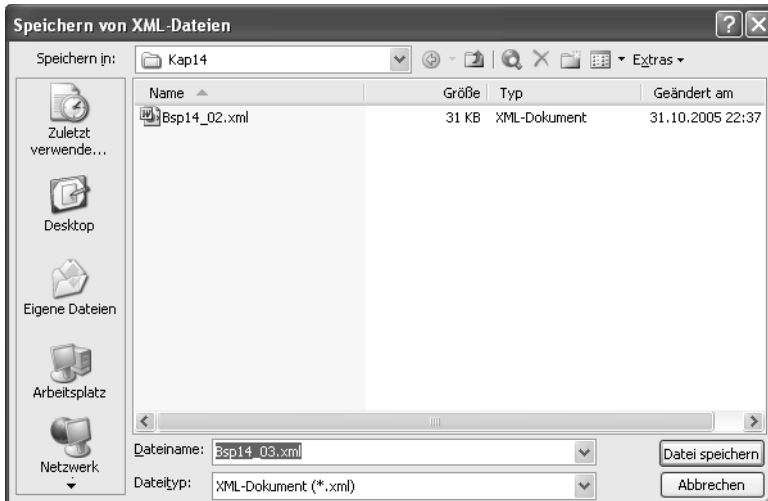
Listing 14.18 Speichern der aktuellen Datei im XML-Format (Fortsetzung)

```

.InitialFileName = "C:\Beispiele\Kap14\" & strFilename
For intFlt = 1 To dlg.Filters.Count
    If dlg.Filters(intFlt).Extensions = "*.xml" Then
        .FilterIndex = intFlt
    Exit For
End If
Next intFlt
If CStr(.Filters(.FilterIndex).Extensions) <> "*.xml" Then
    MsgBox "Es konnte kein XML-Dateifilter gefunden werden." & vbCrLf & _
        "Das Speichern wird abgebrochen.", vbCritical, "msoFileDialogSaveAs-Dialog"
    Exit Sub
End If
If .Show = -1 Then
    ret = MsgBox("Soll die aktuelle Datei als XML-Datei in folgendem Pfad " & _
        "gespeichert werden?" & vbCrLf & .InitialFileName, vbQuestion + _
        vbYesNo, "msoFileDialogSaveAs-Dialog")
    If ret = vbYes Then
        ActiveDocument.SaveAs FileName:=.InitialFileName, fileformat:=wdFormatXML
    End If
End If
End With
Set dlg = Nothing
End Sub

```

Abbildg. 14.21 Dialogfeld zum Speichern der aktuellen Datei im XML-Format



## Anwendung des *FileDialog*-Typs *msoFileDialogFilePicker*

Das Dialogfeld `msoFileDialogFilePicker` ermöglicht es dem Anwender, eine oder mehrere Dateien auszuwählen. Im Gegensatz zum `msoFileDialogOpen`-Dialogfeld besitzt dieses Dialogfeld keine wirkliche Aktion, sondern liefert nur den Namen und Pfad der ausgewählten Dateien wieder.

Die weiteren Konfigurationsmöglichkeiten, auch in Bezug auf die Einschränkung oder Veränderung der Dateifilter, entsprechen ebenfalls denen des `msoFileDialogOpen`-Dialogfeldes (siehe den Abschnitt »Anwendung des *FileDialog*-Typs *msoFileDialogOpen*« in diesem Kapitel), weshalb an dieser Stelle nicht weiter auf diesen `FileDialog`-Typ eingegangen wird.

## Anwendung des *FileDialog*-Typs *msoFileDialogFolderPicker*

Das Dialogfeld `msoFileDialogFolderPicker` erlaubt nur die Auswahl eines Ordners. Wie das `msoFileDialogOpen`-Dialogfeld führt auch dieses keine Aktion aus, sondern liefert nur den ausgewählten Ordnersnamen inklusive Pfad über die `Show`-Methode zurück. Für dieses Dialogfeld können auch weder Filter festgelegt noch mehrere Ordner gleichzeitig ausgewählt werden.

Im Beispiel in Listing 14.19 wird nach Festlegung des Titels und Ansichtstyps die Aktionsschaltfläche in *Ordner einlesen* geändert. Denn nach Bestätigung der Auswahl über diese Schaltfläche werden in der Funktion *fktReadDirFiles* alle im ausgewählten Ordner enthaltenen Dateien ermittelt und mit ihren Dateiattributen, wie z.B. »Ordner« oder »Schreibgeschützt«, angezeigt.

Dir-  
Funktion

Dazu durchläuft die Funktion *fktReadDirFiles* mit Hilfe der `Dir`-Funktion (siehe Kapitel 3) alle Dateien des ausgewählten Ordners, ermittelt die Dateiattribute und speichert die Daten in einem Array. Dieses Array wird dann über den Funktionsnamen an die aufrufende Prozedur zurückgeliefert, wo die Einträge im Array ausgelesen und in einem Dialogfeld angezeigt werden.

**Listing 14.19** Auswahl eines Ordners und Anzeige aller im Ordner enthaltenen Dateien mit Attributen

```
Sub subFileDialogFolderPicker()
' Beispiel, das alle Dateien im ausgewählten Ordner
' mit den jeweiligen Dateiattributen anzeigt
Dim dlg As FileDialog
Dim strFiles() As String
Dim intFile As Integer
Set dlg = Application.FileDialog(msoFileDialogFolderPicker)
fktStoreFilters dlg
With dlg
    .ButtonName = "Ordner einlesen"
    .Title = "Ordner einlesen"
    .InitialView = msoFileDialogViewLargeIcons
    .InitialFileName = "C:\Beispiele\"
    If .Show = -1 Then
        strFiles() = fktReadDirFiles(.SelectedItems(1))
        strMSG = "Attribut" & vbTab & "Datei" & vbCrLf
        For intFile = LBound(strFiles(), 2) To UBound(strFiles(), 2) - 1
            strMSG = strMSG & strFiles(1, intFile) & vbTab & strFiles(0, intFile) & vbCrLf
        Next intFile
        MsgBox strMSG, vbInformation, "Inhalt von " & .SelectedItems(1)
    End If
End With
End Sub

Function fktReadDirFiles(strDir As String) As Variant
Dim strFile As String, strAttr As String
Dim strFiles() As String
ReDim strFiles(1, 0)
strDir = IIf(Right(strDir, 1) = "\", strDir, strDir & "\")
```

Listing 14.19 Auswahl eines Ordners und Anzeige aller im Ordner enthaltenen Dateien mit Attributen (Fortsetzung)

```

strFile = Dir(strDir, vbDirectory) ' Ersten Eintrag abrufen.
Do While strFile <> "" ' Schleife beginnen.
    ' Aktuelles und übergeordnetes Verzeichnis ignorieren.
    If strFile <> "." And strFile <> ".." Then
        strAttr = ""
        ReDim Preserve strFiles(1, UBound(strFiles(), 2) + 1)
        strFiles(0, UBound(strFiles(), 2) - 1) = strFile
        ' Attribute in Kürzel umsetzen
        If (GetAttr(strDir & strFile) And vbReadOnly) = vbReadOnly Then
            strAttr = strAttr & "R"
        End If
        If (GetAttr(strDir & strFile) And vbHidden) = vbHidden Then
            strAttr = strAttr & "H"
        End If
        If (GetAttr(strDir & strFile) And vbSystem) = vbSystem Then
            strAttr = strAttr & "S"
        End If
        If (GetAttr(strDir & strFile) And vbDirectory) = vbDirectory Then
            strAttr = strAttr & "D"
        End If
        If (GetAttr(strDir & strFile) And vbArchive) = vbArchive Then
            strAttr = strAttr & "A"
        End If
        If (GetAttr(strDir & strFile) And vbAlias) = vbAlias Then
            strAttr = strAttr & "L"
        End If
        strFiles(1, UBound(strFiles(), 2) - 1) = strAttr
    End If
    strFile = Dir ' Nächsten Eintrag abrufen.
Loop
fktReadDirFiles = strFiles()
End Function

```

Abbildg. 14.22 Anzeige aller Dateien im ausgewählten Ordner mit ihren Dateiattributen



Die dargestellten Beispiele finden Sie in der Beispieldatei *Bsp14\_04.doc* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap14*.

# Zusammenfassung

In diesem Kapitel wurde der Umgang mit Dialogfeldern aufgezeigt. Dazu gehören nebst den benutzerdefinierten Dialogfeldern (UserForm-Objekt) auch die internen Dialogfelder von Word:

- Im Kapitel wurde aufgezeigt, wie ein benutzerdefiniertes Dialogfeld angelegt und in verschiedenen Projekten genutzt werden kann (Seite 609 ff.)
- Es wurden die wichtigsten Eigenschaften (Seite 610 ff.), Methoden (Seite 611 ff.) und Ereignisse (Seite 614 ff.) des UserForm-Objekt vorgestellt.
- Es wurden die allgemeinen Steuerelemente aus der »Microsoft Forms 2.0 Object Library« vorgestellt (Seite 618 ff.) und aufgezeigt, wie zusätzliche Steuerelemente eingebunden werden können (Seite 620 ff.).
- Eine Zusammenfassung, welche Anforderungen der Anwender an ein Dialogfeld hat (Seite 621 ff.) und wie ein benutzerdefiniertes Dialogfeld zur Laufzeit beeinflusst werden kann (Seite 626 ff.), schließen den ersten Teil des Kapitels ab.
- Nach den benutzerdefinierten Dialogfeldern wurde im folgenden Abschnitt des Kapitels gezeigt, wie Sie auf die integrierten Dialogfelder von Word zugreifen (Seite 631 ff.) und über die zugehörigen Argumente konfigurieren können (Seite 634). Dabei wurde hervorgehoben, welche Unterschiede in den verschiedenen Aufruf-Methoden bestehen.
- Der letzte Abschnitt stellte die verschiedenen FileDialog-Dialogfelder vor (Seite 638). Anhand von Beispielen wurden die Einstellmöglichkeiten und Einschränkungen aufgezeigt (Seite 642 ff.).

## Kapitel 15

# Menüs und Symbolleisten

### In diesem Kapitel:

CommandBars	650
Symbolschaltflächen	655
Symbolleisten erstellen	661
Zusammenfassung	664

In Microsoft Office spielen Menüs und Symbolleisten eine wichtige Rolle. Nicht nur sind fast alle internen Anwendungsbefehle darüber zu erreichen. Auch der Entwickler stellt dadurch seine Werkzeuge zur Verfügung.

---

**HINWEIS** Obwohl es auf den ersten Blick vielleicht nicht offensichtlich ist, sind auch Menüs Symbolleisten mit Symbolschaltflächen.

---

Word unterstützt, wie in Kapitel 13 beschrieben, die Erstellung von Symbolleisten für den globalen Gebrauch, oder für dokumentspezifische Aufgaben. Solange der Code hinter den Symbolschaltflächen sich in der gleichen Datei mit der Symbolleiste befindet, werden Symbolleisten eher selten programmtechnisch, sondern über die Benutzerschnittstelle (Menübefehl *Extras/Anpassen*) bei der Vorbereitung der Vorlage oder Datei erstellt. Für die Fernsteuerung außerhalb von Word sieht die Lage anders aus. Ein COM-Add-In oder VSTO-Lösung beispielsweise wird die für die Aufgabe benötigten Symbolleisten meistens (beim Laden) dynamisch erstellen.

In diesem Kapitel werden wir zuerst die allgemein beim Programmablauf nützlichen Eigenschaften der Symbolleisten vorstellen. Danach werden die Symbolschaltflächen behandelt. Und abschließend zeigen wir ein Beispiel, wie eine Symbolleiste vom Grund auf programmtechnisch erstellt wird.

## CommandBars

Im Objektmodell stellt das Objekt `CommandBar` eine Symbol- oder Menüleiste dar. Ein `CommandBar` ist eine Eigenschaft des `Application`-Objekts, stammt jedoch vom *Office*-Objektmodell, was bei der Deklaration einer Objektvariablen offensichtlich wird. Die Symbolleisten eines *Kontextes* werden in einer `CommandBars`-Auflistung geführt. Eine spezifische Symbolleiste kann durch einen numerischen Indexwert oder durch seine **englische** Beschriftung angesprochen werden:

```
Dim cb1 as Office.CommandBar
Dim cb2 as Office.CommandBar
Set cb1 = Application.CommandBars(1)
Set cb2 = Application.CommandBars("Menu Bar")
```

Bezeichnung

Das Objektmodell stellt für das `CommandBar`-Objekt drei Eigenschaften zur Verfügung, womit der Indexwert sowie die englische und deutsche Bezeichnung ermittelt werden können: `Index`, `Name` bzw. `NameLocal`. Zudem ist die Eigenschaft `BuiltIn` nützlich, um festzustellen, ob es sich um eine benutzerdefinierte oder Word-interne Symbolleiste handelt. Eine Tabelle der im gegenwärtigen Kontext zur Verfügung stehenden Symbolleisten ist damit schnell mit einer Prozedur wie die in Listing 15.1 erstellt.

---

**HINWEIS** Eine entsprechende Liste für die standardmäßige Dokumentvorlage *Normal.dot* finden Sie im Anhang C.

---

**Listing 15.1**

Eine Tabelle der verfügbaren Symbolleisten generieren, die die englischen und deutschen Bezeichnungen gegenüber stellt

```
Sub SymbolleistenAuflisten()
    Dim cb As Office.CommandBar
    Dim s As String
```

**Listing 15.1** Eine Tabelle der verfügbaren Symbolleisten generieren, die die englischen und deutschen Bezeichnungen gegenüber stellt (Fortsetzung)

```

Dim sTrenn As String
Dim rng As Word.Range
Dim tbl As Word.Table

sTrenn = ";"
'Tabellenüberschriften
s = "Deutsche Bezeichnung" & sTrenn & "Englische Bezeichnung" & _
sTrenn & "Index" & sTrenn & "Benutzerdefiniert"
For Each cb In Application.CommandBars
    s = s & vbCrLf & cb.NameLocal & sTrenn & cb.Name & sTrenn & _
        cb.Index & sTrenn & (Not cb.BuiltIn)
Next
'Eine Tabelle wird anstelle der gegenwärtigen Markierung erstellt.
Set rng = Selection.Range
rng.Text = s
Set tbl = rng.ConvertToTable(sTrenn)
tbl.Rows(1).Range.Bold = True
End Sub

```



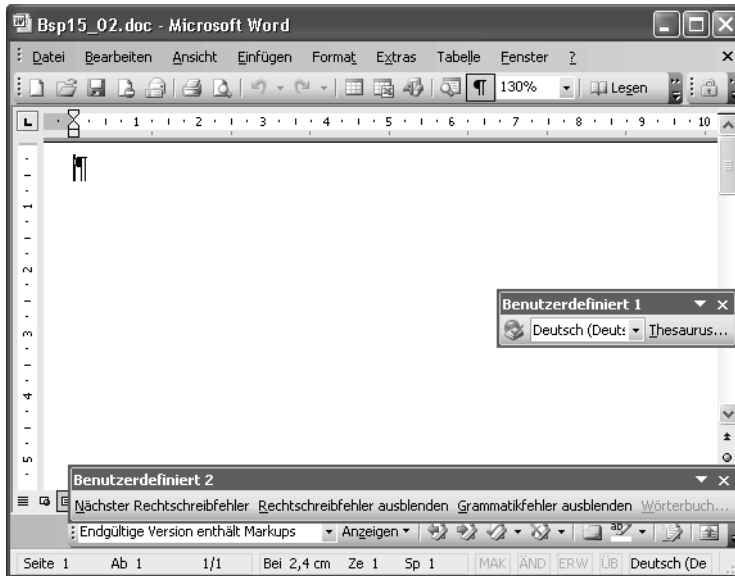
Die Beispieldatei *Bsp15\_01.doc* finden Sie auf der CD-ROM zum Buch im Ordner \Beispiele\Kap15.

**Typ** Das Objektmodell führt auch eine Type-Eigenschaft für das CommandBar-Objekt auf. Dafür gibt es drei Konstanten: `msoBarTypeMenuBar`, `msoBarTypeNormal` sowie `msoBarTypePopup`. Lediglich eine Symbolleiste pro Kontext darf als Menüleiste bezeichnet werden. Die Menüs, die beim Anklicken aufklappen, sind Symbolleisten des Typs `msoBarTypePopup`. Die restlichen sind normale Symbolleisten. Type ist nur lesbar; der Typ einer Symbolleiste wird bei ihrer Erstellung festgelegt.

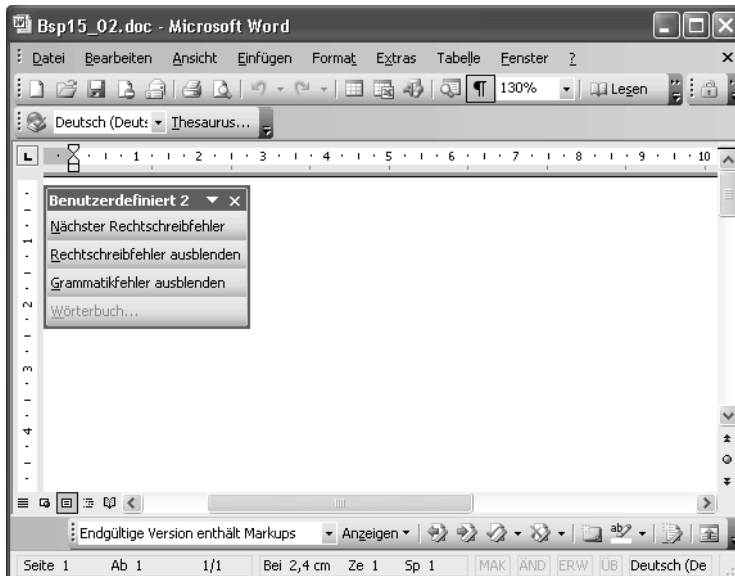
**Positionierung** Symbolleisten können entweder an allen vier Seiten des Dokumentfensters verankert oder frei auf dem Bildschirm positioniert werden. Im Objektmodell wird dies über die Eigenschaft `Position` geregelt, in Kombination mit Eigenschaften wie `Left` (links), `Top` (oben), `Width` (Breite) und `Height` (Höhe). Zusätzlich wird für Symbolleisten, die angedockt werden, mit `RowIndex` die jeweilige Zeile festgelegt, wie die Abbildung 15.1, die Abbildung 15.2 sowie das Listing 15.2 verdeutlichen.

Das Code-Beispiel veranschaulicht die Verwendung dieser Eigenschaften. Die Symbolleiste »Benutzerdefiniert 1« wird am oberen Rand (`msoBarTop`), ganz links (`Left = 0`), in der Zeile unter allen anderen oben angedockten Symbolleisten (`RowIndex`) angedockt. Die zweite Symbolleiste bleibt freistehend (`msoBarFloating`), über der linken oberen Ecke des Dokuments. Ihre Symbolschaltflächen werden, durch Festlegung der Breite, untereinander aufgereiht.

**Abbildg. 15.1** Beliebige Ausganglage: die Symbolleisten befinden sich »irgendwo«



**Abbildg. 15.2** Nach Ausführung des Positionierungscodes





**HINWEIS** Bitte bedenken Sie, dass eine genaue Positionierung von Symbolleisten ein recht komplexes Unterfangen ist. Die Ausgangslage – wie viele und welche Symbolleisten wo platziert sind – wird von Benutzer zu Benutzer variieren. Zudem beeinflusst die Bildschirmauflösung die Wirkung der Maßangaben.

Listing 15.2 Symbolleisten positionieren

```
Sub SymbolleistePositionieren()
    Dim cb1 As Office.CommandBar
    Dim cb2 As Office.CommandBar
    Dim doc As Word.Document
    Dim win As Word.Window
    Dim rng As Word.Range
    Dim lLinks As Long, lTop As Long, lWidth As Long, lHeight As Long
    Dim sngKorrektur As Single

    Set cb1 = Application.CommandBars("Benutzerdefiniert 1")
    Set cb2 = Application.CommandBars("Benutzerdefiniert 2")
    Set doc = ActiveDocument
    Set win = ActiveWindow

    'Die erste Symbolleiste ganz links andocken
    'in der nächsten, leeren Zeile, im oberen Teil
    'ganz links andocken
    cb1.Position = msoBarTop
    cb1.RowIndex = IndexErmitteln + 1
    'Dafür sorgen, dass sie die erste der Zeile ist
    cb1.Left = 0

    'Die zweite Symbolleiste nahe des oberen linken Fensterrahmens positionieren
    cb2.Position = msoBarFloating
    cb2.Width = 100
    cb2.Left = Application.Left + 30
    Set rng = doc.Range(0, 0)
    win.GetPoint lLinks, lTop, lWidth, lHeight, rng
    If win.View.DisplayPageBoundaries Then
        With doc.Sections(1).PageSetup
            sngKorrektur = .TopMargin + .HeaderDistance
        End With
    End If
    cb2.Top = CSng(lTop) - sngKorrektur
End Sub

Function IndexErmitteln() As Long
    Dim cb As Office.CommandBar
    Dim index As Long

    For Each cb In Application.CommandBars
        If cb.Position = msoBarTop Then
            If cb.RowIndex > index Then
                index = cb.RowIndex
            End If
        End If
    Next
    IndexErmitteln = index
End Function
```

Schutz Wird Code benutzt, um Symbolleisten zu positionieren, ist es möglich, dass sie vom Anwender nicht verschoben oder angepasst werden dürfen. Wenn der Anwender gar nichts mit einer Symbolleiste machen soll, muss ihre `Enabled`-Eigenschaft auf `False` festgelegt werden. Somit wird sie auch unsichtbar und im Untermenübefehl *Ansicht/Symbolleisten* nicht aufgelistet.

Im Gegensatz dazu blendet die Eigenschaft `Visible` (sichtbar) eine Symbolleiste ein oder aus, bleibt aber im Untermenü zum Befehl *Ansicht/ Symbolleisten* verfügbar.

Eine etwas verfeinerte Kontrolle bietet die Eigenschaft `Protection` an. Die in Tabelle 15.1 aufgelisteten `MsoBarProtection`-Konstanten dürfen nicht nur einzeln, sondern miteinander addiert zugewiesen werden. Soll beispielsweise eine Symbolleiste nicht ausgeblendet und auch nicht am rechten oder linken Rand angedockt werden dürfen, verwenden Sie die folgende Anweisung:

```
cb.Protection = msoBarNoVerticalDock + msoBarNoChangeVisible
```

**Tabelle 15.1** Mögliche Schutzeinstellungen für Symbolleisten

MsoBarProtection-Enum	Wert	Beschreibung
<code>msoBarNoChangeDock</code>	16	Die Symbolleiste kann nicht anderswo angedockt werden.
<code>msoBarNoChangeVisible</code>	8	Die Symbolleiste kann nicht ausgeblendet werden, wenn sie sichtbar ist, bzw. nicht eingeklappt werden, wenn sie nicht sichtbar ist.
<code>msoBarNoCustomize</code>	1	Die Symbolleiste kann weder über den Word-Befehl <i>Extras/Anpassen</i> noch mit VBA geändert werden.
<code>msoBarNoHorizontalDock</code>	64	Die Symbolleiste darf weder oben noch unten angedockt werden.
<code>msoBarNoMove</code>	4	Die Symbolleiste kann überhaupt nicht verschoben werden.
<code>msoBarNoProtection</code>	0	Es sind keine Schutzmaßnahmen aktiviert.
<code>msoBarNoResize</code>	2	Die Größe der Symbolleiste darf nicht geändert werden.
<code>msoBarNoVerticalDock</code>	32	Die Symbolleiste darf weder links noch rechts angedockt werden.

```
DisableCustomize
```

Falls es dem Anwender untersagt werden soll, irgendwelche Anpassungen vorzunehmen, kann die `DisableCustomize`-Eigenschaft der `CommandBars`-Auflistung auf `True` festgelegt werden. Damit wird der Menübefehl *Extras/Anpassen* abgeblendet dargestellt und ist somit nicht mehr wählbar. Bitte beachten Sie, dass diese Eigenschaft kontextbezogen ist. Bestimmen Sie daher sorgfältig die `CustomizationContext`-Eigenschaft, bevor `DisableCustomize` festgelegt wird:

```
Application.CustomizationContext = ActiveDocument
Application.CommandBars.DisableCustomize = True
```



Die Beispieldatei *Bsp15\_02.doc* finden Sie auf der CD-ROM zum Buch im Ordner `\Beispiele\Kap15`.

Adaptive  
Menus  
Adaptive  
Menu

Die Eigenschaft `AdaptiveMenus` gehört der Auflistung `CommandBars` des `Application`-Objekts und entspricht der Option *Menüs immer vollständig anzeigen* unter *Extras/Anpassen/Optionen*. Ist die Einstellung `True`, werden nur die zuletzt oder am häufigsten benutzten Befehle in den Symbolleisten angezeigt.

Priority

Die Eigenschaft `AdaptiveMenu` bezieht sich auf eine einzelne Symbolleiste (`CommandBar`) und legt dieses Verhalten dafür fest. Die Eigenschaft kommt jedoch nur zur Geltung, wenn `AdaptiveMenus` für die ganze Umgebung `True` ist.

Damit verwandt ist die Symbolschaltflächen- (`Control`) Eigenschaft `Priority`. Diese legt fest, welche Symbolschaltflächen in jedem Fall sichtbar sein sollen, egal wie oft sie vom Anwender ausgeführt werden. Eine interessante Nebenwirkung ist, dass damit *alle* Symbolschaltflächen einer Symbolleiste zwingend eingeblendet werden können, auch wenn dadurch eine angedockte Symbolleiste über mehrere Zeilen umbrochen wird.

Disable-  
Aska-  
Question  
Dropdown

Auch interessant ist die Eigenschaft `DisableAskAQuestionDropdown` der `CommandBars`-Auflistung. Wird ihr der Wert `True` zugewiesen, verschwindet das Eingabefeld *Frage hier eingeben* aus der Menüleiste. Im Gegensatz zu `DisableCustomize` bezieht sich diese Eigenschaft immer auf die gesamte Word-Umgebung.

## Symbolschaltflächen

Ohne Symbolschaltflächen ist eine Symbolleiste bedeutungslos. Die meisten Schaltflächen, die Sie in der Benutzerschnittstelle sehen, sind des Typs `msoControlButton` (entspricht dem `CommandBarButton`-Objekt im Word-Objektmodell).

Wer im Objektkatalog des VB-Editors die Liste der `MsoControlType` betrachtet und auf hochfliegende Gedanken kommt, was alles damit angestellt werden könnte, wird schnell wieder auf den Boden der Realität geholt. Von den 27 aufgelisteten Konstanten stehen davon lediglich vier dem VBA-Entwickler für die Definition eigener Symbolleisten zur Verfügung: `msoControlButton`, `msoControlComboBox`, `msoControlEdit` sowie `msoControlPopup`. (Die restlichen sind nur lesbar, als Rückgabewert bei Prüfung des Typs.)

### HINWEIS

Ein Entwickler, der Word von der .NET-Umgebung aus automatisiert, kann Schaltflächen der anderen `MsoControlButton`-Typen definieren.

10

Während die internen Symbolleisten von Word über die englische Bezeichnung angesprochen werden müssen, ist der beschreibende Indexwert einer Symbolschaltflächen, lokalisiert; die Beschriftung (`Caption`-Eigenschaft) ist gleichzeitig ein Indexwert.

Dieser Umstand würde, was die Word-eigenen Symbolschaltflächen betrifft, eine Internationalisierung fast zu einem Ding der Unmöglichkeit machen, gäbe es nicht die `ID`-Eigenschaft des `Control`-Objekts. Haben Sie den `ID`-Wert einer Word-internen Symbolschaltfläche ermittelt, kann diese, egal in welcher Sprachumgebung, durch den Einsatz der Methode `FindControl` eindeutig angesprochen werden.

Ein Beispiel hilft, dies zu veranschaulichen. In der `Dialogs`-Auflistung befindet sich keine Konstante für das Dialogfeld zum Menübefehl *Datei/Eigenschaften*. Folglich ist es scheinbar nicht möglich, die-

ses programmtechnisch einzublenden. Es gibt jedoch einen Weg: über die Methode `Execute` einer Symbolschaltfläche kann deren Befehl direkt ausgeführt werden.

Um den ID-Wert einer Symbolschaltfläche zu ermitteln:

```
Dim IID as Long
IID = Application.CommandBars("Menu Bar").Controls( _
    "Datei").CommandBar.Controls("Eigenschaften").Id - 'Ergebnis: 750
```

**PROFITIPP**

Bitte beachten Sie, wie untere Menüebenen angesprochen werden. Ein `Control`-Objekt des Typs `msoControlPopup` besitzt eine `CommandBar`-Eigenschaft (auch wenn sie nicht in der `IntelliSense`-Liste erscheint). Diese ihrerseits hat ebenfalls eine `Control`-Eigenschaft. Ist dieses Steuerelement ebenfalls vom Typ `msoControlPopup`, geht's weiter im gleichen Stil, bis eine Symbolschaltfläche einer anderen Art vorliegt, die ausführbar ist.

Find-  
Control  
Execute

Mit `FindControl` wird die Symbolschaltfläche über den ID-Wert angesprochen, und mit `Execute` wird sie ausgeführt, genau so, als ob der Anwender darauf geklickt hätte. In unserem Beispiel wird das Dialogfeld, egal in welcher Sprachumgebung, eingeblendet:

```
Application.CommandBars.FindControl(ID:=750).Execute
```

Enabled,  
Visible

Während eines Programmablaufs sind sonst hauptsächlich die Eigenschaften `Visible` (sichtbar) und `Enabled` (zugänglich) interessant. Im Gegensatz zu den gleichnamigen Eigenschaften des `CommandBar`-Objekts bleibt eine Symbolschaltfläche sichtbar, wenn `Enabled` auf `False` gesetzt ist. Allerdings wird sie abgeblendet dargestellt und ist somit nicht mehr wählbar.

Eine unsichtbare Schaltfläche verschwindet aus der Symbolleiste, kann jedoch weiterhin mit `Execute` ausgeführt werden.

## Symbolschaltflächen erstellen

In der Benutzerschnittstelle werden Symbolschaltflächen über den Menübefehl *Extras/Anpassen* auf der Registerkarte *Befehle* definiert. Der Word-Befehl oder das Makro wird aus der Liste *Befehle* (rechts in Abbildung 15.3) in die Symbolleiste oder Menü gezogen, wo die Symbolschaltfläche stehen soll. Über die Schaltfläche *Auswahl ändern* wird das Aussehen angepasst: die Beschriftung (`Caption`) wird festgelegt und bestimmt, ob und welches Symbol angezeigt werden soll (`Style`).

**HINWEIS**

Obwohl in der Word-Umgebung vier Steuerelementtypen definiert werden können, lassen sich über die Benutzerschnittstelle nur zwei erstellen: eine Symbolschaltfläche zum Anklicken (`msoControlButton`) und ein Untermenü (`msoControlPopup`). Um ein Untermenü zu erstellen, wird im Dialogfeld *Anpassen* auf der Registerkarte *Befehle* aus der Liste *Kategorien* der Eintrag *Neues Menü* gewählt. Dann wird der Befehl *Neues Menü* in eine Symbolleiste, Menüleiste oder Popup-Menü gezogen.

Abbildg. 15.3 Symbolleisten in der Benutzeroberfläche definieren



Über die Automatisierungsschnittstelle stehen mehr Gestaltungsmöglichkeiten zur Verfügung. Quick-Info (ToolTipText), Tastaturzuweisung (ShortcutText), Sichtbarkeit (Visible) sowie Zugang (Enabled) können einer Schaltfläche zusätzlich zugewiesen werden.

**Add-Methode** Eine neue Symbolschaltfläche wird mit der Add-Methode der Controls-Auflistung hinzugefügt, die folgende Syntax aufweist:

Add(Type, Id, Parameter, Before, Temporary)

Das Type-Argument ist das einzig erforderliche; es erwartet irgendeinen der vier MsoControlType-Werte. ID wird nur gebraucht, wenn die Symbolschaltfläche einen Word-internen Befehl ausführen soll (wie der ID-Wert ermittelt wird, wurde weiter oben beschrieben). Mit Parameter kann eine unsichtbare Zeichenkette mitgespeichert werden. Um die Symbolschaltfläche an einer bestimmten Stelle einzufügen, wird dem Argument Before eine Ganzzahl zugewiesen, die die Ordinalzahl in der Symbolleiste angibt. Temporary hat keine Bedeutung in der Word-Umgebung; eine Schaltfläche wird nie automatisch beim Schließen des Kontextes entfernt.

#### HINWEIS

Wenn Sie außerhalb von Word Symbolschaltflächen, die in mehreren Dokumenten benutzt werden, erstellen und über ein Ereignis mit Code verbinden, müssen Sie der Tag-Eigenschaft jeder Symbolschaltfläche den gleichen eindeutigen Wert zuweisen. Sonst werden sie nur im ersten Dokument lauffähig sein. Mehr Informationen finden Sie im Artikel »Hooking a COM Add-in Up to a Command Bar Control« unter der URL <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/odeopg/html/deovrhookingcomaddinuptocommandbarcontrol.asp>.

**Word-interner Befehl** Um den Word-Formatierungsbefehl »Kursiv« einer eigenen Symbolleiste an erster Stelle zuzuweisen, können Sie die folgenden Anweisungen verwenden:

```
Application.CustomizationContext = ActiveDocument
Commandbars("Meine Befehle").Controls.Add(msoControlButton, 114, 1)
```

**OnAction** Das Listing 15.3 veranschaulicht, wie man dem allgemeinen Kontextmenü *Text* eine Schaltfläche zuweist, die mit einem Makro verbunden ist. Wird eine Symbolschaltfläche nicht mit einem Word-internen Befehl verbunden, muss der *OnAction*-Eigenschaft der Name einer Prozedur zugewiesen werden, wenn die Schaltfläche irgendeine Handlung ausführen soll.

**Listing 15.3** Eine Schaltfläche im Kontextmenü erstellen, die mit einem Makro verbunden ist

```
Sub SymbolschaltflächeDefinieren1()
    Dim ctl As Office.CommandBarButton

    Application.CustomizationContext = ActiveDocument
    'Set ctl = CommandBars("Meine Befehle").Controls.Add(msoControlButton, 114, 1)
    Set ctl = CommandBars("Text").Controls.Add(msoControlButton)
    ctl.Caption = "Dokumentschutz aktivieren"
    ctl.OnAction = "DokumentschutzAktivieren"
    ctl.DescriptionText = _
        "Aktiviert 'Änderungen verfolgen' und schützt das Dokument für Revisionen"
    ctl.Style = msoButtonCaption
    ctl.TooltipText = "Dokument für Revisionen schützen"
End Sub

Sub DokumentSchutzAktivieren()
    Dim doc As Word.Document

    Set doc = ActiveDocument
    doc.TrackRevisions = True
    If doc.ProtectionType = wdNoProtection Then
        doc.Protect Type:=wdAllowOnlyRevisions
    End If
End Sub
```

#### HINWEIS

Es können nur Public Sub-Prozeduren, die keine Argumente verwenden, einer Symbolschaltfläche zugewiesen werden. Falls eine Schaltfläche einen Wert dieser Prozedur liefern soll, kann er in der Parameter-Eigenschaft gespeichert werden.

**Action-Control** Wollen Sie feststellen, welche Symbolschaltfläche betätigt wurde, können Sie die *ActionControl*-Eigenschaft der *CommandBars*-Auflistung benutzen. In Listing 15.4 werden mehrere Symbolschaltflächen einer Symbolleiste hinzugefügt. Als Parameter wird ein Farbname festgelegt. Alle diese Steuerelement-Objekte werden an die Prozedur *HervorhebungsSchaltflächen* übergeben, die die übrigen Eigenschaften festlegt. Unter anderem wird der *OnAction*-Eigenschaft aller Objekte das gleiche Makro *MarkierungHervorheben* zugewiesen. Klickt der Anwender auf eine dieser vier Symbolschaltflächen (in Abbildung 15.4 abgebildet), wird diese durch *ActionControl* ermittelt. Je nach Parameter (dem Farbennamen), wird der Wert für die Hervorhebungsfarbe festgelegt.

**Listing 15.4** Mehrere Schaltflächen erstellen, die mit dem gleichen Makro verbunden sind. Die *OnAction*-Eigenschaft entscheidet, welche Handlung auszuführen ist.

```
Sub SymbolschaltflächeDefinieren2()
    Dim ctl As Office.CommandBarButton

    Application.CustomizationContext = ActiveDocument
    Set ctl = CommandBars("Meine Befehle").Controls.Add(Type:=msoControlButton, _
        Parameter:="Gelb")
```

**Listing 15.4** Mehrere Schaltflächen erstellen, die mit dem gleichen Makro verbunden sind. Die *OnAction*-Eigenschaft entscheidet, welche Handlung auszuführen ist. (Fortsetzung)

```
HervorhebungsSchaltflächen ctl
Set ctl = CommandBars("Meine Befehle").Controls.Add(Type:=msoControlButton, _
    Parameter:="Grün")
HervorhebungsSchaltflächen ctl
Set ctl = CommandBars("Meine Befehle").Controls.Add(Type:=msoControlButton, _
    Parameter:="Rosa")
HervorhebungsSchaltflächen ctl
Set ctl = CommandBars("Meine Befehle").Controls.Add(Type:=msoControlButton, _
    Parameter:="Keine")
HervorhebungsSchaltflächen ctl
End Sub

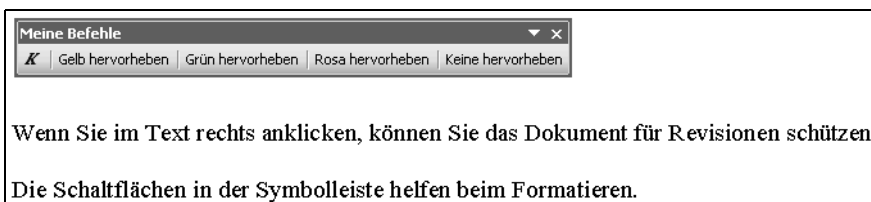
Sub HervorhebungsSchaltflächen(ctl As Office.CommandBarButton)
    Dim sFarbe As String

    sFarbe = ctl.Parameter
    ctl.Caption = sFarbe & " hervorheben"
    ctl.OnAction = "MarkierungHervorheben"
    ctl.DescriptionText = "Hervorhebung der gegenwärtigen Markierung " & sFarbe
    ctl.Style = msoButtonCaption
    ctl.TooltipText = "Markierung " & sFarbe & " hervorheben"
    ctl.BeginGroup = True
End Sub

Sub MarkierungHervorheben()
    Dim ctl As Office.CommandBarButton
    Dim lFarbenWert As Long

    Set ctl = Application.CommandBars.ActionControl
    Select Case ctl.Parameter
        Case "Gelb"
            lFarbenWert = wdYellow
        Case "Grün"
            lFarbenWert = wdBrightGreen
        Case "Rosa"
            lFarbenWert = wdPink
        Case "Keine"
            lFarbenWert = wdWhite
        Case Else
            lFarbenWert = wdRosa
    End Select
    Selection.Range.HighlightColorIndex = lFarbenWert
End Sub
```

**Abbildg. 15.4** Schaltflächen, die über die Automatisierungsschnittstelle erstellt wurden





Die Beispieldatei *Bsp15\_03.doc* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap15*.

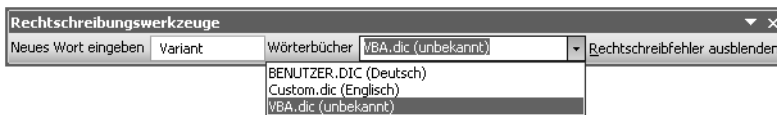
## Bearbeitungsfelder und Dropdownlisten

Obwohl der Word-Umgebung vier Befehlsleisten-Steuerelementtypen zur Verfügung stehen, können über die Benutzerschnittstelle nur zwei davon definiert werden: Schaltflächen (`msoControlButton`) sowie Popup-Menüs (`msoControlPopup`). Um Bearbeitungsfelder oder Dropdownlisten einer Symbolleiste hinzuzufügen, braucht es ein Makro.

Diese zwei Control-Typen sind in Abbildung 15.5 ersichtlich. Deren Style-Eigenschaft wurde jeweils auf `msoComboLabel` festgelegt (ein Bearbeitungsfeld ist eine Dropdownliste ohne die Liste). Das `OnAction`-Makro eines Bearbeitungsfelds wird durch Drücken der -Taste ausgeführt; das einer Dropdownliste durch die Auswahl eines Eintrags.

In diesem Beispiel wird das vom Anwender in das Bearbeitungsfeld eingegebene Wort dem aktuellen Benutzerwörterbuch hinzugefügt. Das aktuelle Wörterbuch kann aus der Liste der geladenen Wörterbücher in der Dropdownliste festgelegt werden (das aktive Wörterbuch erscheint im Textfeld der Dropdownliste).

**Abbildg. 15.5** Bearbeitungsfelder und Dropdownlisten, wie hier abgebildet, können nur programmtechnisch erstellt werden.



Der Code, um diese beiden Symbolschaltflächen zu erstellen, befindet sich in Listing 15.5. Elemente werden einer Dropdownliste durch die `AddItem`-Methode hinzugefügt.

**Listing 15.5** Edit- und ComboBox-Schaltflächen einer Symbolleiste hinzufügen

```
Sub EditSchaltflächeErstellen()
    Dim ctl As Office.CommandBarControl

    Set ctl = Application.CommandBars("
        RechtschreibWerkzeuge").Controls.Add(msoControlEdit)
    ctl.Style = msoComboLabel
    ctl.Caption = "Neues Wort eingeben"
    'Wird beim Drücken der Eingabe-Taste ausgelöst
    ctl.OnAction = "WortHinzufügen"
End Sub

Sub WörterbuchListeErstellen()
    Dim cbo As Office.CommandBarComboBox
    Dim dic As Word.Dictionary
    Dim lZaehler As Long

    Application.CustomizationContext = ActiveDocument
    Set cbo = Application.CommandBars( _
```



Listing 15.5 Edit- und ComboBox-Schaltflächen einer Symbolleiste hinzufügen (Fortsetzung)

```

"RechtschreibungsWerkzeuge").Controls.Add(Type:=msoControlComboBox)
cbo.Style = msoComboLabel
cbo.Caption = "Wörterbücher"
cbo.Width = 250
cbo.OnAction = "AktivesWörterbuchAendern"
lZaehler = 1
For Each dic In Application.CustomDictionaries
    cbo.AddItem dic.Name & Sprache(dic.LanguageID)
    If Application.CustomDictionaries.ActiveCustomDictionary.Name _
        = dic.Name Then
        cbo.ListIndex = lZaehler
    End If
    lZaehler = lZaehler + 1
Next
End Sub

```



Die Beispieldatei *Bsp15\_04.doc* mit der gesamten Lösung finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap15*.

## Symbolleisten erstellen

Neue Symbolleisten werden mit der Add-Methode der CommandBars-Auflistung erstellt:

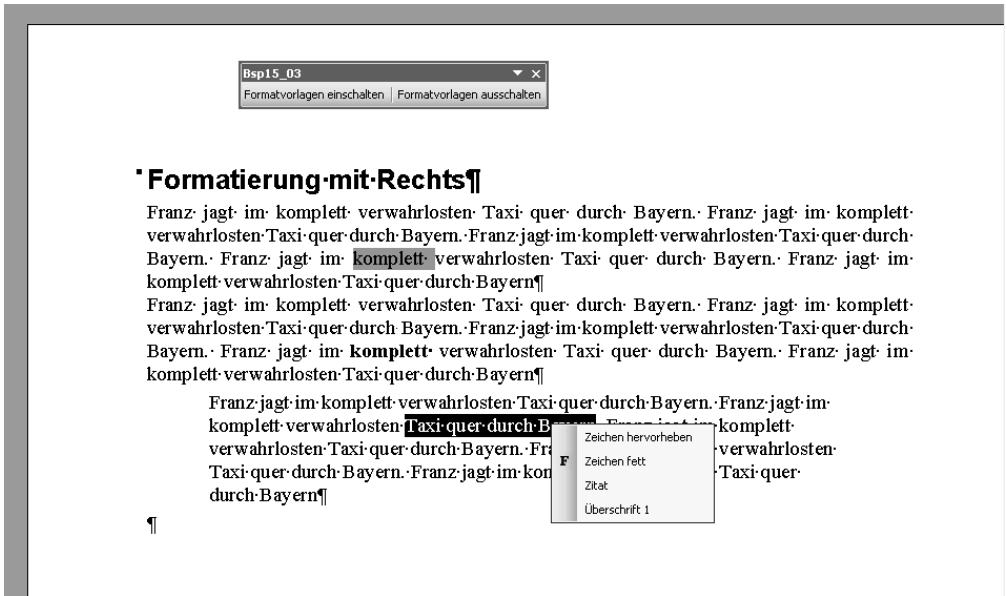
```
Add(Name, Position, MenuBar, Temporary)
```

Alle Argumente sind optional. Die Werte für Position wurden bereits im Abschnitt »CommandBars« am Anfang dieses Kapitels vorgestellt. Soll die neue Symbolleiste die Menüleiste des gegenwärtigen Kontextes ersetzen, muss MenuBar auf True festgelegt werden. Für Temporary bedeutet True, dass die Symbolleiste beim Schließen des Kontextes verworfen wird.

Interessant als Beispiel, das das Erstellen einer Symbolleiste samt Steuerelemente veranschaulicht, ist das Erstellen eines Kontextmenüs, das eines der Word-eigenen ersetzt. Die Abbildung 15.6 zeigt dazu ein einfaches Beispiel. Um Text in einem Dokument schnell zu formatieren, werden die benötigten Formatvorlagen in einem Kontextmenü aufgelistet. Dadurch muss die Maus nicht bis zu einer Symbolleiste, die entweder weit entfernt liegt oder deren Sicht darauf verdeckt ist, bewegt werden, sondern kann unmittelbar neben der Markierung den gewünschten Befehl auswählen.

Mit den Symbolschaltflächen in der »normalen« Symbolleiste (oben in Abbildung 15.6) wird die Aktivierung dieses Menü ein- und ausgeschaltet (das Word-eigene Kontextmenü wird wieder aktiv).

Das Kontextmenü wurde mit der Prozedur KontextMenuErstellen in Listing 15.6 erstellt. Bitte beachten Sie dabei den Control-Typ msoControlPopup. Stellvertretend für die Erstellung der einzelnen Schaltflächen und ihre OnAction-Makros, die alle nach dem gleichen Muster ablaufen, befinden sich MenuEintrag1Erstellen bzw. MenuEintrag1 im aufgeführten Code.

**Abbildg. 15.6** Ein Dokument mit einem eigens dafür erstellten Kontextmenü effizient formatieren

**Listing 15.6** Diese Prozedur einmal im erwünschten Speicherort durchführen, um das Kontextmenü zu erstellen

```

Sub KontextMenuErstellen()
    Dim cb As Office.CommandBar

    Application.CustomizationContext = ActiveDocument
    Set cb = Application.CommandBars.Add(Name="Kontext", Position:=msoBarPopup)
    MenuEintrag1Erstellen cb
    MenuEintrag2Erstellen cb
    MenuEintrag3Erstellen cb
    MenuEintrag4Erstellen cb
End Sub

Sub MenuEintrag1Erstellen(cb As Office.CommandBar)
    Dim ctl As Office.CommandBarButton

    Set ctl = cb.Controls.Add(Office.msoControlButton)
    ctl.Caption = "Zeichen hervorheben"
    ctl.Enabled = True
    ctl.OnAction = "MenuEintrag1"
    ctl.Style = msoButtonCaption
    ctl.Visible = True
End Sub

Public Sub MenuEintrag1()
    Dim rng As Word.Range
    Set rng = Application.Selection.Range
    rng.Style = "Zeichen hervorheben"
    rng.Shading.ForegroundPatternColor = wdColorSkyBlue
End Sub

```

Ein Popup-Menü kann nur programmtechnisch eingeblendet werden; die Eigenschaft `Visible` verursacht einen Laufzeitfehler. Dieses Beispiel bedient sich des Ereignisses `WindowBeforeRightClick`. Das Ereignis wird mit der Prozedur `EreignisseEinschalten` in Listing 15.7 aktiviert und mit `EreignisseAusschalten` wieder außer Kraft gesetzt. Die beiden sind den Schaltflächen der »normalen« Symbolleiste zugewiesen, so dass der Benutzer das Kontextmenü nach Bedarf benutzen kann.

**HINWEIS**

Einzelheiten über den Einsatz von Ereignissen sind in Kapitel 7 beschrieben.

Die Ereignis-Prozedur `app_WindowBeforeRightClick` ruft `ShowKontext` auf, die das Kontextmenü einblendet. Die standardmäßige Handlung des Rechtsklicks wird unterdrückt (`Cancel=True`).

**Listing 15.7** Die Teile des Ereignisses, das das Rechtsklick-Verhalten steuert

```
'Inhalt des normalen Moduls
Public appEvents As New clsBsp15_03

Public Sub EreignisseEinschalten()
    Set appEvents.app = Word.Application
End Sub

Public Sub EreignisseAusschalten()
    Set appEvents.app = Nothing
End Sub

Public Sub ShowKontext(cb As Office.CommandBar)
    cb.ShowPopup
End Sub

'Inhalt des Klassenmoduls
Public WithEvents app As Word.Application
Private Sub app_WindowBeforeRightClick(ByVal sel As Selection, Cancel As Boolean)
    Dim cb As Office.CommandBar

    app.CustomizationContext = ActiveDocument
    Set cb = app.CommandBars("Kontext")
    ShowKontext cb
    Cancel = True
End Sub
```

**HINWEIS**

Müssen viele komplexe Symbolleisten erstellt und entfernt werden, ist es mühsam, alle Einzelheiten im Code aufzuschreiben und zu verwalten. Für solche Aufgaben lohnt es sich, die Einzelheiten des Aufbaus einer Symbolleiste mit allen Eigenschaften in einer externen Datei festzuhalten. Diese könnte beispielsweise eine Excel-Tabelle, eine Access-Datenbank oder eine XML-Datei sein. Der Code öffnet die Datei, arbeitet diese ab, und erstellt nach deren Angaben die Symbolleisten.



Die Beispieldatei *Bsp15\_05.doc* mit der gesamten Lösung finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap15*.

# Zusammenfassung

Dieses Kapitel beschäftigte sich mit der programmtechnischen Erstellung von Symbolleisten. Beschrieben wurde dabei,

- wie mit den Eigenschaften einer Symbolleiste umzugehen ist (Seite 650),
- wie sich verschiedene Arten von Bedienfeldern für eine Symbolleiste erstellen und mit einem Befehl verbinden lassen (Seite 656),
- wie Symbolleisten erstellt werden (Seite 661).

## Kapitel 16

# Tastaturbelegungen

### In diesem Kapitel:

Tastenbelegungen im Objektmodell	667
Tastaturkombinationen verwalten	675
Tastenbelegung mit COM-Add-In verbinden	677
Zusammenfassung	681

Da Word eine Textverarbeitung ist, arbeiten viele seiner Anwender intensiv mit der Tastatur. Für geübte Finger lässt sich schneller arbeiten, wenn sie möglichst wenig zur Maus greifen müssen. Es ist daher wenig überraschend, dass Word mit einer großen Anzahl vordefinierter Tastenkombinationen geliefert wird. Sie finden ausführliche Listen in der Hilfe, unter dem Begriff »Tastenkombination«. Diese sind jedoch nicht im Stein gemeißelt. So ziemlich alle Tastenbelegungen können angepasst werden. Damit kann der Entwickler seine Werkzeuge, oder der Ersteller einer Dokumentvorlage wichtige Formatierungsbefehle und AutoTexte zugänglich machen.

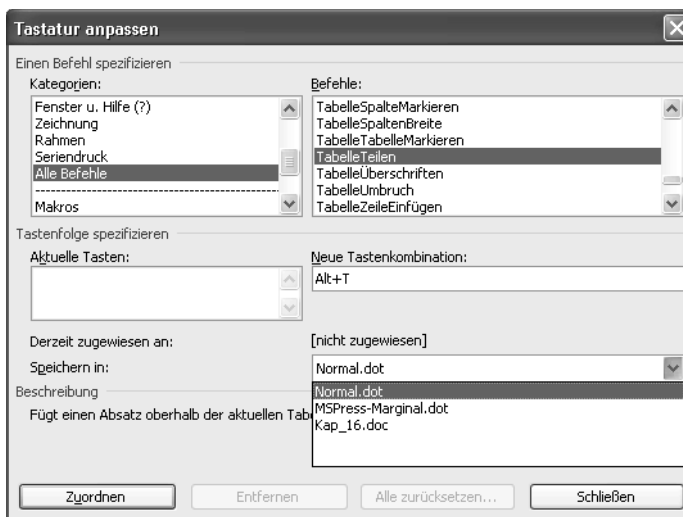
In der Benutzerschnittstelle geschieht dies, indem Sie den Menübefehl *Extras/Anpassen* aufrufen und dort die Schaltfläche *Tastatur* anklicken; das Dialogfeld ist in Abbildung 16.1 ersichtlich. Wie in den übrigen *Anpassen*-Dialogfeldern wird eine Kategorie ausgewählt, dann der Befehl, wofür eine Tastenkombination zu definieren ist. In der Liste *Aktuelle Tasten* erscheinen vorhandene Kombinationen für diesen Befehl. Die vorgeschlagene Kombination wird in das Feld *Neue Tastenkombination* eingegeben. Wurde sie noch nicht belegt, erscheint »[nicht zugewiesen]« darunter, sonst der Befehlsname, dem diese Kombination schon zugewiesen ist. Durch Betätigung der Schaltfläche *Zuordnen* wird die Belegung bestätigt, oder eine andere Kombination kann eingegeben werden.

Auch hier ist es wichtig, den *Kontext* festzulegen. Wurden in mehreren Kontexten die gleiche Tastenkombination verschiedenen Befehlen zugeordnet, gelten die in Kapitel 13 festgelegten Richtlinien.

#### TIPP

Um eine Liste der vom Benutzer definierten Tastaturkürzel zu sehen, wählen Sie aus der Liste *Drucken* im Dialogfeld *Datei/Drucken* den Eintrag *Tastenbelegung*, dann klicken Sie auf *OK*.

**Abbildg. 16.1** Fast jedem Befehl, Makro, Formatvorlage, Schriftart oder AutoTextEintrag kann eine Tastenkombination zugewiesen werden.



#### HINWEIS


Bekanntlich können Makros, Formatvorlagen, AutoText-Einträge und Symbolleisten über den Menübefehl *Extras/Vorlagen und Add-Ins* und Anklicken der Schaltfläche *Organisieren* im zugehörigen Dialogfeld zwischen Word-Dokumenten kopiert werden. Es gibt jedoch kein Word-eigenes Werkzeug, mit dem sich auch Tastaturkürzel übertragen ließen. Unter <http://>





[www.chriswoodman.co.uk/Shortcut%20Organizer.htm](http://www.chriswoodman.co.uk/Shortcut%20Organizer.htm) kann kostenlos ein Werkzeug namens »Shortcut Organizer« heruntergeladen werden, das dies für Sie erledigen kann. Davon gibt es auch eine deutsche Version (suchen Sie die Versionen mit dem Text »German legends by Cindy Meister«).

# Tastenbelegungen im Objektmodell

**Key-Bindings** Im Objektmodell werden Tastenbelegungen durch die KeyBindings-Auflistung, eine Eigenschaft des Application-Objekts, verwaltet. Die Anzahl der im angegebenen Kontext sich befindlichen Tastenkombinationen wird mit der Count-Eigenschaft ermittelt.

Fünf Parameter bzw. Eigenschaften definieren ein KeyBinding-Objekt:

**KeyCode** und **KeyCode2** umfassen alle Tasten der Tastaturbelegung. Das Word-Objektmodell stellt rund einhundert vordefinierte WdKey-Konstantwerte zur Verfügung, so dass der Entwickler die Ganzzahl für jede Taste nicht auswendig lernen muss. Um beispielsweise auf die -Taste zu verweisen, wird der Konstantwert wdKeyTab benutzt. Weitere, im Word-Objektmodell fehlende Konstantwerte befinden sich im allgemeinen Visual Basic-Objektmodell, in der vbKey-Enumeration. Hier finden Sie beispielsweise Werte für die Pfeiltasten, wie vbKeyLeft und vbKeyRight.

KeyCode2 beträgt 255 (wdKeyNoKey), es sei denn, eine zweite Taste folgt der ersten Tastenkombination, wie beispielsweise  +  + , , wobei 82 (wdKeyR), der AscW-Code für »R«, der Wert von KeyCode2 wäre.

**Key-Category** Die Art von Tastaturbelegung wird durch die Eigenschaft **KeyCategory** festgelegt. Diese könnte beispielsweise ein Word-interner Befehl, ein Makro, eine Formatvorlage o.ä. sein. Eine Liste der WdKeyCategory-Werte mit Beispielen finden Sie in der Tabelle 16.1.

**Command** **Command** spezifiziert, was zu machen ist, beispielsweise der Befehl-, Makro- oder Formatvorlagenname (siehe auch Tabelle 16.1). Handelt es sich um einen Word-Befehl, wird der englische Befehlsname zurückgegeben, bzw. bei der Definition muss der englische Befehlsname angegeben werden.

## HINWEIS

Auf der CD befindet sich im Ordner \Beilagen\Interne Word-Befehle die Datei *Interne WordBefehle.doc*, die eine Tabelle enthält mit allen englischen und deutschen Befehlsnamen, gegenüber gestellt.

**Command-Parameter** Falls ein interner Word-Befehl einen Parameter verlangt, wird dies über **CommandParameter** festgelegt. Beispiele solcher Befehle sind *Farbe*, *Spalten*, *DateiDateiÖffnen*, *Rahmen*, *Schattierung*, *Schmal*, *Erweitert*, *Schriftgröße*, *Tiefgestellt* und *Erhöht* (siehe auch Tabelle 16.1.) Benutzerdefinierte Prozeduren, die Parameter verlangen, stehen Tastenbelegungen nicht zur Verfügung.

Zudem befindet sich eine Spalte »WordBasic.[KeyMacro\$]()« in Tabelle 16.1. Diese Funktion liefert für einige KeyCategory nützliche Informationen, die über die VBA-Schnittstelle nicht zur Verfügung gestellt werden, wie etwa der deutsche Befehlsname eines Word-internen Befehls. Diese Funktion ist nur für benutzerdefinierte Tastenbelegungen anwendbar und verwendet zwei Parameter:

- Index: der numerische Index-Wert des *benutzerdefinierten* KeyBinding
- Context: 0 für das aktuelle Dokument; 1 für seine Vorlage

**Tabelle 16.1** Beispiele für die Eigenschaften *KeyCategory*, *Command* und *CommandParameter*

<b>KeyCategory</b>	<b>Command</b>	<b>Command-Parameter</b>	<b>WordBasic. [KeyMacro\$]()</b>
<b>wdKeyCategoryDisable</b> (gesperrt) 0	[Leer]	[Leer]	""
<b>wdKeyCategoryCommand</b> (Befehl) 1	"ParaKeepWithNext"	[Leer]	"AbsätzeNichtTrennen"
<b>wdKeyCategoryCommand</b> (erweiterter Befehl) 1	"Farbe"	"12"	"Violet"
<b>wdKeyCategoryCommand</b> (erweiterter Befehl) 1	"DateiDateiÖffnen"	"C:\My Documents\test.doc"	"Open test.doc"
<b>wdKeyCategoryCommand</b> (Zeichen in der angegebenen Symbol-Schriftart) 1	"Symbol"	"ZapfDingbats BT" ?	"ZapfDingbats BT: 61512"
<b>wdKeyCategoryMacro</b> (Makro) 2	"Normal.NewMacros.Makro1"	[Leer]	"Normal.NewMacros.Makro1"
<b>wdKeyCategoryFont</b> (Schriftart) 3	"Arial Narrow"	[Leer]	"Arial Narrow"
<b>wdKeyCategoryAutotext</b> (AutoText) 4	"Seite X von Y"	[Leer]	"Seite X von Y"
<b>wdKeyCategoryStyle</b> (Formatvorlage) 5	"Umschlagadresse"	[Leer]	"Umschlagadresse Formatvorlage"
<b>wdKeyCategorySymbol</b> (Symbol in normal Text) 6	"¼"	[Leer]	""
<b>wdKeyCategoryPrefix</b> 7	<p>Diese Konstante wird bei der Erstellung einer Tastaturbelegung nicht gebraucht, sondern nur bei der Ermittlung bereits belegter Tastenkombinationen.</p> <p>Haben Sie eine Belegung wie [Alt] + [A], [B], die beispielsweise ein Makro ausführt, ist der erste Teil [Alt] + [A] das »Präfix«. Die folgende Kommandozeile gibt den Wert »2« zurück, da das ganze KeyBinding ein Makro ausführt.</p> <p><b>MsgBox KeyBindings(1).KeyCategory</b></p> <p>Die folgende Kommandozeile gibt jedoch 7 zurück, da die Kombination [Alt] + [A] die erste Hälfte einer zweiteiligen Belegung ist.</p> <p><b>MsgBox Application.FindKey(wdKeyAlt+wdKeyA).KeyCategory</b></p>		



Tabelle 16.1 Beispiele für die Eigenschaften *KeyCategory*, *Command* und *CommandParameter* (Fortsetzung)

KeyCategory	Command	Command-Parameter	WordBasic. [KeyMacro\$]()
wdKeyCategoryNil -1	<p>Diese <i>KeyCategory</i>-Eigenschaft gibt <b>wdKeyCategoryNil</b> (-1) zurück, wenn die Tastenkombination im fraglichen Kontext nicht zugewiesen wurde. Beispiel:</p> <p>Die Kombination <span>⌘</span> + <span>⇧</span> + <span>X</span> ist nicht belegt. Folgendes Makro schreibt den Wert -1 in das Direktfenster:</p> <pre>Sub KeyCategoryNilDemo()     Dim o_Key As Word.KeyBinding     Set o_Key = Application.FindKey(wdKeyCtrl + _         wdKeyShift + wdKeyX)     Debug.Print o_Key.KeyCategory End Sub</pre> <p>Wenn VBA eine solche KeyBinding ausführt (<b>o_Key.Execute</b>), soll, analog wie in der Benutzerschnittstelle, nichts passieren, keine Fehlermeldung und keine Aktion erfolgt.</p>		

SATZ: Bitte den folgenden Text als "Fußnote" zur Tabelle formatieren!

Zum »?« in der Spalte »CommandParameter« der Wenn das erste Zeichen der **CommandParameter**-Eigenschaft eines »Symbol«-**Command** in der angegebenen Schriftart im VBA-Editor nicht wiedergeben werden kann, weil es sich um ein 2-Byte-Unicode-Zeichen handelt, das mit der Funktion **AscW** ermittelt wurde, gibt VBA ein Fragezeichen zurück. Das Gleiche gilt für das einzige Zeichen der **Command**-Eigenschaft von **KeyCategory** **wdKeyCategorySymbol**.

BuildKey  
Code

Ein spezifisches KeyBinding wird mit einem KeyCode identifiziert. Ein KeyCode besteht aus der Summe seiner Tastenwerte und wird meistens mit der **BuildKeyCode**-Eigenschaft zusammengestellt. Diese akzeptiert bis zu vier Argumente; beispielsweise **BuildKeyCode(wdKeyShift, wdKeyControl, wdKeyF)** ist gleich **[⇧] + [Strg] + [F]**.

Bestehende Tastenbelegungen ermitteln

FindKey

Das Objektmodell unterscheidet zwischen benutzerdefinierten und Word-internen Tastenbelegungen. Die **FindKey**-Eigenschaft des **Application**-Objekts erkennt beide Arten:

```
Dim kb1 as Word.KeyBinding  
Dim kb2 as Word.KeyBinding  
Set kb1 = Application.FindKey(BuildKeyCode(wdKeyF1))  
Set kb2 = Application.FindKey(BuildKeyCode(wdKeyAlt, wdKeyT))  
MsgBox kb1.Command  
'Ergebnis: »Help«, falls F1 noch der Hilfe-Befehl zugewiesen ist  
MsgBox kb2.Command  
'Ergebnis: »Beispiel«, falls Alt+T einem AutoText dieses Namens zugewiesen ist
```

Key

Die **Key**- sowie **KeyBinding**-Eigenschaften der **KeyBindings**-Auflistung hingegen erkennen nur benutzerdefinierte Tastenkombinationen:

```
MsgBox Application.KeyBindings.Key(BuildKeyCode(wdKeyF1)).Command  
'Ergebnis: eine leere Zeichenkette, falls F1 noch der Hilfe-Befehl zugewiesen ist  
MsgBox Application.KeyBindings.Key(BuildKeyCode(wdKeyAlt, wdKeyT)).Command  
'Ergebnis: »Beispiel«, falls Alt+T einem AutoText dieses Namens zugewiesen ist
```

Tastaturkürzel können auch mit zwei aufeinander folgenden Tastenfolgen definiert werden. In Word sind für internationale Zeichen einige Tastenkombinationen vordefiniert, wie `Strg+„`, `U`, mit der das Zeichen »ú« eingefügt wird. Dann haben die Eigenschaften wie `FindKey` folgende Syntax:

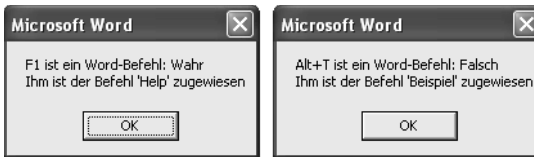
```
Set kb2 = Application.FindKey(BuildKeyCode(wdKeyControl, wdKeySingleQuote), wdKeyU)
```

Es gibt keine Eigenschaft, die ausschließlich Word-Befehle anspricht oder identifiziert. Dies ist jedoch über einen Umweg möglich, wie die Prozedur `IstWordBefehl` in Listing 16.1 veranschaulicht. Wird `Key` oder `KeyBinding` der `KeyCode` einer nicht benutzerdefinierten Tastenkombination übergeben, führt dies zu einem Laufzeitfehler. Mit `On Error Resume Next` kann dieser ignoriert werden. Dann wird geprüft, ob ein `KeyBinding` der Objektvariablen zugewiesen werden konnte. Wenn nicht, handelt es sich um einen Word-Befehl.

Key-  
String

Das Resultat der Prozedur `TestWordBefehl` in Listing 16.1 ist in Abbildung 16.2 ersichtlich. Die Zeichenkette mit der Tastenkombination wurde durch die Eigenschaft `KeyString` des `KeyBinding`-Objekts erzeugt.

**Abbildg. 16.2** Zwischen einer benutzerdefinierten Tastenbelegung und einem Word-Befehl unterscheiden



**Listing 16.1** Feststellen, ob eine Tastenbelegung mit einem benutzerdefinierten oder Word-internen Befehl verbunden ist

```
Sub TestWordBefehl()
    Dim lKeyCode1 As Long, lKeyCode2 As Long
    Dim kb1 As Word.KeyBinding, kb2 As Word.KeyBinding

    Application.CustomizationContext = ActiveDocument.AttachedTemplate
    lKeyCode1 = BuildKeyCode(wdKeyF1)
    Set kb1 = Application.FindKey(lKeyCode1)
    MsgBox kb1.KeyString & " ist ein Word-Befehl: " & _
        IstWordBefehl(lKeyCode1) & vbCrLf & "Ihm ist der Befehl '" & _
        & kb1.Command & "' zugewiesen"

    lKeyCode2 = BuildKeyCode(wdKeyAlt, wdKeyT)
    Set kb2 = Application.FindKey(lKeyCode2)
    MsgBox kb2.KeyString & "ist ein Word-Befehl: " & _
        IstWordBefehl(lKeyCode2) & vbCrLf & "Ihm ist der Befehl '" & _
        & kb2.Command & "' zugewiesen"
End Sub

Function IstWordBefehl(lKeyCode As Long) As Boolean
    Dim kb As Word.KeyBinding

    On Error Resume Next
    Set kb = Application.KeyBindings.Key(lKeyCode)
    On Error GoTo 0
    If kb Is Nothing Then
```

**Listing 16.1** Feststellen, ob eine Tastenbelegung mit einem benutzerdefinierten oder Word-internen Befehl verbunden ist (*Fortsetzung*)

```
IstWordBefehl = True
Else
    IstWordBefehl = False
End If
End Function
```

#### PROFITIPP

Um schnell herauszufinden, welche Tastenbelegung (wenn überhaupt) einem Befehl im gegenwärtigen Kontext zugewiesen wurde, drücken Sie **[Strg] + [Alt] + [Num+]**. Der Mauszeiger ändert sich in ein Kleeblatt **☛**. Klicken Sie damit auf den Befehl, der Sie interessiert. Wenn ihm eine Tastenkombination zugewiesen wurde, erscheint das Dialogfeld *Tastatur anpassen* mit der Belegung.



Die Beispieldatei *Bsp16\_01.dot* finden Sie auf der CD-ROM zum Buch im Ordner `\Beispiele\Kap16`.

Das Listing 16.2 enthält Ausschnitte einer Lösung, die alle Tastenbelegungen eines Kontextes in einer Tabelle auflistet. Der Code veranschaulicht die Verwendung der bisher vorgestellten Eigenschaften. Ein Teil des Resultats ist in Abbildung 16.3 ersichtlich. Vergleichen Sie die Spalte »Kat.« mit den Angaben in Spalte »KeyCategory« der Tabelle 16.1.

#### HINWEIS

Bitte beachten Sie, wie der Ausdruck »Shift« bei einigen Kombinationen mit der **[⇧]**-Taste fehlt. Stattdessen wird das Zeichen angezeigt, das als Ergebnis der Kombination **[⇧] + [Taste]** entsteht. **[Alt] + [%]** beispielsweise statt **[⇧] + [Alt] + [5]**.

**Abbildg. 16.3** Word-interne sowie benutzerdefinierte Tastenbelegungen auflisten

Tastaturk.	Kat.	Befehl	allfälliger Parameter	Word-Befehl
Alt+A	3	Book Antiqua		Falsch
Alt+F	5	FarbigerText		Falsch
Alt+M	2	ptjBsp16_01.Bsp16_01.TestMakro		Falsch
Alt+N	1	RemoveBulletsNumbers		Falsch
Alt+P	6	¶		Falsch
Alt+S, P	1	ToolsCustomize		Falsch
Alt+T	4	Beispiel		Falsch
Abwärts	1	LineDown		Wahr
Alt+''	1	ShowHeading2		Wahr
Alt+%	1	ShowHeading5		Wahr

Die Prozedur `AlleTastenBelegungenAuflisten` schleift zuerst durch alle `wdKey`-Konstantenwerte, bis auf die Befehlstasten **[⇧]**, **[Strg]**, und **[Alt]**. Es wird geprüft, ob diese einem Befehl zugewiesen sind (`KeyCategory <> -1`). Wenn ja, werden in der Funktion `BefehlslisteZusammenstellen1` die Angaben für die Tabelle in einer zeichengetrennten Liste zusammengetragen.

Die gleichen Handlungen werden für alle Tasten in Kombination mit den Befehlstasten, einzeln sowie in Kombination ausgeführt. Für jede mögliche Anzahl Argumente stehen Funktionsvariationen für `BefehlslisteZusammenstellen#` bereit (diejenigen für zwei Argumente sind ebenfalls in Listing 16.2 enthalten).

Am Schluss wird die Zeichenkette `sBefehlsliste` in das aktuelle Dokument eingefügt und in eine Tabelle umwandelt. Diese wird nach den Spalten »Word-Befehl«, dann »Tastaturk.« sortiert (die benutzerdefinierte Tastenbelegungen stehen damit am Tabellenanfang).

**Listing 16.2** Alle Tastenbelegungen eines Kontextes auflisten

```
Sub AlleTastenBelegungenAuflisten()
    Dim lKeyArg1 As Long, lKeyArg2 As Long, lKeyArg3 As Long
    Dim sBefehl As String
    Dim sBefehlsListe As String
    Dim aArg2() As Variant
    Dim lCounter As Long
    Dim lKeyArg As Long
    Dim index As Long
    Dim rng As Word.Range
    Dim tbl As Word.Table

    Application.CustomizationContext = ActiveDocument
    'Sanduhr anzeigen
    System.Cursor = wdCursorWait
    aArg2() = Array(wdKeyAlt, wdKeyControl, wdKeyShift)
    sBefehlsListe = "Tastaturk." & vbTab & "Kat." & vbTab & "Befehl" _
        & vbTab & "allfälliger Parameter" & vbTab & "Word-Befehl"

    'Alle "normalen" Tasten durchschleifen
    For lKeyArg1 = 1 To 254
        'Wenn die Tastenkombination einem Befehl zugewiesen ist
        'die Informationen zusammenstellen
        If Application.FindKey(BuildKeyCode(lKeyArg1)).KeyCategory <> -1 Then
            'Anzahl der Befehle aufzählen
            index = index + 1
            sBefehlsListe = sBefehlsListe & vbCr & BefehlslisteZusammenstellen1(lKeyArg1)
        End If
        'Alle "normalen" Tasten mit Befehlstasten kombiniert prüfen
        For lCounter = LBound(aArg2) To UBound(aArg2)
            lKeyArg2 = aArg2(lCounter)
            If Application.FindKey(BuildKeyCode(lKeyArg1, lKeyArg2), wdNoKey).KeyCategory _
                <> -1 Then
                index = index + 1
                sBefehlsListe = sBefehlsListe & vbCr & _
                    BefehlslisteZusammenstellen2(lKeyArg1, lKeyArg2, wdNoKey)
            End If
        End If
        'Dann mit allen "normalen" Tasten als KeyCode2-Erweiterung prüfen
        For lKeyArg = 1 To 254
            If Application.FindKey(BuildKeyCode(lKeyArg1, lKeyArg2), lKeyArg).KeyCategory _
                <> -1 Then
                index = index + 1
                sBefehlsListe = sBefehlsListe & vbCr & _
                    BefehlslisteZusammenstellen2(lKeyArg1, lKeyArg2, lKeyArg)
            End If
        Next lKeyArg
    Next lKeyArg1
End Sub
```

Listing 16.2 Alle Tastenbelegungen eines Kontextes auflisten (Fortsetzung)

```

'Code, um Tastenbelegungen mit zwei und mehr Befehlstasten zu prüfen,
'folgen an dieser Stelle in der vollständiger Prozedur auf der CD
    Next lCounter
Next lKeyArg1

'Liste in die Markierungsstelle eingeben und...
Set rng = Selection.Range
rng.Text = sBefehlsListe
'daraus eine sortierte Tabelle erstellen
Set tbl = rng.ConvertToTable(Separator:=vbTab, NumColumns:=5)
tbl.Rows(1).Range.Font.Bold = True
tbl.Sort FieldNumber:=5, FieldNumber2:=1

Debug.Print index & " Tastaturbelegungen"
System.Cursor = wdCursorNormal
End Sub

Function BefehlslisteZusammenstellen1(ByVal lArg1 As Long) As String
    Dim bWordBefehl As Boolean
    Dim kb As Word.KeyBinding
    Dim s As String

    On Error Resume Next
    Set kb = Application.KeyBindings.Key(BuildKeyCode(lArg1))
    On Error GoTo 0
    If kb Is Nothing Then
        bWordBefehl = True
        Set kb = Application.FindKey(lArg1)
    Else
        bWordBefehl = False
    End If

    s = Beschreibung(kb, bWordBefehl)
    BefehlslisteZusammenstellen1 = s
End Function

Function BefehlslisteZusammenstellen2(ByVal lArg1 As Long, ByVal lArg2 As Long, _
    ByVal lArg As Long) As String
    Dim bWordBefehl As Boolean
    Dim kb As Word.KeyBinding
    Dim s As String
    Dim WBAusdruck As String

    On Error Resume Next
    Set kb = Application.KeyBindings.Key(BuildKeyCode(lArg1, lArg2), lArg)
    On Error GoTo 0
    If kb Is Nothing Then
        bWordBefehl = True
        Set kb = Application.FindKey(BuildKeyCode(lArg1, lArg2), lArg)
    Else
        bWordBefehl = False
    End If

    s = Beschreibung(kb, bWordBefehl)
    BefehlslisteZusammenstellen2 = s

```

**Listing 16.2** Alle Tastenbelegungen eines Kontextes auflisten (*Fortsetzung*)

```
End Function

'Die Funktionen BefehlslisteZusammenstellen3 und BefehlslisteZusammenstellen4,
'die mehr lArg-Argumente nehmen, folgen an dieser Stelle

Function Beschreibung(kb As Word.KeyBinding, bWordBefehl) As String
    Dim s As String
    Dim sBefehl As String
    Dim WBAusdruck As String

    sBefehl = kb.Command
    If sBefehl = vbTab Then sBefehl = "TAB"
    s = Replace(kb.KeyString, ",", ", ") & vbTab & kb.KeyCategory & vbTab & sBefehl _
        & vbTab & Replace(kb.CommandParameter, vbTab, "") & vbTab & bWordBefehl

    Beschreibung = s
End Function
```



Die Beispieldatei *Bsp16\_02.dot* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap16*.

## Tastenbelegungen eines Befehls ermitteln

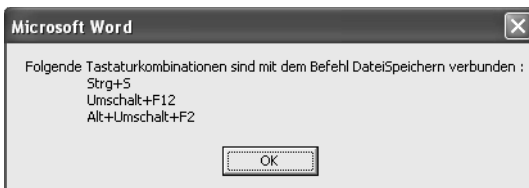
Gelegentlich will man wissen, welche Tastenkombinationen, wenn überhaupt, mit einem bestimmten Befehl verbunden sind. Es wäre möglich, wie oben beschrieben, durch alle KeyBindings durchzuschleifen, und die Command-Eigenschaft zu prüfen. Dies wäre jedoch kompliziert und zudem relativ langsam.

Keys-  
BoundTo

Das Objektmodell bietet die KeysBoundTo-Eigenschaft des Application-Objekts, um dem Entwickler diese Arbeit abzunehmen. Diese Eigenschaft gibt eine Auflistung der KeyBindings zurück, die mit einem Befehl einer bestimmten Kategorie verbunden sind. Die Syntax lautet:

```
KeysBoundTo(KeyCategory, Command, CommandParameter)
```

Das Listing 16.3 veranschaulicht den Gebrauch dieser Eigenschaft. Alle Tastaturbelegungen für den Befehl DateiSpeichern werden in einer Liste angezeigt. Vergessen Sie nicht, dass der Eigenschaft die englische Bezeichnung des Befehls zu übergeben ist.

**Abbildg. 16.4** Die Tastaturbelegungen für den Befehl *DateiSpeichern*


**Listing 16.3** Mit *KeysBoundTo* können alle Tastaturbelegungen eines Befehls ermittelt werden.

```
Sub AlleTastenkombinationenDateiSpeichern()
    Dim sCommand As String
    Dim lKeyCategory As Long
    Dim kb As Word.KeyBinding
    Dim sList As String

    sCommand = "FileSave"
    lKeyCategory = wdKeyCategoryCommand

    sList = "Folgende Tastaturkombinationen sind mit dem Befehl " & _
        "DateiSpeichern verbunden : "
    For Each kb In Application.KeysBoundTo(lKeyCategory, sCommand)
        sList = sList & vbCrLf & vbCrLf & kb.KeyString
    Next
    MsgBox sList
End Sub
```



Die Beispieldatei *Bsp16\_03.dot* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap16*.

## Tastenkombinationen verwalten

Nachdem nun die Grundlagen bekannt sind, schauen wir uns die Befehle an, um Tastenbelegungen zu verwalten.

### Tastenbelegungen entfernen

ClearAll  
-Methode

Auch hier unterscheidet Word zwischen seinen eigenen und benutzerdefinierten Tastenkombinationen. Die Methode *ClearAll* entfernt alle *benutzerdefinierten* Tastenbelegungen des angegebenen Kontextes.

```
Application.Customizationcontext = ActiveDocument
Application.KeyBindings.ClearAll
```

Clear-  
Methode

Mit der *Clear*-Methode wird eine bestimmte, *benutzerdefinierte* *KeyBinding* aus dem aktuellen Kontext gelöscht.

```
'Die Kombination Alt+A wird entfernt
Application.FindKey(BuildKeyCode(wdKeyAlt, wdKeyA)).Clear
```

Disable-  
Methode

Word-eigene Tastenkombinationen können nicht entfernt werden. Sie dürfen lediglich unterdrückt werden, was mit der Methode *Disable* gemacht wird. Das eventuell in den Menüs aufgeführte Kürzel einer gesperrten Tastenkombination wird automatisch entfernt. Wird die *Clear*-Methode auf eine gesperrte, Word-eigene Tastenbelegung ausgeführt, wird die Sperrung aufgehoben und der standardmäßige Befehl kann damit wieder ausgeführt werden. Gleichzeitig erscheint das Kürzel wieder in den Menüs.

```
'Die Kombination Strg+S hat keine Wirkung; Strg+S verschwindet auch aus dem Menü Datei
Application.FindKey(BuildKeyCode(wdKeyControl, wdKeyS)).Disable
'Die Kombination Strg+S führt wieder Datei/Speichern aus
Application.FindKey(BuildKeyCode(wdKeyControl, wdKeyS)).Clear
```

**HINWEIS** Gibt es mehr als eine Word-interne Tastenbelegung für einen Wordbefehl, wird diese im Menü erscheinen anstelle einer gesperrten. Im obigen Beispiel erscheint beispielsweise Umschalt+F12 anstelle von Strg+S.

Benutzerdefinierte Tastenbelegungen können zwar ebenfalls mit Disable ausgeschaltet werden. In diesem Fall hebt Clear die Sperrung allerdings nicht auf; die Verbindung zwischen Befehl und der Tastenkombination geht verloren.



## Tastenkombinationen definieren

Add-  
Methode





Wie für die meisten Objekte des Objektmodells werden neue Tastaturbelegungen mit der Add-Methode definiert. Die Syntax lautet:

```
Application.Add(KeyCategory, Command, KeyCode, [KeyCode2], [CommandParameter])
```

Mit den Argumenten sind Sie aus der vorangehenden Diskussion schon vertraut. KeyCategory, Command, sowie KeyCode sind erforderlich. Eine Liste gültiger CommandParameter befindet sich in der Hilfe zu diesem Thema.

Diese Methode ermöglicht es, Tastenbelegungen zu definieren, die im *Extras/Anpassen/Tastatur* nicht erlaubt sind, wie etwa Kombinationen mit den - und -Tasten.


**HINWEIS** Um zu ermitteln, ob eine Taste oder Tastenkombination in *Extras/Anpassen/Tastatur* benutzt werden darf, kann ihre Protected-Eigenschaft abgefragt werden.

Folgender Codeschnipsel zeigt auf, wie ein Satz Tastenkombinationen erstellt wird, um eine Gruppe ähnlicher AutoText-Einträge einzufügen. Jede Kombination beginnt mit +, gefolgt von einer anderen Taste. Bitte beachten Sie, dass damit allein + der KeyCategory wdCategoryPrefix zugewiesen wird.

```
Application.Customizationcontext = ActiveDocument.AttachedTemplate
With Application.KeyBindings
    .Add KeyCategory:=wdKeyCategoryAutoText, Command:="BegrüßungFormel", _
        KeyCode:=BuildKeyCode(wdKeyAlt, wdKeyT), KeyCode2:=wdKeyF
    .Add KeyCategory:=wdKeyCategoryAutoText, Command:="BegrüßungNeutral", _
        KeyCode:=BuildKeyCode(wdKeyAlt, wdKeyT), KeyCode2:=wdKeyN
    .Add KeyCategory:=wdKeyCategoryAutoText, Command:="BegrüßungFamiliär", _
        KeyCode:=BuildKeyCode(wdKeyAlt, wdKeyT), KeyCode2:=wdKeyF
End With
```





**HINWEIS**

Oft wird gefragt, wie in Formularfeldern das Einfügen eines neuen Absatzes durch drücken der -Taste unterbunden werden kann. Dazu steht das Knowledge Base-Artikel »How to code the ENTER key to move between form fields in a protected form in Word« bereit bei <http://support.microsoft.com/default.aspx?scid=kb;en-us;211219>.

**Rebind**

Es gibt auch die Methode Rebind des KeyBinding-Objekts. Damit kann einer bestehenden Tastenkombination ein anderer Befehl zugewiesen werden. Die Syntax hierfür lautet

```
Rebind(KeyCategory, Command, CommandParameter)
```

Um der Tastenkombination +, die gegenwärtig mit einem Makro verbunden ist, die Formatvorlage »FarbigerText« zuzuweisen:

```
Dim kb as Word.Keybinding
Set kb = Application.FindKey(BuildKeyCode(wdKeyAlt, wdKeyF))
kb.Rebind KeyCategory:=wdKeyCategoryStyle, Command="FarbigerText"
```

Bitte beachten Sie, dass Sie durchaus die Methode Add benutzen dürfen, um eine bestehende Kombination mit einem anderen Befehl zu belegen. Rebind bietet sich an, wenn im Code bereits ein KeyBinding-Objekt vorliegt.

## Tastenbelegung mit COM-Add-In verbinden

Die gängige Literatur beschreibt die Erstellung von COM-Add-Ins und deren Verbindung mit Symbolleisten in der Word-Anwendung. Was weniger diskutiert wird, ist der Aufruf von Prozeduren eines COM-Add-Ins durch Tastenkombinationen in Word, weshalb wir hier die Problematik kurz vorstellen.

Bekanntlich können Tastenkombinationen nur öffentliche Prozeduren eines Word-Moduls zugewiesen werden. Es ist also nicht möglich, eine Tastenkombination unmittelbar mit einer Prozedur zu verbinden, die Teil eines COM-Add-Ins bildet. Dazu braucht es Code in der Word-Umgebung, die die Verbindung vermittelt: eine so genannte »Callback«-Prozedur.

### Ein Beispiel

Das Listing 16.4 beinhaltet den Code, der sich in der Word-Umgebung befindet. Im Fall dieses Beispiels ist das Modul Teil eines Vorlagen-Add-Ins, das im *Startup*-Ordner gespeichert wird. Eine globale Variable des Typs Object wird deklariert; ihr wird das COM-Add-In zugewiesen.

Die Prozedur AltL\_TasteVerbinden wird durch das COM-Add-In aufgerufen und sich selbst als Argument angeben. Der Kontext für die Tastenbelegung wird festgelegt (die Vorlage selbst) und diese dann erstellt. Um dem Benutzer die Speicheraufforderung zu ersparen, wird die Saved-Eigenschaft auf True festgelegt.

Die Tastenkombination wird mit dem Makro AltL\_Makro verbunden, das seinerseits eine Prozedur (AltL\_Prozedur) im COMAdd-In aufruft.

Die letzte Prozedur dieses Listings befindet sich im ThisDocument-Modul und wird ausgeführt, wenn das Vorlage-Add-In geschlossen wird (beim Beenden der Word-Anwendung). Sie gibt das COM-Objekt frei und sorgt dafür, dass keine Speicheraufforderung erscheint.

**Listing 16.4** Code im VBA-Modul, um eine Tastenkombination mit einer Prozedur in einem COM-Add-In zu verbinden

```
'Code im normalen Modul in Word-Add-In-Vorlage
Dim COMObject As Object

Public Sub AltL_TasteVerbinden(ca As Object)
    Set COMObject = ca
    CustomizationContext = ThisDocument
    Application.KeyBindings.Add wdKeyCategoryCommand, "AltL_Makro", _
        BuildKeyCode(wdKeyAlt, wdKeyL)
    ThisDocument.Saved = True
End Sub

Public Sub AltL_Makro()
    COMObject.AltL_Prozedur
End Sub

'Code in ThisDocument-Modul
Private Sub Document_Close()
    Set ca = Nothing
    ThisDocument.Saved = True
End Sub
```

Der Code im COM-Add-In ist in Listing 16.5 ersichtlich. Da es die Word-Anwendung automatisiert, wird dafür anfangs eine Objektvariable deklariert. Diese wird in der Ereignis-Prozedur `AddinInstance_OnConnection` instanziiert, die beim Aktivieren des COM-Add-Ins ausgelöst wird.

---

**HINWEIS** Dieses Beispiel wird beim Starten von Word geladen.

---

Der Aufruf der Prozedur in Word findet in der Ereignis-Prozedur `AddinInstance_OnStartupComplete` statt. Da sich die Prozedur in einem Add-In befindet, muss diese Datei zuerst geöffnet werden, bevor `AltL_TasteVerbinden` zur Verfügung steht. Deshalb wird durch alle geladenen Add-Ins geschleift, bis das gewünschte gefunden wird. Die Pfadangabe zur Vorlage dient dazu, diese Datei als Dokument zu öffnen. Das Makro wird ausgeführt und die Vorlage anschließend geschlossen.

---

**HINWEIS** Mehr über den Aufruf von Prozeduren in Vorlagen-Add-Ins steht in Kapitel 9 beschrieben.

---

Beim Beenden von Word wird das COM-Add-In entladen, was das `OnDisconnection`-Ereignis auslöst. Darin wird die Objekt-Variable für die Word-Anwendung freigestellt.

**Listing 16.5** Code im COM-Add-In

```
'Code in COM-Add-In Modul
Dim wdApp As Word.Application

Private Sub AddinInstance_OnConnection(ByVal Application As Object, _
```

Listing 16.5 Code im COM-Add-In (Fortsetzung)

```

ByVal ConnectMode As AddInDesignerObjects.ext_ConnectMode, _
ByVal AddInInst As Object, custom() As Variant)

    Set wdApp = Application
End Sub

Public Sub AltL_Prozedur()
    MsgBox "Die Tastaturkombination Alt+L wurde betätigt."
End Sub

Private Sub AddinInstance_OnDisconnection(ByVal RemoveMode As
AddInDesignerObjects.ext_DisconnectMode, custom() As Variant)
    Set wdApp = Nothing
End Sub

Private Sub AddinInstance_OnStartupComplete(custom() As Variant)
    Dim adin As Word.AddIn
    Dim path As String
    Dim bAddinGeladen As Boolean
    Dim doc As Word.Document

    For Each adin In wdApp.Addins
        If adin.Name = "COMAddin_Test.dot" Then
            path = adin.path & "\" & adin.Name
            bAddinGeladen = True
        End If
    Next
    If bAddinGeladen Then
        wdApp.ScreenUpdating = False
        Set doc = wdApp.Documents.Open(Filename:=path, AddToRecentFiles:=False, _
            Visible:=False)
        wdApp.Run "AltL_TasteVerbinden", Me
        doc.Close SaveChanges:=wdDoNotSaveChanges
        Set doc = Nothing
        wdApp.ScreenUpdating = True
    End If
End Sub

```

## Das Beispiel installieren

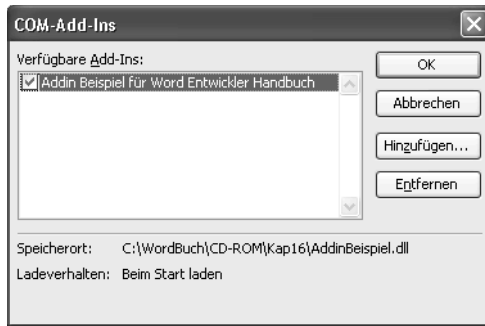
Es ist nicht schwer, ein COM-Add-In zu installieren. Gehen Sie wie folgt vor:

1. Beenden Sie Word.
2. Kopieren Sie die Vorlage *COMAddin\_Test.dot* in den *StartUp*-Ordner Ihres Rechners.
3. Das COM-Add-In, *AddinBeispiel.dll*, darf sich in einem beliebigen Ordner befinden.
4. Starten Sie Word. Fügen Sie den Menübefehl *COM-Add-Ins* der Umgebung zu:
  - Rufen Sie den Menübefehl *Extras/Anpassen* auf und aktivieren Sie die Registerkarte *Befehle*.
  - Im Listenfeld *Kategorie* wählen Sie den Eintrag *Extras*.
  - Ziehen Sie aus der Liste *Befehle* den Eintrag *COM-Add-Ins...* in ein Menü (beispielsweise unter den Eintrag *Vorlagen und Add-Ins* des Menüs *Extra*) oder in eine Symbolleiste.

- Klicken Sie auf *Schließen*, um das Dialogfeld wieder zu verlassen.
- 5. Führen Sie den Menübefehl *COM-Add-Ins* aus; das Dialogfeld in Abbildung 16.5 erscheint.

Abbildg. 16.5

COM-Add-Ins verwalten



- 6. Klicken Sie auf *Hinzufügen* und navigieren Sie zum Ordner, in dem sich das COM-Add-In befindet.
- 7. Wählen Sie dieses aus und bestätigen Sie das Dialogfeld mit *OK*. Es wird registriert und aktiviert.
- 8. Schließen Sie Word und starten Sie die Anwendung erneut.
- 9. Drücken Sie **[Alt] + [L]**. Wenn alles korrekt läuft, sollte das folgende Meldungsfeld erscheinen.

Abbildg. 16.6

Die Tastenkombination blendet dieses Meldungsfeld ein.



#### HINWEIS

Falls Sie die DLL-Datei in einen anderen Ordner verschieben, müssen Sie die Schritte 5 bis 8 wiederholen.

Um die Funktionalität zu entfernen, deaktivieren Sie das COM-Add-In über das oben dargestellte Dialogfeld.



Die Beispieldateien *COMAddin\_Test.dot* sowie *AddinBeispiel.dll* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap16*. Falls Sie mit einer anderen Version als Word 2003 arbeiten, benutzen Sie die Datei *AddinBeispiel\_AlleVersionen.dll*. Diese benutzt Late statt Early Binding für die Verknüpfung zu Word.

# Zusammenfassung

Dieses Kapitel befasst sich mit der programmtechnischen Verwaltung und Erstellung von Tastenkombinationen. Folgende Schwerpunkte wurden veranschaulicht:

- Wie Tastenbelegungen im Objektmodell verwaltet werden (Seite 667).
- Wie die Tastenkombinationen eines Kontextes ermittelt werden (Seite 669).
- Wie herausgefunden wird, welche Tastenkombinationen einem bestimmten Befehl zugewiesen sind (Seite 674).
- Wie Tastenbelegungen entfernt werden (Seite 675).
- Wie neue Tastenkombinationen hinzugefügt werden (Seite 676).
- Wie eine Prozedur in einem COM-Add-In einer Tastenkombination in Word zugewiesen werden kann (Seite 677).



## Kapitel 17

# Aufgabenbereiche

### In diesem Kapitel:

Allgemeines zum Aufgabenbereich oder, was steckt dahinter	684
Der programmtechnische Umgang mit Aufgabenbereichen (Task Panes)	688
Zusammenfassung	706

In Word 2002 wurde eine neue Funktionalität eingeführt: der Aufgabenbereich. Damit kann der Benutzer im Dokument arbeiten und hat gleichzeitig Zugang zu Befehlen und Werkzeugen, die ihm damit helfen. Entwickler stellen naturgemäß die Frage, ob es möglich wäre, die Word-internen Aufgabenbereiche zu ändern oder gar eigene zu erstellen.

Für eine Diskussion des letzten Punktes lesen Sie bitte in Kapitel 30 zum Thema »SmartDocuments« nach. Kurz gefasst lautet die Antwort »Nein«. Das Objektmodell von Word sieht dies nicht vor.

---

**HINWEIS** VSTO 2.0, ein Teil von Visual Studio 2005, bietet neue Möglichkeiten, Aufgabenbereiche programmtechnisch zu definieren. Mehr hierzu finden Sie auf der MSDN-Seite von Microsoft.com.

---

Was den ersten Teil dieser Frage angeht – Word-eigene Arbeitsbereiche den eigenen Bedürfnissen anpassen – stehen uns nur beschränkte Möglichkeiten zur Verfügung. Unser australische MVP-Kollege, Steve Hudson, hat sie eingehend erforscht und uns erfreulicherweise die Erlaubnis gegeben, seine Erkenntnisse für dieses Buch zu übersetzen.




---

**WICHTIG** Während seinen Nachforschungen hat Steve Hudson festgestellt, dass die von Microsoft zur Verfügung gestellten Schnittstellen für die Anpassung von Aufgabenbereichen unvollständig sind. Unter Umständen können Automatisierungsversuche Word ziemlich durcheinander bringen. Falls Word ausschert und sich nicht mehr bändigen lässt, benennen Sie die *Normal.dot* um und löschen Sie den Schlüssel *Data* in der Registrierung (eine ausführliche Beschreibung dieser Schritte finden Sie in Anhang E).

---

## Allgemeines zum Aufgabenbereich oder, was steckt dahinter

Ganz genau genommen erscheint ein Aufgabenbereich innerhalb eines »Arbeitsbereichs« (Work Pane), ähnlich wie ein Dokument sich immer in einem Dokumentfenster befindet. Um die Analogie weiterzuführen, ein Work Pane dient als Behälter für jeden in der Anwendung definierten Aufgabenbereich. Der Begriff »Work Pane« ist kaum dokumentiert und wird auch im Objektmodell nicht konsequent verwendet. Genau genommen müsste es beispielsweise *Ansicht/Arbeitsbereich* statt *Ansicht/Aufgabenbereich* heißen.

Ein Work Pane besteht aus zwei Objekten – ein CommandBar-Objekt und ein CommandBarControl-Objekt des Typs `msoControlWorkPane` – worauf der Benutzer sowie der Entwickler nur bedingt zugreifen können. Beispielsweise wird der Befehl zum Löschen eines Menüeintrags, der mit  +  +  aufgerufen wird, außer Kraft gesetzt, sobald sich der Mauszeiger über diesem Bereich befindet. Der Makrorekorder »sieht« auch nicht alle im Bereich ausgeführten Handlungen, und ein VBA-Projekt kann die durch einen Aufgabenbereich ausgelösten Handlungen nicht abfangen (mehr zu diesem Thema finden Sie in Kapitel 18).

---

**HINWEIS** CommandBars (Symbolleisten) wurden in Kapitel 15 dargestellt.

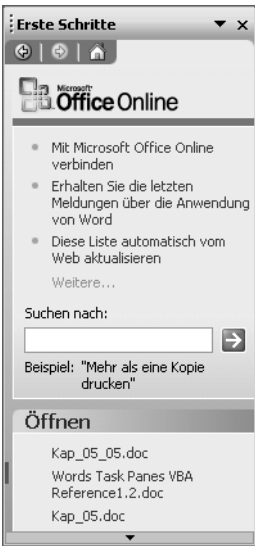
---

Die Einstellungen dieser besonderen Art von CommandBar werden, wie bei allen Einstellungen der Benutzeroberfläche, in der Registrierung im Schlüssel `HKCU\Software\Microsoft\Office\11.0\Word\Data` (110.0 ist für Word 2003; 10.0 für Word 2002; 9.0 für Word 2000) verwaltet. Die Ähn-



lichkeit zu einer üblichen Symbolleiste ist in Abbildung 17.1 unverkennbar. Das einzige Steuerelement erstreckt sich in die Höhe, statt in die Breite. Dessen Inhalt ist grundsätzlich HTML-Code, der seinerseits weitere ActiveX-Steuerelemente definiert, um die Skripts auszuführen. Standardmäßig steht der Aufgabenbereich immer bündig zum rechten Rand, er darf jedoch links, oben oder unten neben anderen Symbolleisten oder freistehend über dem Dokumentfenster liegen (was allerdings seiner Größe wegen kaum wünschenswert wäre).

Abbildg. 17.1 Ein Work Pane in Word 2003



Die einzigen CommandBar-Eigenschaften, die der Entwickler ändern darf, sind Height, Left, Position, RowIndex, Top, Visible und Width. Die meisten anderen sind les-, jedoch nicht schreibbar (siehe Tabelle 17.1).

## Objektmodell-Schnittstellen für Work Panes

Die Objekte der Tabelle 17.1 und einige ihrer Eigenschaften sind, wenn Sie das Kapitel 15 schon gelesen haben, alte Bekannte. Damit werden Sichtbarkeit und Position des Work Panes, sofern möglich, kontrolliert.

**HINWEIS** Wie Sie mit einzelnen Aufgabenbereichen umgehen, steht im Abschnitt »Bestimmte Aufgabenbereiche manipulieren« in diesem Kapitel beschrieben.

Tabelle 17.1 Eigenschaften des Work Pane-Objekts (eine Symbolleiste)

Objekt	Eigenschaft/Methode	Datentyp
Application	ShowStartupDialog	Boolean
CommandBars("Task Pane")	Height	Long

**Tabelle 17.1** Eigenschaften des Work Pane-Objekts (eine Symbolleiste) (Fortsetzung)

Objekt	Eigenschaft/Methode	Datentyp
CommandBars("Task Pane")	Left	Long
CommandBars("Task Pane")	Position	msoBarPosition
CommandBars("Task Pane")	RowIndex	Long
CommandBars("Task Pane")	Top	Long
CommandBars("Task Pane")	Visible	Boolean
CommandBars("Task Pane")	Width	Long
<i>(nur lesbar)</i>		
CommandBars("Task Pane")	Controls(index)	
CommandBars("Task Pane")	Name	String
CommandBars("Task Pane")	NameLocal	String
<i>(Eigenschaften der Control-Eigenschaft – nur lesbar)</i>		Wert
.Controls(index)	Caption	Name des Aufgabenbereichs
	Left	3 + Bildschirmbreite – CommandBars("Task Pane").Width
	ID	5746
	OLEUsage	msoControlOLEUsageServer (=1)
	Top	Gerechnet vom Work Pane
	Type	msoControlWorkPane (=25)
	Width	Breite des Work Panes – 6

## Registry-Einträge

Der folgende Eintrag reguliert, ob der Aufgabenbereich *Neues Dokument* geöffnet bleibt, wenn ein neues Dokument erstellt oder eines geöffnet wird. Der Eintrag wird nicht automatisch von Office erstellt. Weisen Sie ihm den Wert 1 zu, um den Aufgabenbereich dazu zu zwingen, offen zu bleiben.

**WICHTIG** Gehen Sie sorgfältig mit diesem Eintrag um, da er die Option *Startaufgabenbereich* in der Menüfolge *Extras/Optionen* auf der Registerkarte *Ansicht* überlagert, was dem Benutzer unter Umständen nicht gefallen würde.

### Office XP

```
HKEY_CURRENT_USER\Software\Microsoft\Office\10.0\Common\General\DoNotDismissFileNewTaskPane
```

## Office 2003

```
HKEY_CURRENT_USER\Software\Microsoft\Office\11.0\Common\General\DoNotDismissFileNewTaskPane
```

Falls ein Add-In beim Starten von Word geladen wird, wird der Startaufgabenbereich *Erste Schritte* standardmäßig geschlossen. Dieser Eintrag wurde eingeführt, um den Startaufgabenbereich bei einem Wert von 1 geöffnet zu halten. Auch er wird nicht automatisch von Office erstellt.

## Office XP SP-3

```
HKEY_CURRENT_USER\Software\Microsoft\Office\10.0\Word\Options\StartupDialog
```

## Office 2003

```
HKEY_CURRENT_USER\Software\Microsoft\Office\11.0\Word\Options\StartupDialog
```

## Fehlermeldungen

Es gibt einige Fehlermeldungen, die auftauchen könnten, wenn Sie mit dem Work Pane-Objekt arbeiten. Diese gelten für alle Aufgabenbereiche und sind in der Arbeitsmappe *errormsg.xls* des Office Resource Kits nicht dokumentiert, da sie aus externen Bibliotheken stammen, hauptsächlich der von InfoPath.

**ACHTUNG** Achten Sie insbesondere auf die zwei Fehlermeldungen mit der gleichen Nummer!

**Tabelle 17.2** Fehlermeldungen, die in Verbindung mit Aufgabenbereichen erscheinen können

Nummer	Beschreibung	Mögliche Ursache
4605	Dieser Befehl ist nicht verfügbar.	Es wurde versucht, in der falschen Umgebung (keine Smart Document Lösung vorhanden), den Aufgabenbereich <b>wdTaskPaneDocumentActions</b> einzublenden.
4605	Diese Methode oder Eigenschaft ist nicht verfügbar, weil das aktuelle Dokument verändert wurde oder nicht mit einer XML-Transformation verbunden ist.	Es wurde versucht, in der falschen Umgebung (kein XML-Dokument vorhanden), den Aufgabenbereich <b>wdTaskPaneXMLDocument</b> einzublenden.
4605	Diese Methode oder Eigenschaft ist nicht verfügbar, weil kein Dokumentfenster aktiv ist.	Es wurde versucht, einen Aufgabenbereich einzublenden, ohne dass ein Dokument in der Word-Umgebung geöffnet ist.
5941	Anwendungs- oder objektdefinierter Fehler	Der verwendete <b>TaskPanes</b> -Indexwert existiert nicht.
6162	Dieser Befehl ist außerhalb des Fax-E-Mail-Umschlags nicht verfügbar.	Es wurde versucht, in der falschen Umgebung den Aufgabenbereich <b>wdTaskPaneFaxService</b> einzublenden.

**Tabelle 17.2** Fehlermeldungen, die in Verbindung mit Aufgabenbereichen erscheinen können (Fortsetzung)

Nummer	Beschreibung	Mögliche Ursache
-2147467259	Automatisierungsfehler	Es wurde versucht, einen Aufgabenbereich sichtbar zu machen, der im Dokument noch nie eingeblendet wurde. Oder Es wurde versucht, eine Eigenschaft zu ändern, die nur lesbar ist.

Wie in Kapitel 3 diskutiert, ist es oft besser, voraussehbaren Fehler auszuweichen, anstatt sie abzufangen. Die Tests in Tabelle 17.3 sind nützlich, um festzustellen, ob die Umgebung gewisse Aufgabenbereiche unterstützt.

**Tabelle 17.3** Prüfen, ob die Umgebung das Anzeigen eines Aufgabenbereichs unterstützt

Funktionalität	Prüfen mit
Geöffnetes Dokument vorhanden	<code>Application.Documents.Count &gt; 0</code>
SharePoint vorhanden	<code>ActiveDocument.SharedWorkspace.Connected</code>
XML im Dokument	<code>ActiveDocument.XMLNodes.Count &gt; 0</code>
XSL- und XSD-Unterstützung vorhanden	<code>Application.ArbitraryXMLSupportAvailable</code> <code>ActiveDocument.XMLSchemaReferences.Count &gt; 0</code>
Online-Fax-Dienstleistungsanbieter (Service Provider)	Kontrollieren, ob folgender Registry-Schlüssel vorhanden ist: <code>HKEY_CURRENT_USER\Software\Microsoft\Office\11.0\Common\Services\Fax</code>

## Der programmtechnische Umgang mit Aufgabenbereichen (Task Panes)

Wenn wir in diesem Kapitel von »Task Panes« reden, reden wir von einem `CommandBar.Control` des Typs `msoControlWorkPane`, das den Inhalt der einzelnen Aufgabenbereiche definiert. Es gibt zwei Arten von Aufgabenbereichen; diejenige, wofür Microsoft einen `WdTaskPanes`-Wert definiert hat, und alle anderen, die nicht so direkt angesprochen werden können.

Die Tabelle 17.4 listet die Aufgabenbereiche auf mit ihren `WdTaskPanes`-Typen, deren Werte, die ID-Nummer des `CommandBar.Controls`, in welcher Word-Version der Aufgabenbereich eingeführt wurde, und zu welchem Kontext er gehört. Als Faustregel gilt: Hat ein Aufgabenbereich einen `WdTaskPanes`-Konstantwert, soll dieser darüber anstatt über die `Control.ID` angesprochen werden.

**Tabelle 17.4** Die von Word zur Verfügung gestellten Aufgabenbereiche mit Informationen für den programmtechnischen Umgang

Beschriftung	WdTaskPanes-Enumerator	Wert	Command-Bar.Control ID	Eingeführt	Kontext
<i>Erste Schritte</i>	[kein]	[kein]	[kein]	Word 2003	Wird beim Starten von Word eingeblendet. Enthält Links zu Office Online sowie zu den zuletzt geöffneten Dateien.
<i>ClipArt</i>	[kein]	[kein]	682	Word 2002	ClipArt einfügen
<i>Neues Dokument</i>	[kein]	[kein]	18	Word 2002	Bietet Listen von Dokumentarten, Speicherorte für Vorlagen und der zuletzt benutzten Vorlagen an.
<i>Zwischenablage</i>	[kein]	[kein]	809	Word 2002	Enthält die Einträge in der Office-Zwischenablage. Ein programmatischer Zugriff auf den Inhalt dieser Zwischenablage steht im Word-Objektmodell nicht zur Verfügung.
<i>Dokumentaktionen</i>	<i>wdTaskPaneDocument-Actions</i>	7	[kein]	Word 2003	Eine Smart Document-Lösung ist aktiv.
<i>Dokument schützen</i>	<i>wdTaskPaneDocument-Protection</i>	6	7116	Word 2003	Schnittstelle für die Festlegung der Schutzart für ein Dokument und aktiviert diesen Schutz.
<i>Dokumentaktualisierungen</i>	<i>wdTaskPaneDocument-Updates</i>	13	7423	Word 2003	Das aktive Dokument ist eine Kopie eines im Dokumentarbeitsbereich vorhandenen Dokument.
<i>Faxdienste</i>	<i>wdTaskPaneFaxService</i>	11	[kein]	Word 2003	Eine Faxdienstleistung ist verfügbar.
<i>Formatvorlagen und Formatierungen</i>	<i>wdTaskPaneFormatting</i>	0		Word 2002	Stellt eine Liste von Formatvorlagen und im Dokument vorhandenen Formatierungen zur Verfügung.

**Tabelle 17.4** Die von Word zur Verfügung gestellten Aufgabenbereiche mit Informationen für den programmtechnischen Umgang (Fortsetzung)

Beschriftung	WdTaskPanes-Enumerator	Wert	Command-Bar.Control ID	Eingeführt	Kontext
Hilfe	wdTaskPaneHelp	9	984	Word 2003	Schnittstelle für die Suche nach einem Thema in der Hilfe und zeigt eine Liste der Suchergebnisse an.
Seriendruck	wdTaskPaneMailMerge	2	6070	Word 2002	Enthält sechs Schritte, die den Benutzer durch die Erstellung eines Seriendrucks führt. Kann teilweise mit VBA beeinflusst werden (siehe Kapitel 6).
Recherchieren	wdTaskPaneResearch	10	7343	Word 2003	<p>Schnittstelle für das Recherchieren eines Themas. Bietet standardmäßig u.a. den Thesaurus sowie Übersetzungsdienste an, falls die Werkzeuge installiert sind.</p> <p>Die dem Benutzer zur Verfügung gestellten Dienste sind Webdienste und können erweitert werden. Siehe dazu den Artikel »Customizing the Microsoft Office 2003 Research Task Pane« auf MSDN <a href="http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dno2k3ta/html/odc_customizingthe_researchpane.asp">http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dno2k3ta/html/odc_customizingthe_researchpane.asp</a></p>
Formatierungen anzeigen	wdTaskPaneReveal-Formatting	1	6094	Word 2002	Zeigt an, mit welchen Formatierungen die Markierung formatiert wurde, und ob diese direkt oder von einer bestimmten Formatvorlage stammen.

**Tabelle 17.4** Die von Word zur Verfügung gestellten Aufgabenbereiche mit Informationen für den programmtechnischen Umgang (Fortsetzung)

Beschriftung	WdTaskPanes-Enumerator	Wert	Command-Bar.Control ID	Eingeführt	Kontext
Einfache Suchoptionen	wdTaskPaneSearch	4	5905	Word 2002	Eine Schnittstelle, um nach Dateien zu suchen
Freigegebener Arbeitsbereich	wdTaskPaneShared-Workspace	8	7710	Word 2003	Verwaltung von Dokumenten, die auf einer Sharepoint-Seite gespeichert sind
Recherchieren	wdTaskPaneTranslate	3	7021	Word 2002	Der Arbeitsbereich <i>Recherchieren</i> im Übersetzungsmodus
XML-Dokument	wdTaskPaneXMLDocument	12		Word 2003	Wird beim Öffnen eines XML-Dokuments eingeblendet und bietet XSLTransforms an, die mit dem Dokument oder seinem Schema verknüpft sind. Nur gültig, wenn das XML-Dokument noch nicht verändert wurde (siehe auch Kapitel 30).
XML-Struktur	wdTaskPaneXMLStructure	5			Zeigt die Benutzerdefinierten XML-Tags in einem Dokument als Baumstruktur an sowie eine Liste der XML-Tags im angehängten Schema, die ins Dokument eingefügt werden können (siehe auch Kapitel 30).

## Allgemeine VBA-Schnittstellen für die Arbeit mit Aufgabenbereichen

Zuerst werden wir die Eigenschaften und Methoden behandeln, die für beide Arten gelten.

### Welcher Aufgabenbereich ist eingeblendet

Das folgende Codeschnipsel zeigt, wie man feststellt, welcher Aufgabenbereich gegenwärtig im Work Pane sichtbar ist (die Eigenschaft `Caption` ist nur lesbar).

```
Dim strTaskPaneName as String
strTaskPaneName = CommandBars("Task Pane").Controls(1).Caption
```

## Einen bestimmten Aufgabenbereich einblenden

Aufgabenbereiche, die einen wdTaskPane-Typ haben, können direkt ein- und ausgeblendet werden. Denken Sie daran, dass bestimmte Aufgabenbereiche erst eingeblendet werden dürfen, wenn ein Dokument zur Verfügung steht. In diesem Fall sollen Sie immer zuerst die Anzahl der Dokumente kontrollieren:

```
If Application.Documents.Count > 0 Then
    Application.TaskPanes(wdTaskPaneFormatting).Visible = True
Else
    MsgBox "Öffnen Sie zuerst ein Dokument."
End If
```

### PROFITIPP

Um beim Starten von Word einen bestimmten Aufgabenbereich einzublenden, brauchen Sie ein Makro namens AutoExec (mehr darüber lesen Sie im Kapitel 5), das sich in einer globalen Vorlage befindet und daher beim Starten ausgeführt wird.

Manche Aufgabenbereiche setzen ein offenes Dokument voraus. Falls Sie beim Starten von Word einen solchen einblenden wollen, kontrollieren Sie in einer Schleife die Anzahl Dokumente. Sobald die Anzahl größer ist als 0 darf der Aufgabenbereich eingeblendet werden. Vergessen Sie nicht, einen »Notausgang« einzubauen, falls Word ohne Dokument gestartet wird.

```
counter = 0
Do While Application.Documents.Count = 0
    'Die Gefahr besteht, dass kein Dokumentfenster geöffnet wird (/n-Schalter)
    'Um eine endlose Schleife zu vermeiden, einen Ausstiegspunkt festlegen
    counter = counter + 1
    If counter > 5000 Then Exit Sub
Loop
Application.TaskPanes(wdTaskPaneFormatting).Visible = True
```

Hat Microsoft einem Aufgabenbereich keinen wdTaskPane-Wert zugewiesen, braucht es einen kleinen Umweg:

```
CommandBars.FindControl(ID:=<Num>).Execute
```

wobei <Num> der Spalte CommandBar.Control ID in Tabelle 17.4 zu entnehmen ist.

Diese Aufgabenbereiche können nicht direkt ausgeblendet werden. Entweder muss das Work Pane geschlossen (CommandBars("Task pane").Visible = False) oder an seiner Stelle ein anderer Aufgabenbereich geöffnet werden.



## Inhalt eines Aufgabenbereichs aktualisieren

Falls Sie programmtechnisch den Inhalt eines Aufgabenbereichs aktualisieren, werden diese Änderungen nicht immer automatisch wiedergeben. Um eine Aktualisierung zu erzwingen, blenden Sie den Aufgabenbereich aus und anschließend wieder ein:

```
With CommandBars("Task Pane")
    .Visible = False
    .Visible = True
End With
```

## Aufgabenbereiche mit VBA abfangen oder sie außer Kraft setzen

Die meisten Befehle, die Aufgabenbereiche ein- oder ausblenden, sind in einem Word VBA-Projekt abfangbar: Unter Umständen können Sie auch eine Alternative anbieten. Diese Möglichkeiten sind in Tabelle 17.5 vorgestellt.

**Tabelle 17.5** Die Menübefehlsfolgen (alphabetisch nach Menüpunkt), die einen Aufgabenbereich einblenden, und der VBA-Code, um die Handlung abzufangen

Menüpunkt	Codeschnipsel
<i>Ansicht/Aufgabenbereich</i>	'Fängt den Menübefehl ab Sub ViewTaskPane() End Sub
<i>Bearbeiten/Office-Zwischenablage</i> sowie der Eintrag <i>Zwischenablage</i> aus der Liste im Aufgabenbereich	'Fängt den Menübefehl ab Sub EditOfficeClipboard() End Sub
Dokumentaktionen (wird von einer SmartDocument Lösung eingesetzt)	'Fängt den Befehl ab Sub DocumentActionsPane() End Sub
<i>Datei/Dateisuche</i>	'Statt des Aufgabenbereichs Einfache Suchoptionen das 'Dialogfeld Datei/Öffnen einblenden. 'Die Dateisuche befindet sich im 'Menü Extras, also "Alt+X" durchgeben Sub FileSearch() SendKeys "%x{Enter}" Dialogs(wdDialogFileOpen).Show End Sub
<i>Datei/Neu</i> sowie der Eintrag <i>Neues Dokument</i> aus der Liste im Aufgabenbereich	'Das Datei/Neu-Dialogfeld unmittelbar anzeigen Sub FileNew() Dialogs(wdDialogFileNew).Show End Sub
<i>Dokumentaktualisierungen</i> (aus der Liste im Aufgabenbereich)	'Fängt den Befehl ab Sub ShowSmPane() End Sub
<i>Erste Schritte</i> (aus der Liste im Aufgabenbereich)	Kann nicht abgefangen werden. Einige Einträge im Online-Bereich werden durch <i>Extras/Optionen/Allgemein//Dienstoptionen/Onlineoptionen</i> (Abbildung 17.2) gesteuert.

**Tabelle 17.5** Die Menübefehlsfolgen (alphabetisch nach Menüpunkt), die einen Aufgabenbereich einblenden, und der VBA-Code, um die Handlung abzufangen (*Fortsetzung*)

Menüpunkt	Codeschnipsel
<i>Extras/Briefe und Sendungen/Serienbrief-erstellung</i> sowie der Eintrag <i>Seriendruck</i> aus der Liste im Aufgabenbereich	'Den alten Seriendruckassistent statt 'des Aufgabenbereichs einblenden (Abbildung 17.5) Sub MailMergeWizard() Dialogs(wdDialogMailMergeHelper).Show End Sub
<i>Extras/Dokument schützen</i> sowie der gleichnamige Eintrag aus der Liste im Aufgabenbereich	'Das Dialogfeld für Formatierungseinschränkungen direkt einblenden Sub ToolsProtect() Dialogs( _ wdDialogFormattingRestrictions). _ Show End Sub  'Das alte Dialogfeld für den Dokumentschutz (Abbildung 17.3) Sub ToolsProtect() Dialogs( _ wdDialogToolsProtectDocument). _ Show End Sub
<i>Extras/Freigegebener Arbeitsbereich</i> sowie der gleichnamige Eintrag aus der Liste im Aufgabenbereich	Es gibt dafür keine Schnittstelle im Objektmodell. Einige Einstellungen können im Bereich <i>Freigegebener Arbeitsbereich</i> des Dialogfelds <i>Dienstoptionen</i> unter <i>Extras/Optionen/Allgemein</i> vorgenommen werden.
<i>Extras/Recherchieren</i> sowie der gleichnamige Eintrag aus der Liste im Aufgabenbereich	'Den Befehl abfangen Sub Research() End Sub  Sub ResearchLookup End Sub
<i>Extras/Sprache/Übersetzen</i>	'Fängt den Befehl ab Sub Translate() End Sub  Sub TranslatePane() End Sub
<i>Extras/Thesaurus</i> (fängt die Suche im Aufgabenbereich <i>Recherchieren</i> <b>nicht</b> ab)	'Das alte Dialogfeld einblenden (Abbildung 17.4) Sub ToolsThesaurusRR() Dialogs(wdDialogToolsThesaurus).Show End Sub
<i>Faxdienst</i> (aus der Liste im Aufgabenbereich)	'Fängt den Befehl ab Sub FaxService() End Sub

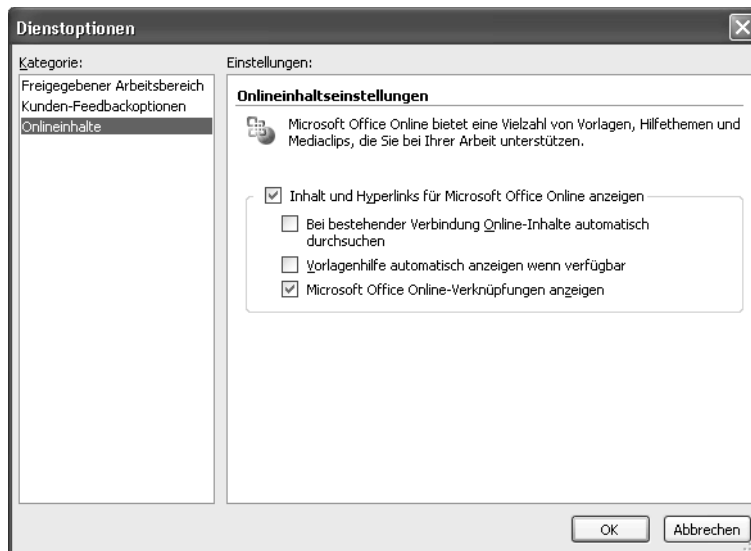
**Tabelle 17.5** Die Menübefehlsfolgen (alphabetisch nach Menüpunkt), die einen Aufgabenbereich einblenden, und der VBA-Code, um die Handlung abzufangen (Fortsetzung)

Menüpunkt	Codeschnipsel
<i>Format/Formatierung anzeigen</i> sowie der gleichnamige Eintrag aus der Liste im Aufgabenbereich	'Statt den Aufgabenbereich das Dialogfeld Format/ Zeichen einblenden Sub FormattingProperties() Dialogs(wdDialogFormatFont).Show End Sub
<i>Format/Formatvorlagen und Formatierungen</i> sowie der gleichnamige Eintrag aus der Liste im Aufgabenbereich	'Das Dialogfeld Formatvorlage anstelle des Aufgabenbereichs einblenden Sub FormattingPane() Dialogs(wdDialogFormatStyle).Show End Sub
Der Eintrag <i>XML Struktur</i> in der Liste im Aufgabenbereich	'Fängt den Befehl ab Sub ViewXMLStructure() End Sub

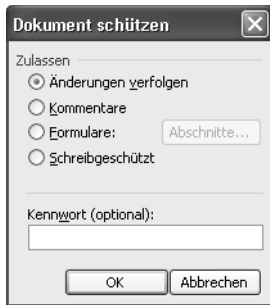


Die Beispieldatei *Bsp17\_01.doc* enthält einige der Prozedurskelette der Tabelle 17.5. Sie befindet sich auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap17*.

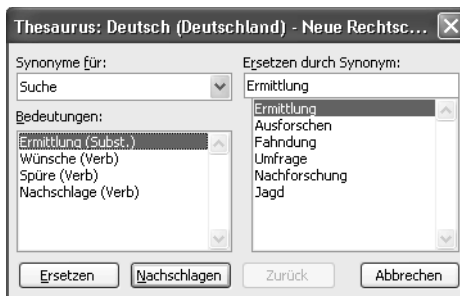
**Abbildg. 17.2** Einstellungen für Dienste, die über eine Inter- oder Intranet-Verbindung erfolgen, befinden sich im Dialogfeld *Dienstoptionen* in der Befehlsfolge *Extras/Optionen*, Registerkarte *Allgemein*, Schaltfläche *Dienstoptionen*.



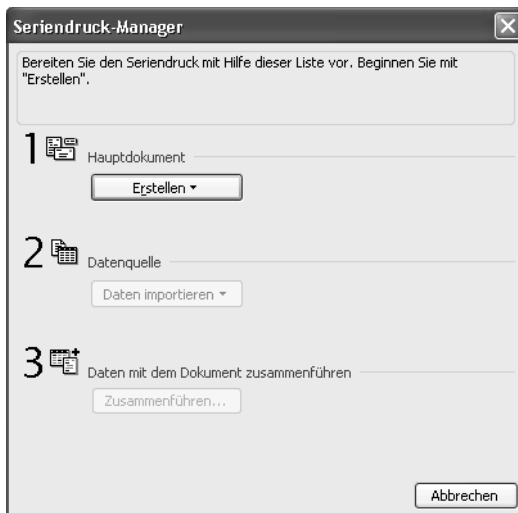
**Abbildg. 17.3** Das alte Dialogfeld für den Dokumentschutz. Es fehlen allerdings die in Word 2003 eingeführte Funktionalität für den Schutz von markierten Bereichen und die Zuweisung von Berechtigungen (IRM).



**Abbildg. 17.4** Das alte Dialogfeld für das Thesaurus ist nützlich, wenn der Benutzer die ganze Breite des Bildschirms für die Dokumentbearbeitung behalten will.



**Abbildg. 17.5** Der alte Seriendruck-Manager verbindet Datenquellen standardmäßig wie Word-Versionen vor 2002.



## Zur Verfügung stehende Aufgabenbereiche auflisten

Die Prozedur in Listing 17.1 wurde für die Nachforschungen für dieses Kapitel verwendet. Sie schleift durch die TaskPanes-Auflistung und listet alle TaskPanes in einem neuen Dokument auf. Sie veranschaulicht, wie Aufgabenbereiche eingblendet und angesprochen werden.

**ACHTUNG** Aus irgendeinem Grund wird Word nach Ausführung dieses Makros instabil und stürzt meistens bald ab. Um keine Arbeit zu verlieren, speichern Sie vor der Ausführung alle Dokumente und unmittelbar danach das neue Dokument. Anschließend beenden Sie Word und starten die Anwendung erneut.

Listing 17.1 Alle Aufgabenbereiche auflisten

```
Public Sub ShowAllTaskPanes()
    Dim docReport As Document
    Dim rngInsertion As Range
    Dim strName As String
    Dim strOldName As String
    Dim lZähler As Long

    Set docReport = Application.Documents.Add
    Set rngInsertion = docReport.Content
    rngInsertion.Collapse wdCollapseStart

    'Meldungen ausschalten
    Application.DisplayAlerts = wdAlertsNone
    On Error GoTo ErrorHandler
    'Durch die Auflistung schleifen
    For lZähler = 0 To 2 ^ 16 - 1
        Application.TaskPanes(lZähler).Visible = True

        'Die Beschriftung ermitteln
        strName = CommandBars("Task Pane").Controls(1).Caption
        'Mit dem vorherigen vergleichen
        If strName <> strOldName Then
            'Falls sie anders ist, einen neuen Eintrag schreiben
            rngInsertion.InsertAfter lZähler & vbTab & strName
            rngInsertion.InsertParagraphAfter
            strOldName = strName
        End If
    Next
    'Das Ergebnis in eine Tabelle umwandeln
    rngInsertion.ConvertToTable AutoFit:=True, _
    AutoFitBehavior:=wdAutoFitContent, _
    DefaultTableBehavior:=wdWord9TableBehavior

ErrorHandler:
    With Err
        If .Number > 0 Then
            If .Number <> 5941 Then
                'Fehlermeldung statt Beschriftung schreiben
                rngInsertion.InsertAfter lZähler & vbTab & "Err # " & .Number & "(" & .Description &
                ")"
                rngInsertion.InsertParagraphAfter
            End If
            .Clear
        End With
    End Sub
```

**Listing 17.1** Alle Aufgabenbereiche auflisten (*Fortsetzung*)

```

        Resume Next
    End If
End With

Application.DisplayAlerts = wdAlertsAll
Set rngInsertion = Nothing
Set docReport = Nothing
End Sub

```



Die Beispieldatei *Bsp17\_02.doc*, finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap17*.

## Bestimmte Aufgabenbereiche manipulieren

Nicht alle Aufgabenbereiche stellen eine VBA-Schnittstelle bereit. Nachfolgend werden diejenigen vorgestellt, die eine solche haben. Manche Aufgabenbereiche sind in Word 2002 sowie 2003 vorhanden, getestet wurde für dieses Buch jedoch nur in Word 2003, und das Verhalten in Word 2002 kann abweichen.

### Aufgabenbereich *Neues Dokument*

Der Aufgabenbereich *Neues Dokument* bietet eine Schnittstelle, um Einträge in der Liste der vorhandenen Dokumente und Vorlagen hinzuzufügen oder zu entfernen.

#### HINWEIS

Sie werden in der Hilfe *NewDocument* weder unter der Liste der Objekte für die *Add*-Methode noch unter der für die *Remove*-Methode finden. Die Informationen befinden sich unter den Einträgen für *NewFile*. Diese Klasse wird von mehreren Office-Anwendungen verwendet, wobei jede dafür einen eigenen Namen für das Objekt hat: *NewWorkbook* für Excel, beispielsweise, oder *NewPresentation* für PowerPoint.

Das *NewFile*-Objekt wurde für die Integration mit SharePoint eingeführt. Sharepoint benutzt diese Schnittstelle, um dem Benutzer Office-Dateien zur Verfügung zu stellen, statt die Anwendungen spezifisch automatisieren zu müssen.

Add-  
Methode

Die *Add*-Methode des *NewDocument*-Objekts fügt Einträge hinzu und hat die Syntax:

```
NewDocument.Add(FileName, [Section], [DisplayName], [Action])
```

- Der erforderliche Parameter *FileName* erwartet eine Zeichenkette mit den Pfadangaben der Datei. Im Aufgabenbereich dient er als *Hyperlink*.
- Der optionale Parameter *Section* legt fest, in welchem Abschnitt des Aufgabenbereichs der Eintrag erscheint. Er erwartet einen der *msoFileNewSection*-Werte: *msoOpenDocument* (0), *msoNew* (1), *msoNewFromExistingFile* (2), *msoNewFromTemplate* (3), *msoBottomSection* (4).

Vier Abschnitte sind in Abbildung 17.6 ersichtlich. Der oberste entspricht dem Wert `msoNew`, der zweite `msoNewFromTemplate`, der dritte `msoExistingFile` und die unterste `msoBottomSection`. (Der Wert `msoOpenDocument` gilt für den Aufgabenbereich *Erste Schritte*.)

Wenn Sie den Abschnitt nicht festlegen, erscheint der Eintrag im untersten Bereich.

Dateien, die vom Benutzer für die Erstellung neuer Dokumente verwendet wurden, erscheinen im dritten Abschnitt, *Zuletzt verwendete Vorlagen*. Diese werden von Word verwaltet. Ist in der Liste die maximale Anzahl von Einträgen erreicht, werden Einträge, die von einer Benutzerhandlung stammen, entfernt, im Gegensatz zu denen, die programmtechnisch hinzugefügt wurden.

Abbildg. 17.6

Der Aufgabenbereich *Neues Dokument* mit den vier Zielabschnitten, die den *Add-* sowie *Remove-*Methoden zur Verfügung stehen



- Der optionale Parameter `DisplayName` erwartet eine Zeichenkette und legt den Text fest, der im Aufgabenbereich erscheinen soll.  
Geben Sie keinen Texteintrag an, erscheint der Dateiname oder manchmal auch ein »0«.
- Der optionale Parameter `Action` legt die Handlung fest, die beim Anklicken des Eintrags auszuführen ist. Er erwartet einen der `msoNewFileAction`-Werte: `msoEditFile (0)`, `msoCreateNewFile (1)`, `msoOpenFile (2)`.

Der Wert `msoCreateNewFile` erstellt ein neues Dokument, basierend auf dem angegebenen Dokument oder der angegebenen Vorlage. Der Unterschied zwischen `msoEditFile` und `msoOpenFile` ist, dass der erste Wert die Datei in Word, während der zweite Wert sie in ein Browser-Fenster (Internet Explorer) öffnet.

Bei fehlender Angabe wird dem Eintrag den Wert `msoEditFile` zugewiesen.

Remove-  
Methode

Die `Remove`-Methode des `NewDocument`-Objekts entfernt einen Eintrag aus dem Aufgabenbereich und hat die folgende Syntax, wobei alle Parameter die gleichen sind wie bei der `Add`-Methode:

```
NewDocument.Add(Filename, [Section], [DisplayName], [Action])
```

Obwohl alle Parameter außer `Filename` optional sind, ist auch `DisplayName` für das Entfernen erforderlich, sofern er beim Hinzufügen festgelegt wurde.

### Bemerkungen

- Die von Word erstellten, standardmäßigen Einträge können nicht entfernt werden.
- Um die Änderungen im Aufgabenbereich zu sehen, müssen Sie ihn durch dessen Aus- und erneutes Einblenden aktualisieren (siehe den Abschnitt »Inhalt eines Aufgabenbereichs aktualisieren« in diesem Kapitel).
- Es ist nicht möglich, programmtechnisch eine Liste der bestehenden Einträge im Aufgabenbereich zu lesen.
- Die Schnittstelle verhindert duplizierte Einträge nicht.

Um den obigen Umständen Rechnung zu tragen, ist es daher ratsam, das Hinzufügen und Entfernen von Einträgen mit Hilfe eines »Wrapper« (eines »Umschlags«) vorzunehmen, wie in Listing 17.2 vorgestellt. Die Prozedur `NewDocumentAdd` versucht zuerst, den Eintrag im Aufgabenbereich zu entfernen, bevor sie ihn erstellt (um doppelte Einträge zu vermeiden). Da die `NewDocument.Remove`-Methode keinen Fehler verursacht, wenn ein Eintrag nicht vorhanden ist, passiert nichts, wenn der Eintrag nicht schon im gegebenen Abschnitt des Aufgabenbereichs steht.

`NewDocumentRemove` ihrerseits ermittelt einen möglichen Eintragstext (der Dateiname), falls dem Parameter `DisplayName` keine Angabe übergeben wurde.

Beide Prozeduren haben einen optionalen Parameter, um eine Aktualisierung des Aufgabenbereichs vorzunehmen.

Listing 17.2

Die Prozeduren `NewDocumentAdd` und `NewDocumentRemove` sind Wrapper für das Hinzufügen und Entfernen von Einträgen in Aufgabenbereichen.

```
Sub NeueEintraege()  
    NewDocumentAdd "C:\test\Dok1.doc", msoBottomSection, "Dok", , False  
    NewDocumentAdd "C:\test\Dok2.doc", msoBottomSection, "Dok2", , False  
    NewDocumentAdd "C:\Test\Dok3.doc", msoBottomSection, "Dok3", , True  
End Sub  
  
'Der Wrapper, um einen Eintrag hinzuzufügen  
Sub NewDocumentAdd(  
    Filename As String, _  
    Optional FileSection As MsoFileNewSection = msoOpenDocument, _  
    Optional DisplayName As String, _  
    Optional Action As MsoFileNewAction = msoEditFile, _
```



**Listing 17.2** Die Prozeduren *NewDocumentAdd* und *NewDocumentRemove* sind Wrapper für das Hinzufügen und Entfernen von Einträgen in Aufgabenbereichen. (Fortsetzung)

```

Optional Refresh As Boolean = True)

'Doppelte Einträge unterbinden
NewDocumentRemove Filename, FileSection, DisplayName, Action, False

'Hinzufügen
With Application
    .NewDocument.Add Filename, FileSection, DisplayName, Action
    If Refresh Then
        'Aktualisieren
        With .CommandBars("Task Pane")
            .Visible = False
            .Visible = True
        End With
    End If
End With

'Der Wrapper, um einen Eintrag zu entfernen
Sub NewDocumentRemove( _
    Filename As String, _
    Optional FileSection As MsoFileNewSection = msoOpenDocument, _
    Optional DisplayName As String, _
    Optional Action As MsoFileNewAction = msoEditFile, _
    Optional Refresh As Boolean = True)

    Dim FileStart As Long

    With Application
        'Falls kein Eintragtext festgelegt wurde,
        'ihn ermitteln (der Dateiname)
        If Len(DisplayName) = 0 Then
            FileStart = InStrRev(Filename, .PathSeparator)
            If FileStart = 0 Then
                DisplayName = Filename
            Else
                DisplayName = Mid$(Filename, FileStart + 1)
            End If
        End If

        'Den Eintrag entfernen
        .NewDocument.Remove Filename, FileSection, DisplayName, Action

        If Refresh Then
            With .CommandBars("Task Pane")
                .Visible = False
                .Visible = True
            End With
        End If
    End With
End Sub

```

## Erste Schritte

Um die Liste der zuletzt geöffneten Dateien auszuschalten (der oberste Abschnitt in Abbildung 17.7) oder die Länge der Liste zu beeinflussen, ändern Sie die gleichnamige Einstellung:

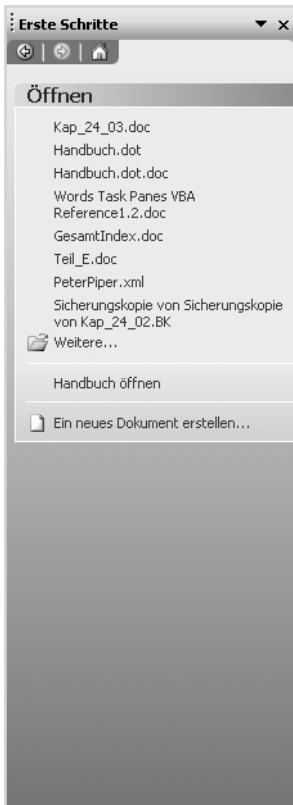
```
Application.DisplayRecentFiles = True
Application.RecentFiles = 5
'Entsprechen dem Kontrollkästchen und der Liste in Extras/Optionen/Allgemein
```

Auch der Aufgabenbereich *Erste Schritte* ist mit dem `NewDocument`-Objekt und seine `Add`- und `Remove`-Methoden ansprechbar. Die Syntax ist die gleiche, nur ist der einzige zur Verfügung stehende Abschnitt `msoOpenDocument`. Diese Einträge erscheinen im mittleren Abschnitt des Aufgabenbereichs.

Das folgende Code-Segment zeigt, wie ein Eintrag in den Aufgabenbereich mit Hilfe des Wrappers in Listing 17.2 eingefügt wird.

```
Sub EintraginErsteSchritteEinfügen()
    NewDocumentAdd "C:\test\Dok4.doc", msoOpenDocument, "Dok 4", msoEditFile
End Sub
```

**Abbildg. 17.7** Der Eintrag »Handbuch öffnen« im Aufgabenbereich *Erste Schritte* wurde mit der `NewDocument.Add`-Methode hinzugefügt.





Die Beispieldatei *Bsp17\_03.doc* mit Prozeduren, die den Umgang mit den Aufgabenbereichen *Neues Dokument* und *Erste Schritte* veranschaulicht, finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap17*.

## Formatvorlagen und Formatierungen

Dieser Aufgabenbereich stellt für den durchschnittlichen Benutzer eine echte Verbesserung dar. Die für ein Dokument zur Verfügung stehenden Formatierungen können übersichtlich präsentiert werden. Die größte Sorge ist: Wie wird diese Liste gezähmt?

Zunächst im Aufgabenbereich befindet sich eine Dropdown-Liste *Anzeigen*. Sie enthält fünf Einträge, um verschiedene Zusammenstellungen von Formatvorlagen und im Dokument verwendeten Formatierungen anzuzeigen: *Verfügbare Formatierungen*, *Benutzte Formatierungen*, *Verfügbare Formatvorlagen* und *Alle Formatvorlagen*. Dazu kommt der letzte Eintrag *Benutzerdefiniert*, wie in Abbildung 17.8 abgebildet.

### TIPP

Um die Einträge für benutzte und verfügbare Formatierungen zu sehen, muss das Kontrollkästchen *Formatierung mitverfolgen* im Dialogfeld *Optionen* auf der Registerkarte *Bearbeiten* aktiviert sein. Sonst erscheinen nur Einträge für Formatvorlagen.

Abbildg. 17.8 Die Liste des Aufgabenbereichs *Formatvorlagen und Formatierungen* lässt sich anpassen.

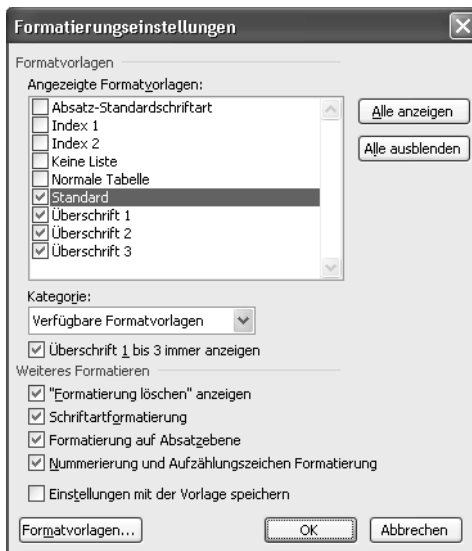


**HINWEIS**

Die Einträge mit »verfügbar« in der Bezeichnung beziehen sich auf die in Word 2003 eingeführte Möglichkeit, nur bestimmte Formatvorlagen und Formatierungen in einem Dokument freizugeben. Damit kann eine Firma oder eine Buchprojektleitung gewährleisten, dass bestimmte Dokumentarten immer gleich formatiert werden und einen Wildwuchs an Formatierungen unterbinden. In der Benutzerschnittstelle werden diese Optionen durch den Aufgabenbereich *Dokument schützen* festgelegt. Mehr zu diesem Thema finden Sie in Kapitel 6 im Abschnitt über Formatvorlagen.

Wie der Abbildung 17.9 zu entnehmen ist, wählt man eine Kategorie und passt den Inhalt an. Formatvorlagen können einzeln ausgeblendet werden. Zudem darf festgelegt werden, ob die Word-eigenen Überschriften 1 bis 3 immer aufzuführen sind, ob der Eintrag *Formatierung löschen* zuoberst in der Liste erscheint, und ob die Liste direkte Zeichen, Absatz und Listenformatierungen Formatierungen anzeigen soll.

**Abbildg. 17.9** Die Elemente, Formatierungen und Formatvorlagen nach Kategorie festlegen, die im Aufgabenbereich aufzulisten sind



Das Anpassen der Liste hört sich leicht an, ist aber in der Handhabung etwas komplex, vor allem wenn sie programmtechnisch erfolgen soll.

Die Tabelle 17.6 listet die benötigten Eigenschaften, um den Inhalt der Liste zu bestimmen. Demzufolge, wenn Sie den Eintrag *Formatierung löschen* aus der Liste verbannen, müsste es mit dieser Codezeile klappen:

```
ActiveDocument.FormattingShowClear = False
```

VBA führt sie auch ohne zu meckern aus, leider ist keine Wirkung ersichtlich. Würden Sie das Dialogfeld *Formatierungseinstellungen* öffnen, wäre das Kontrollkästchen tatsächlich deaktiviert. Am Kopf kratzend schließen Sie das Dialogfeld wieder und sehen mit Verwunderung, dass der Eintrag

nicht mehr aufgeführt ist. Das Geheimnis: Die Eigenschaften wirken sich auf das Dialogfeld und nicht direkt auf die Liste aus. Um die Einstellung in der Liste widerzuspiegeln, müssen Sie das Dialogfeld ausführen und bestätigen lassen:

```
SendKeys "{Enter}"
Dialogs(wdDialogFormatStylesCustom).Execute
```

Ziemlich unschön, weshalb es ratsam ist, auch hierfür eine Wrapper-Prozedur für die Festlegung der Eigenschaften aufzurufen.

**Tabelle 17.6** Eigenschaften, die das Aussehen des Aufgabenbereichs *Formatvorlagen und Formatierungen* beeinflussen

Dialogfeldeinstellung	Eigenschaft	Datentyp
"Formatierung löschen" anzeigen	Document.FormattingShowClear	Boolean
Kategorie	Document.FormattingShowFilter	WdShowFilter (siehe Tabelle 17.7)
Schriftartformatierung	Document.FormattingShowFont	Boolean
Nummerierung und Aufzählungszeichen Formatierung	Document.FormattingShowNumbering	Boolean
Formatierung auf Absatzebene	Document.FormattingShowParagraph	Boolean
Extra/Optionen/Bearbeiten	Options.FormatScanning	Boolean

**Tabelle 17.7** WdShowFilter-Werte und die entsprechende Option im Dialogfeld *Formatierungseinstellungen*

WdShowFilter-Enumeration	Wert	Entspricht der Option
wdShowFilterStylesAvailable	0	Verfügbare Formatvorlagen
wdShowFilterStylesInUse	1	Benutzte Formatvorlagen
wdShowFilterStylesAll	2	Alle Formatvorlagen
wdShowFilterFormattingInUse	3	Benutzte Formatierungen
wdShowFilterFormattingAvailable	4	Verfügbare Formatierungen

Soviel zum unteren Teil des Dialogfelds. Wie steht's mit der Auflistung der Formatvorlagen?

Das Style-Objekt hat eine verborgene Eigenschaft: *Visibility* (mehr über verborgene Elemente finden Sie in Kapitel 2). Vermutlich ist sie verborgen, weil sie sich nur im beschränkten Rahmen anwenden lässt. Damit lässt sich die Sichtbarkeit einen Formatvorlageneintrag im Aufgabenbereich festlegen. Kurioserweise wird der Eintrag aufgeführt, wenn *Visibility* = *False* ist, und wird unterdrückt, wenn *Visibility* = *True*.

Das Festlegen dieser Eigenschaft zeigt Wirkung nur bei einer neuen *Vorlage*. Wurde das Dialogfeld *Formatierungseinstellungen* nur einmal eingeblendet, gibt es über das Objektmodell keine Möglichkeit mehr, einzelne Formatvorlagen aus der Liste zu verbannen bzw. sie mit einzubeziehen. Das Listing 17.3 veranschaulicht die programmtechnische Voreinstellung des Aufgabenbereichs *Formatvorlagen und Formatierungen*.

**Listing 17.3** Den Inhalt des Aufgabenbereichs *Formatvorlagen und Formatierungen* festlegen

```
Public Sub FormatvorlagenListeAnpassen()
    Dim doc As Word.Document

    CommandBars("Task Pane").Visible = False
    Set doc = Application.Documents.Add(NewTemplate:=True)
    Application.CustomizationContext = doc

    'Nur die erwünschten Formatvorlagen anzeigen
    With doc.Styles
        .Item(wdStyleBodyText).Visibility = False
        .Item(wdStyleListBullet).Visibility = False
        .Item(wdStyleListBullet2) = False
        .Item(wdStyleHeading1).Visibility = False
        .Item(wdStyleHeading2).Visibility = True
        .Item(wdStyleHeading3).Visibility = True
    End With
    'Benutzerdefinierte Einstellungen festlegen
    doc.FormattingShowClear = False
    doc.FormattingShowFont = False
    doc.FormattingShowNumbering = False
    doc.FormattingShowParagraph = False
    Application.TaskPanes(wdTaskPaneFormatting).Visible = True
End Sub
```



Die Beispieldatei *Bsp17\_04.dot* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap17*.

## Zusammenfassung

Aufgabenbereiche stellen eine interessante Alternative zu traditionellen Symbolleisten und Menüs dar. Leider werden dem Entwickler nur minimale Schnittstellen für deren Automatisierung zur Verfügung gestellt. Dieses Kapitel veranschaulichte die Funktionalität und ihre Möglichkeiten. Unter anderem wurde Folgendes vorgestellt:

- Die dem Aufgabenbereich hinterlegende Technologie (Seite 684).
- Welche Schnittstellen das Word-Objektmodell für »Work Panes« zur Verfügung stellt (Seite 685).
- Registry-Einträge, mit denen das Verhalten von Aufgabenbereichen beeinflusst werden kann (Seite 686).
- Von Aufgabenbereichen verursachte Fehlermeldungen (687).
- Die programmtechnische Verwaltung von Aufgabenbereichen (Seite 688).
- Wie der Inhalt bestimmter Aufgabenbereiche (*Neues Dokument*, *Erste Schritte* sowie *Formatvorlagen und Formatierungen*) beeinflusst werden kann (Seite 698).

## Kapitel 18

# Interne Word-Befehle übersteuern

### In diesem Kapitel:

Word-Befehl außer Kraft setzen

708

Zusammenfassung

711

Bereits in Kapitel 1 wurde aufgezeigt, wie sich Word verhält, wenn mehrere Makros mit gleichem Namen aufeinander treffen. Welches der Makros dann aktiviert wird, ist durch eine klar definierte Hierarchie geregelt.

Dieser Hierarchie muss eigentlich noch eine weitere Ebene hinzugerechnet werden, nämlich die der »internen Word-Befehle«, deren Verwendungsmöglichkeiten das vorliegende Kapitel gewidmet ist.

## Word-Befehl außer Kraft setzen

Bei der Entwicklung von Word wurde darauf geachtet, dass Programmierer später über Makros und andere Erweiterungen in der Lage sein sollten, direkten Einfluss auf die ursprüngliche Funktionalität des Programms zu nehmen. Dazu gehört, dass alle Word-internen Befehle durch ein Makro außer Kraft gesetzt werden können.

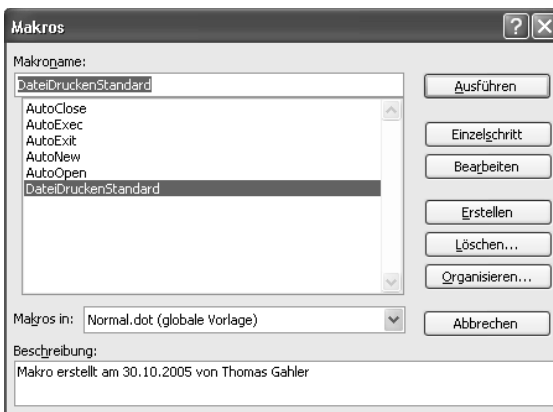
Soll ein einzelner Word-Befehl übersteuert werden, muss ein Makro (`Public Sub`, ohne Parameter) mit identischer Bezeichnung in einem VBA-Projekt angelegt werden. Dabei spielt es keine Rolle, ob dieses Makro in einem Dokument, der zugehörigen Dokumentvorlage, einem Add-In oder in der *Normal.dot* angelegt wird.

Sobald das Makro innerhalb von Word sichtbar, also in der Liste *Makroname* aufgeführt ist, steht die Originalfunktionalität des gleichnamigen Word-Befehls im entsprechenden Kontext nicht mehr zur Verfügung (vgl. Abbildung 18.1). Wird der einzelne Befehl mehrmals übersteuert, so gilt die bereits bekannte Reihenfolge.

- Ein Makro in der *Normal.dot* übersteuert den internen Word-Befehl.
- Ein Makro in einem Add-In übersteuert jedes aus der *Normal.dot*.
- Ein Makro in einer Dokumentvorlage übersteuert jenes in dem Add-In.
- Ein Makro in einem Dokument übersteuert jenes in der zugehörigen Dokumentvorlage.

Um dieses Verhalten nachzuvollziehen, speichern Sie die Beispieldatei *Bsp18\_01.dot* entweder im Vorlagenordner oder im *StartUp*-Ordner von Word. Je nach Speicherort stehen anschließend die drei Originalbefehle (`DateiDrucken`, `DateiDruckenStandard` und `DateiSpeichern`) nur den auf dieser Dokumentvorlage basierenden Dokumenten oder der ganzen Umgebung nicht mehr zur Verfügung.

**Abbildg. 18.1** Übersteuerter Befehl *DateiDruckenStandard* in der Liste der verfügbaren Makros





**WICHTIG** Wird ein interner Word-Befehl übersteuert, steht die ursprüngliche Funktionalität nicht mehr zur Verfügung. Wird diese, wie in Listing 18.1, weiterhin benötigt, muss innerhalb des entsprechenden Makros wieder eine Möglichkeit eingebaut werden.

**Listing 18.1** Übersteuerter Befehl *DateiSpeichern*, mit integrierter Warnung, wenn ein Dokument zu oft gespeichert wird

```
Sub DateiSpeichern()
    Dim dateGespeichert As Date

    If Not Len(ActiveDocument.Path) = 0 Then
        dateGespeichert = FileDateTime(ActiveDocument.FullName)
        If DateDiff("s", dateGespeichert, Now) < 60 Then
            MsgBox "Die Datei wurde vor weniger als 1 Minute das letzte Mal gespeichert." & _
                vbCrLf & "Ein erneutes Speichern ist noch nicht sinnvoll.", vbInformation
        Else
            ActiveDocument.Save
        End If
    End If
End Sub
```

Neben dem Beispiel aus Listing 18.1 sind andere, sinnvollere Anwendungen für das Übersteuern einzelner Word-Befehle denkbar. Beispielsweise könnte vor dem Ausdruck eines Dokuments geprüft werden, ob die Richtlinien des Corporate Designs eingehalten wurden: verwendete Schriftarten und Formatvorlagen, die Position des Logos usw. Eine weitere Anwendung könnte die formale Prüfung des eingegebenen Dateinamens auf die Einhaltung von Namenskonventionen übernehmen.

**HINWEIS** In vielen Fällen reicht es nicht aus, nur einen einzelnen Word-Befehl außer Kraft zu setzen, wie dies am Beispiel des Druckens deutlich wird:

Nebst dem Betätigen der Symbolleistenschaltfläche *Drucken* kann ein Ausdruck auch durch Wählen des Menübefehls *Datei/Drucken* gestartet werden. Abhängig vom Grund für die eigentliche Übersteuerung müssten wohl zwei Makros (*DateiDruckenStandard* und *DateiDrucken*) erzeugt werden, damit die gewünschte Funktionalität erreicht wird (vgl. Listing 18.2 und Listing 18.3).

**Listing 18.2** Übersteuerter Befehl *DateiDruckenStandard*, mit integrierter Überprüfung der Position des Logos

```
Sub DateiDruckenStandard()
    Dim doc As Word.Document

    Set doc = ActiveDocument
    If fktLogoPrüfen(doc) Then
        doc.PrintOut
    Else
        procFehlermeldung doc
    End If
End Sub
```

Die Funktion *fktLogoPrüfen* prüft, ob das Logo überhaupt vorhanden ist und ob es an der richtigen Position eingefügt wurde. Ist dies der Fall, wird das Dokument auf dem aktiven Drucker ausgegeben. Ansonsten wird der Ausdruck verweigert und der Anwender entsprechend informiert.

**Listing 18.3** Übersteuerter Befehl *DateiDrucken*, mit integrierter Überprüfung der Position des Logos

```

Sub DateiDrucken()
    Dim doc As Word.Document

    Set doc = ActiveDocument
    If fktLogoPrüfen(doc) Then
        Dialogs(wdDialogFilePrint).Show
    Else
        procFehlermeldung doc
    End If
End Sub

```

Mit dem zweiten Makro wird die zweite Funktion zum Drucken von Dokumenten übersteuert. Die hinterlegten Prüfungen entsprechen denen des ersten Makros. Sind sie erfolgreich, wird das Dialogfeld *Drucken* eingeblendet. Ansonsten wird der Aufruf des Dialogfelds ebenfalls verweigert und der Anwender entsprechend informiert.

---

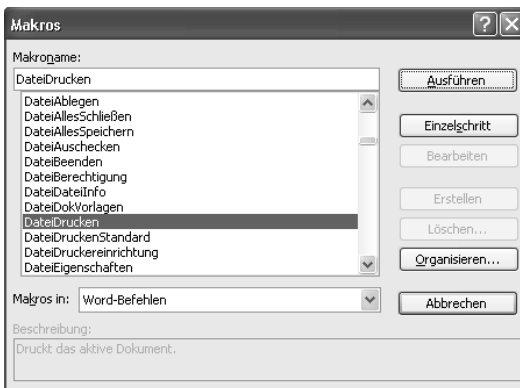
**HINWEIS** Der programmtechnische Umgang mit internen Dialogfeldern von Word wurde bereits in Kapitel 14 erläutert.

---

### Bezeichnung des internen Word-Befehls ermitteln

Die Bezeichnung eines internen Word-Befehls kann nicht mit dem Makrorekorder ermittelt werden. Dafür steht innerhalb von Word eine integrierte Liste zur Verfügung, die diese Bezeichnungen beinhaltet.

Um diese Liste einsehen zu können, wählen Sie den Menübefehl *Extras/Makro/Makros*. Aktivieren Sie im Listenfeld *Makros in* den Eintrag *Word-Befehlen*. Jetzt werden im Listenfeld *Makroname* alle internen Word-Befehle aufgelistet.

**Abbildg. 18.2** Eine Liste der integrierten Word-Befehle anzeigen lassen



---

**HINWEIS** Die Namen der einzelnen Befehle können in vielen Fällen vom Aufbau des Originalmenüs abgeleitet werden. So ist die Bezeichnung des internen Befehls, der durch den Menübefehl *Datei/Drucken* ausgeführt wird, *DateiDrucken*.

---

### Word-Befehle unabhängig von der eingesetzten Programmsprache übersteuern

In Abbildung 18.2 sind die Befehle in der installierten Sprache von Word aufgelistet. Diese Bezeichnungen sind nur der jeweiligen Programmsprache als interne Word-Befehle bekannt. In unserem Beispiel entspricht dies Deutsch.

Damit das Übersteuern von Befehlen unabhängig von der eingesetzten Programmsprache erfolgen kann, müssen die englischen Bezeichnungen der einzelnen Befehle verwendet werden. Einige Befehle und deren englische Bezeichnung sind in Tabelle 18.1 zusammengefasst.

**Tabelle 18.1** Word-Befehle und die zugehörige englische Bezeichnung

Word-Befehl, deutsch	Word-Befehl, englisch
DateiNeu	FileNew
DateiNeuStandard	FileNewDefault
DateiSpeichern	FileSave
DateiSpeichernUnter	FileSaveAs
DateiDrucken	FilePrint
DateiDruckenStandard	FilePrintDefault



Eine komplette Liste der deutschen Word-Befehle und ihrer englischen Benennung finden Sie in der Datei *Interne WordBefehle.doc* auf der CD-ROM zum Buch im Ordner `\Beilagen\Interne Word-Befehle`. Diese Datei wurde uns freundlicherweise von unserem MVP-Kollegen Klaus Linke zur Verfügung gestellt.

#### WICHTIG

Wird ein Word-Befehl auf Deutsch und auf Englisch übersteuert, so wird das Makro mit der englischen Bezeichnung abgearbeitet.



Die Prozeduren aus diesem Kapitel finden Sie in der Beispieldatei *Bsp18\_01.dot* auf der CD-ROM zum Buch im Ordner `\Beispiele\Kap18`.

## Zusammenfassung

In diesem Kapitel wurde aufgezeigt, wie interne Word-Befehle übersteuert und auf diese Art außer Kraft gesetzt werden bzw. auf eigene Bedürfnisse hin optimiert werden können:

- Es wurde besprochen, wo ein entsprechendes Makro abgespeichert werden kann und in welcher Priorität dieses den Word-Befehl übersteuert (Seite 708).
- Es wurde darauf hingewiesen, dass die ursprüngliche Funktionalität nicht mehr zur Verfügung steht, sofern diese nicht im eigentlichen Makro zusätzlich eingebaut wird (Seite 708).
- Schließlich wurde vermittelt, wie die Bezeichnung eines Befehls ermittelt und unabhängig von der eingesetzten Programmsprache verwendet werden kann (Seite 710).



## Kapitel 19

# Zugriff auf den Visual Basic-Editor (VBE)

### In diesem Kapitel:

Notwendige Verweise und Sicherheitseinstellungen	714
Der Visual Basic-Editor	715
Auslesen des VBA-Codes von Komponenten	724
Ersetzen und Entfernen von VBA-Code-Zeilen	731
Hinzufügen von Komponenten zu einem Projekt	734
Entfernen von Komponenten aus einem Projekt	745
Anzeigen von dynamisch erzeugten UserForms	746
Verweise auf Bibliotheken und Dateien ermitteln und zur Laufzeit setzen	749
Zusammenfassung	757

In Kapitel 1 »Word-Makros« und in Kapitel 2 »Der Visual Basic-Editor« wurde der Umgang mit dem Visual Basic-Editor und die Erstellung und Bearbeitung von Makros im Visual Basic-Editor behandelt.

In diesem Kapitel möchten wir Ihnen nun einen Überblick vermitteln, wie Sie aus einem Makro (einer Prozedur) heraus auf den Visual Basic-Editor und somit auf den VBA-Code (Visual Basic für Applikationen-Code), der in Dokumentvorlagen und Dokumenten enthalten sein kann, selbst zugreifen und diesen verändern können.

Nach einigen allgemeinen Begriffserklärungen und Informationen im Abschnitt »Der Visual Basic-Editor« über den Aufbau des Visual Basic-Editors und die Code-Darstellung darin, erhalten Sie im Abschnitt »Auslesen des VBA-Codes von Komponenten« Informationen, wie Sie auf die Benutzerformulare, Module, Prozeduren und ganz allgemein auf den VBA-Code in den einzelnen Dokumentvorlagen und Projekten lesend zugreifen können. Der Abschnitt »Ersetzen und Entfernen von VBA-Code-Zeilen« zeigt Ihnen anschließend, wie Sie Code-Zeilen ersetzen und auch entfernen können.

Im Abschnitt »Hinzufügen von Komponenten zu einem Projekt« erhalten Sie Informationen, wie Sie per VBA-Code dynamisch neue Benutzerformulare und Module erstellen und mit Ereignissen, Prozeduren und VBA-Code füllen können.

Der Abschnitt »Anzeigen von dynamisch erzeugten UserForms« zeigt Ihnen, wie sich die neu erstellten Benutzerformulare und Module aus einem Makro heraus anzeigen und starten lassen.

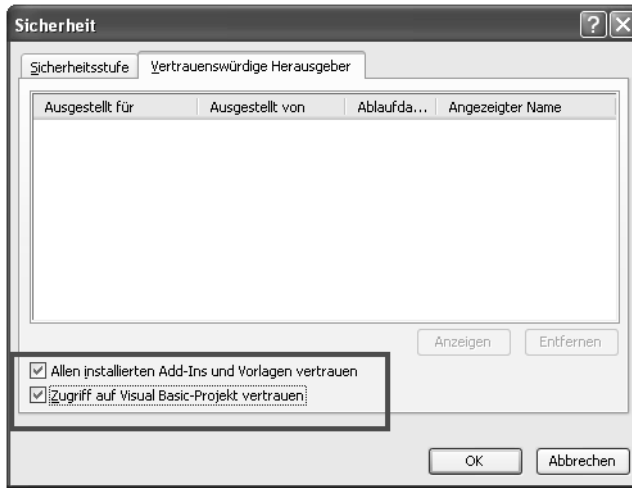
Im letzten Abschnitt »Verweise auf Bibliotheken und Dateien ermitteln und zur Laufzeit setzen« wird das Setzen, Prüfen und Entfernen von Verweisen im Visual Basic-Editor behandelt.

## Notwendige Verweise und Sicherheitseinstellungen

Bevor Sie per VBA-Code auf den Visual Basic-Editor zugreifen können, müssen Sie die Sicherheitsstufe anpassen. Dazu müssen Sie im Word-Menü *Extras* den Menübefehl *Makro/Sicherheit* aufrufen und in dem in Abbildung 19.1 angezeigten Dialogfeld zur Registerkarte *Vertrauenswürdige Herausgeber* wechseln. Über das Kontrollkästchen *Zugriff auf Visual Basic-Projekt vertrauen* legen Sie fest, ob eigene und fremde Anwendungen oder Makros Zugriff auf die Visual Basic-Projekte erhalten dürfen oder nicht. Markieren Sie dieses Kontrollkästchen, damit die Zugriffe auf den Visual Basic-Editor und die Visual Basic-Projekte nicht mit der Fehlermeldung »Dem programmatischen Zugriff auf das Visual Basic-Projekt wird nicht vertraut« verweigert werden.

Wenn der generelle Zugriff auf den Visual Basic-Editor gewährt wird, benötigen Sie zusätzlich einen Verweis auf die Bibliothek *Microsoft Visual Basic for Applications Extensibility 5.3*, in der alle notwendigen Befehle, Eigenschaften und Methoden, die für den Zugriff benötigt werden, enthalten sind (weitere Informationen zum Setzen von Verweisen finden Sie in Kapitel 8).

Abbildg. 19.1 Sicherheitsstufe anpassen: Zugriff auf Visual Basic-Projekt vertrauen

**WICHTIG**

Überprüfen Sie für jedes Projekt, aus dem heraus der Zugriff auf den Visual Basic-Editor erfolgen soll, ob der benötigte Verweis auf die Bibliothek *Microsoft Visual Basic for Applications Extensibility 5.3* gesetzt ist.

Dazu müssen Sie im Visual Basic-Editor im Menü *Extras/Verweise* den obigen Eintrag in der Liste der registrierten Bibliotheken auswählen und markieren, sodass er mit einem Häkchen versehen angezeigt wird.

Ohne diesen Verweis stehen Ihnen die notwendigen Zugriffs-Eigenschaften und Methoden nicht zur Verfügung, und jeder Zugriff auf eine entsprechende Eigenschaft wird mit der Fehlermeldung *Benutzerdefinierter Typ nicht definiert* verweigert!

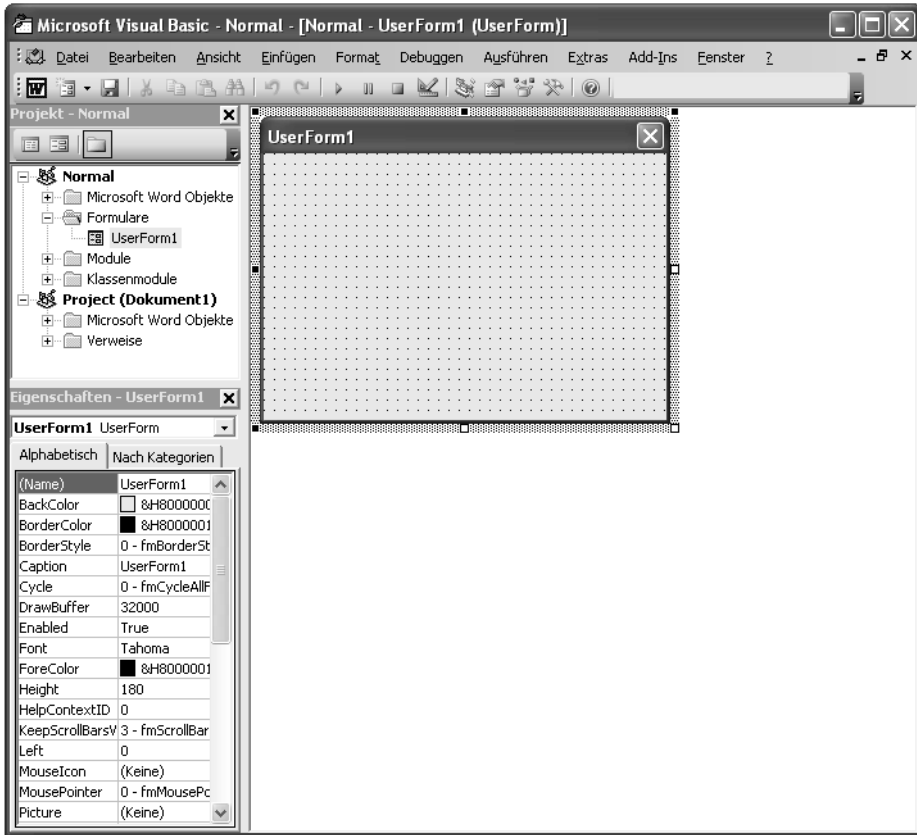
## Der Visual Basic-Editor

Der Visual Basic-Editor (Abbildung 19.2) ist die Umgebung, in der Sie neuen VBA-Code und neue VBA-Prozeduren schreiben bzw. vorhandenen VBA-Code und vorhandene Verfahren bearbeiten.

Der Visual Basic-Editor unterteilt sich grob in folgende Bereiche:

- **Code- bzw. UserForm-Fenster**  
Eingabebereich zum Erstellen, Anzeigen und Bearbeiten von VBA-Code bzw. zum grafischen Erstellen und Bearbeiten von Fenstern oder Dialogfeldern.
- **Projekt-Explorer**  
Zeigt eine hierarchische Liste der Projekte und aller Elemente an, die in den jeweiligen Projekten enthalten sind bzw. von den jeweiligen Projekten referenziert werden.
- **Eigenschaftenfenster**  
Zeigt die Entwurfszeiteigenschaften für ein ausgewähltes Objekt und deren aktuelle Einstellungen an.

Abbildg. 19.2 Übersicht über den Visual Basic-Editor mit UserForm-Fenster, Projekt-Explorer und Eigenschaftfenster



#### HINWEIS

Wird ein Fenster nicht angezeigt, können Sie ihn im Menü *Ansicht* über den entsprechenden Menübefehl *Code* (F7) bzw. *Objekt* (UserForm/Benutzerformular, Alt + F7), *Projekt-Explorer* (Strg + R) und *Eigenschaftfenster* (F4) einblenden (weitere Hinweise zum Umgang mit dem Visual Basic-Editor finden Sie im Kapitel 2).

Im Objektmodell von Microsoft Word ist der Visual Basic-Editor (VBE) direkt in der Hauptebene unter dem Application-Objekt, das auf oberster Ebene die Microsoft Word-Anwendung darstellt, angeordnet. Die VBE-Eigenschaft des Application-Objektes liefert ein VBE-Objekt zurück, das den Visual Basic-Editor darstellt.

Innerhalb des Visual Basic-Editors erfolgt der Zugriff auf dieses VBE-Objekt über den Aufruf:

```
Application.VBE
```



**TIPP**

Alle Eigenschaften der Hauptebene unter dem Application-Objekt können auch ohne die Angabe des übergeordneten Application-Objektes dargestellt werden.

Wenn Sie aber häufiger VBA-Code in andere Sprachen oder Umgebungen portieren, sollten Sie die vollständige Objekt-Hierarchie verwenden, damit das Application-Objekt korrekt mitportiert wird.

Über die Eigenschaften des VBE-Objektes erhalten Sie neben dem Zugriff auf die Fenster des Editors auch Zugriff auf die VBA-Projekte. Die Eigenschaften werden automatisch eingeblendet, sobald Sie hinter dem »VBE« einen Punkt eingeben (Abbildung 19.3).

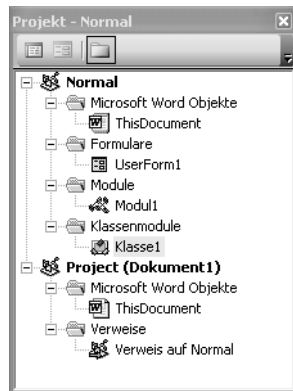
Abbildg. 19.3 Eigenschaft des VBE-Objektes



## Die VBA-Projekte

Im Projekt-Explorer des Visual Basic-Editors (Abbildung 19.4) werden alle geladenen Projekte angezeigt. Dabei wird für jedes Dokument, jede Dokumentvorlage und jedes Add-In, das zu dem Zeitpunkt geladen ist, ein Eintrag im Projekt-Explorer angezeigt.

Abbildg. 19.4 Übersicht über den Projekt-Explorer mit allen Projektkomponenten



Sofern diese Projekte nicht als Add-Ins mit Word geladen sind (das können Sie nach Aufruf des Menübefehls *Extras/Vorlagen und Add-Ins* im zugehörigen Dialogfeld anhand der markierten Einträge überprüfen), automatisch aus dem *StartUp*-Ordner mitgeladen werden oder per Kennwort geschützt sind, können Sie per VBA auf das Projekt zugreifen.

**HINWEIS** Microsoft Word 2003 kennt zwei *StartUp*-Ordner:

- Einen im Programmordner von Microsoft Office 2003  
C:\Programme\Microsoft Office2003\OFFICE11\STARTUP
  - Einen benutzerspezifischen für jeden Anwender  
C:\Dokumente und Einstellungen\<Benutzername>\Anwendungsdaten\Microsoft\Word\StartUp
- sofern Microsoft Office 2003 im vorgeschlagenen Standardverzeichnis auf dem Laufwerk C: installiert wurde.

## Übersicht über alle VBA-Projekte

Die VBProjects-Eigenschaft des Visual Basic-Editors liefert eine Auflistung aller geladenen Projekte, wie sie im Projekt-Explorer angezeigt werden.

Da ein Zugriff nur auf Projekte möglich ist, die nicht als Add-In geladen oder mittels Kennwort geschützt sind, muss vor jedem Zugriff auf die Projekte die Protection-Eigenschaft geprüft werden. Andernfalls würde später ein Zugriff auf den VBA-Code in diesen Projekten mit einer Fehlermeldung verweigert.



Die in diesem Abschnitt verwendeten Beispiel-Prozeduren finden Sie auf der CD-ROM in der Datei \Beispiele\Kap19\Bsp19\_01.doc im Modul *modProjektNamen*.

Das Beispiel in Listing 19.1 zeigt für jedes Projekt neben dem Namen auch die Zugriffsmöglichkeit an:

**Listing 19.1** Zeigt für Projekte an, ob sie gesperrt sind oder ein Zugriff möglich ist

```
Sub subAlleVBProjekte()
    Dim strVBP As String
    Dim vbp As VBProject
    For Each vbp In VBE.VBProjects
        strVBP = strVBP & vbp.Name & IIf(vbp.Protection = vbext_pp_locked, _
            " - gesperrtes Projekt", " - ungesperrtes Projekt") & vbCrLf
    Next vbp
    MsgBox strVBP, vbInformation, "Zugriffsübersicht über die VBProjekte"
End Sub
```

Standardmäßig liefert die Name-Eigenschaft den Namen des Projektes zurück, der im Visual Basic-Editor in der Projekteigenschaft *Name* eingetragen ist. Bei neuen Projekten (unabhängig davon, ob das Projekt ein Dokument oder eine Dokumentvorlage darstellt) ist dies standardmäßig der Eintrag *Project*.

Sie können nun für alle nicht gesperrten Projekte (gespeicherte und nicht gespeicherte) den Projekt-namen über die Name-Eigenschaft ändern und so eine aussagekräftigere Bezeichnung festlegen. Der Projektname wird allerdings **nicht** als Vorschlag für den Dateinamen verwendet, wenn die Datei oder die Dokumentvorlage gespeichert wird.

**ACHTUNG** Die einzige Möglichkeit, mehr Informationen über unbenannte Projekte zu erfahren, z.B. den Pfad der Projektdatei, besteht darin, den Dateinamen, der beim Export der Komponente von Microsoft Word über die `FileName`-Eigenschaft erstellt wird, auszuwerten.

Allerdings erfolgt die Ausgabe des Projektnamens und Projektpfades nur für gespeicherte Dokumente.

Wenn Sie den Befehl aus einem noch nicht gespeicherten Dokument oder einer Dokumentvorlage aufrufen, erhalten Sie die Fehlermeldung: »Laufzeitfehler '76': Pfad nicht gefunden«.

Die `FileName`-Eigenschaft liefert den vollständigen Namen inkl. Pfad des aktuellen Projektes und somit auch den Typ (Dokument oder Dokumentvorlage, erkennbar an der Dateieindung) wie in Listing 19.2 veranschaulicht.

**Listing 19.2** Ausgabe des vollständigen Projekt-Namens aller geladenen Projekte

```
Sub subAlleVBProjektNamen()
    Dim strVBP As String
    Dim vbp As VBProject
    For Each vbp In VBE.VBProjects
        On Error Resume Next
        If vbp.FileName = "" Then
            strVBP = strVBP & vbp.Name & ": " & vbTab & "nicht gespeicherte Datei" & vbCrLf
        Else
            strVBP = strVBP & vbp.Name & ": " & vbTab & vbp.FileName & vbCrLf
        End If
    Next vbp
    MsgBox strVBP, vbInformation, "Projektnamen"
    On Error GoTo 0
End Sub
```

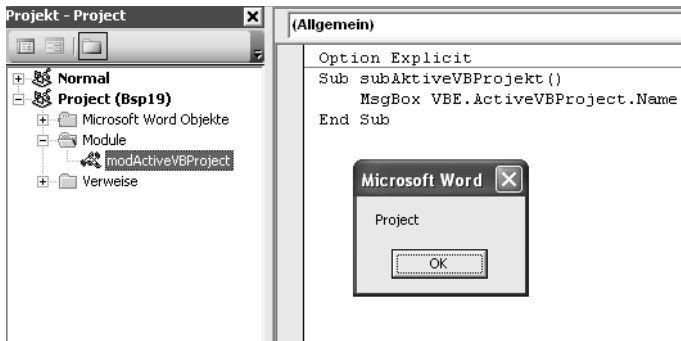
## Das aktive VBA-Projekt

Die `ActiveVBProject`-Eigenschaft des Visual Basic-Editors stellt eine Besonderheit innerhalb der VBA-Projekte dar. Diese Eigenschaft liefert das im Projekt-Explorer ausgewählte Projekt bzw. jenes Projekt, von dem aus diese Eigenschaft aufgerufen wird, zurück.

Den Namen des aktuellen Projektes erhalten Sie, wie in Abbildung 19.5 ersichtlich, wieder über die `Name`-Eigenschaft des `ActiveVBProject`-Objektes:

```
Sub subAktiveVBProjektName()
    MsgBox VBE.ActiveVBProject.Name
End Sub
```

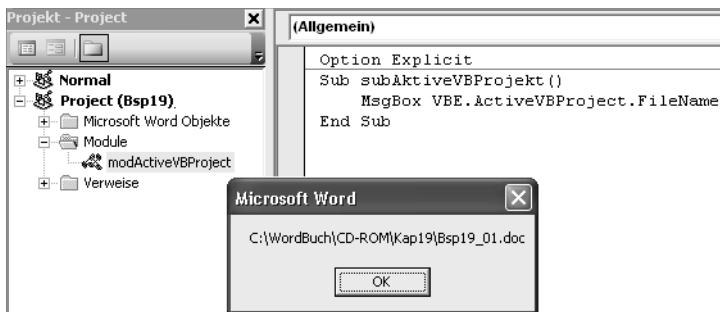
Abbildg. 19.5 Anzeige des aktuellen Projekt-Namens



Den Pfad des aktuellen Projektes erhalten Sie wieder über die FileName-Eigenschaften, sofern das Projekt gespeichert wurde (Abbildung 19.6):

```
Sub subAktiveVBProjekt()  
    MsgBox VBE.ActiveVBProject.FileName  
End Sub
```

Abbildg. 19.6 Anzeige des vollständigen Projektpfades



#### TIPP

Eine weitere Möglichkeit, den Pfad und Dateinamen des aktuellen Projektes zu erhalten, besteht über die MacroContainer-Eigenschaft des Application-Objektes.

Diese Eigenschaft gibt ein Template- oder Document-Objekt zurück, das die Dokumentvorlage oder das Dokument darstellt (das Container-Objekt), in dem die aufrufende Prozedur enthalten ist.

Leider werden für diese Eigenschaft die zur Verfügung stehenden Methoden nicht angezeigt, wenn man nach dem Namen einen Punkt eingibt. Die wichtigsten Methoden der MacroContainer-Eigenschaft sind:

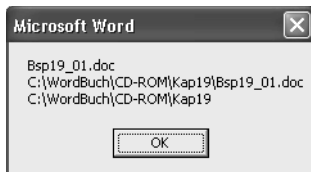
- Name  
Gibt den Namen des Dokumentes oder der Dokumentvorlage, in der die aufrufende Prozedur enthalten ist, zurück;

- **FullName**  
Gibt den vollständigen Namen inkl. Pfad des Dokumentes oder der Dokumentvorlage, in der die aufrufende Prozedur enthalten ist, zurück;
- **Path**  
Gibt den Pfad des Dokumentes oder der Dokumentvorlage, in der die aufrufende Prozedur enthalten ist, zurück.

Die MacroContainer-Eigenschaft eignet sich immer dann, wenn auf das Container-Objekt, in der der VBA-Code enthalten ist, Bezug genommen werden soll, um beispielsweise eine Datei im selben Verzeichnis zu speichern (Abbildung 19.7):

```
Sub subMacroContainerName()
    Dim strMC As String
    strMC = strMC & Application.MacroContainer.Name & vbCrLf
    strMC = strMC & Application.MacroContainer.FullName & vbCrLf
    strMC = strMC & Application.MacroContainer.Path & vbCrLf
    MsgBox strMC
End Sub
```

Abbildg. 19.7 Ermittlung der Projektinformationen über die *MacroContainer*-Eigenschaft



## Zugriff auf die in einem Projekt enthaltenen Komponenten



Die in diesem Abschnitt verwendeten Beispiel-Prozeduren finden Sie auf der CD-ROM in der Datei *\Beispiele\Kap19\Bsp19\_01.doc* im Modul *modActiveVBProject*.

Jedes VBProject-Objekt umfasst eine Reihe von Komponenten: Die VBComponents-Objekte.

Zu diesen Komponenten gehören alle im Projekt-Explorer unterhalb des Projektes angezeigten Elemente, wie in Abbildung 19.4 ersichtlich:

- **Microsoft Word Objekte**  
Das mit dem Projekt verbundene Dokument. In Microsoft Word wird dieses Objekt immer durch *ThisDocument* dargestellt und repräsentiert das aktuelle Dokument oder Dokumentvorlage.
- **Formulare (UserForms)**  
Alle im Projekt enthaltenen Benutzerformulare (UserForms).
- **Module (Sammlung von Prozeduren)**

Alle im Projekt enthaltenen Module mit den Prozeduren (Makros), wie sie auch über den Menübefehl *Extras/Makro/Makros* aufgeführt werden. Ein Modul kann mehrere Prozeduren enthalten.

■ **Klassenmodule**

Alle im Projekt enthaltenen Klassenmodule.

■ **Verweise auf andere Projekte**

Alle Verweise auf andere Projekte, die im Visual Basic-Editor unter *Extras/Verweise* markiert sind. Hierzu zählen aber nur Verweise auf andere Dokumente und Dokumentvorlagen, nicht auf Systembibliotheken.

Die Anzahl aller VBComponents-Elemente eines Projektes erhalten Sie mit der Count-Eigenschaft. Durchlaufen können Sie die Auflistung der Komponenten dann entweder mit einer For...Next-Schleife oder mittels einer For Each...Next-Anweisung.

**TIPP**

Um eine Komponente direkt anzusprechen, können Sie entweder die Index-Nummer oder den Namen der Komponente angeben. Es ist jedoch einfacher, die Komponenten über den Namen anzusprechen, da der Index in der Reihenfolge des Anlegens/Hinzufügens von Komponenten vergeben wird, während sie im Projekt-Explorer alphabetisch sortiert aufgelistet werden.

Wenn Sie eine Komponente über den Index ansprechen möchten, sollten Sie unbedingt erst den zurückgelieferten Namen überprüfen, bevor Sie auf die Komponenten zugreifen.

Den jeweiligen Typ dieser VBComponents-Komponenten können Sie über die Type-Eigenschaft ermitteln; als Rückgabe erhalten Sie einen der numerischen Werte in Tabelle 19.1.

**Tabelle 19.1** Mögliche Werte der Type-Eigenschaft einer VB-Komponente

VBIDE.vbext_Component-Konstantwert	Wert	Beschreibung
vbext_ct_ClassModule	2	Klassenmodul
vbext_ct_Document	100	<i>ThisDocument</i> -Klassenmodul
vbext_ct_MSForm	3	UserForm
vbext_ct_StdModule	1	Standardmodul
vbext_ct_ActiveXDesigner	11	ActiveX Designer

**Listing 19.3** Auflistung aller in einem Projekt enthaltenen VBComponents mit Typangabe

```
Sub subListVBComponentsNamen()
    Dim vbp As VBProject
    Dim vbc As VBComponent
    Dim strVBC As String
    Dim strTyp As String
    Set vbp = VBE.ActiveVBProject
    For Each vbc In vbp.VBComponents
        With vbc
            Select Case .Type
            Case 100
                strTyp = "Word Objekt: "
```

**Listing 19.3** Auflistung aller in einem Projekt enthaltenen *VBComponents* mit Typangabe (Fortsetzung)

```

Case 1
    strTyp = "Modul:  "
Case 2
    strTyp = "Klassenmodul: "
Case 3
    strTyp = "UserForm: "
End Select
strVBC = strVBC & strTyp & vbTab & vbc.Name & vbCrLf
End With
Next vbc
MsgBox strVBC, vbInformation, "VBComponents"
Set vbp = Nothing
End Sub

```

Als Ergebnis von Listing 19.3 werden alle im aktiven VBProject enthaltenen Komponenten mit Typ-Aufschlüsselung und ihrem Namen angezeigt, wie in Abbildung 19.8 ersichtlich.

**Abbildg. 19.8** Ausgabe aller *VBComponents*-Einträge eines Projekts

Der VBA-Code einer jeden Komponente wird im Code-Fenster angezeigt, dort eingegeben oder geändert. Dieses Code-Fenster wird durch das *CodeModule*-Objekt dargestellt, das den gesamten VBA-Code der Komponente umfasst.

Über die Eigenschaften des *CodeModule*-Objektes können Sie VBA-Code hinzufügen, ändern, bearbeiten und löschen.

Die Tabelle 19.2 listet die Methoden auf, die zum Erstellen, Bearbeiten und Löschen von VBA-Code im Code-Modul zur Verfügung stehen. Dabei lassen sich die Methoden grob in zwei Gruppen unterteilen:

- Methoden zum Lesen und Ermitteln von Zeilen
- Methoden zum Erstellen und Ändern von Zeilen

Diese beiden Gruppen werden in den folgenden beiden Abschnitten behandelt.

# Auslesen des VBA-Codes von Komponenten

Über die Methoden in Tabelle 19.2 erhalten Sie Zugriff auf die Zeilen eines Code-Moduls.

**Tabelle 19.2** Übersicht über die Methoden zum Zugriff auf ein Code-Modul

Zugriffsmethode	Beschreibung
CountOfDeclarationLines	Gibt die Anzahl der Code-Zeilen im Deklarationsabschnitt eines Code-Moduls zurück (siehe den Abschnitt »Zugriff auf den Deklarationsbereich«).
CountOfLines	Gibt die Gesamtzeilenzahl des Code-Moduls an (siehe den Abschnitt »Zugriff auf die Code-Zeilen einzelner Prozeduren«).
Find	Gibt an, ob in einem Code-Modul ein angegebener Text enthalten ist (siehe den Abschnitt »Auflisten und Durchsuchen aller Projekte«).
Lines	Gibt eine angegebene Anzahl von Zeilen aus dem Code-Modul zurück (siehe den Abschnitt »Zugriff auf die Code-Zeilen einzelner Prozeduren«).
ProcBodyLine	Gibt die erste Zeile einer Prozedur zurück, in der die <b>Sub</b> -, <b>Function</b> - oder <b>Property</b> -Anweisung enthalten ist (siehe den Abschnitt »Zugriff auf die Code-Zeilen einzelner Prozeduren«).
ProcCountLines	Gibt die Anzahl der Zeilen in der festgelegten Prozedur zurück (siehe den Abschnitt »Zugriff auf die Code-Zeilen einzelner Prozeduren«).
ProcOfLine	Gibt den Namen der Prozedur zurück, in der sich die festgelegte Zeile befindet (siehe den Abschnitt »Zugriff auf die Code-Zeilen einzelner Prozeduren«).
ProcStartLine	Gibt die Zeile zurück, an der die festgelegte Prozedur beginnt (siehe den Abschnitt »Zugriff auf die Code-Zeilen einzelner Prozeduren«).

Diese Methoden lassen sich wieder grob in zwei Gruppen unterteilen:

- Methoden zum allgemeinen Zugriff auf die Code-Zeilen
- Methoden zum Zugriff auf die Prozeduren



Die in diesem Abschnitt verwendeten Beispiel-Prozeduren finden Sie auf der CD-ROM in der Datei `\Beispiele\Kap19\Bsp19_01.doc` im Modul `modReadVBComponents`.

## Allgemeine Zugriffe auf die Code-Zeilen

Über die beiden Eigenschaften `CountOfLines` und `Lines` erhalten Sie Zugriff auf alle Code-Zeilen im Code-Modul. Die erste Methode `CountOfLines` liefert dabei die Zahl aller Zeilen, auch Leerzeilen am Anfang und Ende, wieder, während die Methode `Lines` eine oder mehrere Zeilen aus dem Code-Modul zurückliefert.

**Listing 19.4** Ausgabe des gesamten VBA-Codes eines Code-Moduls

```
Sub subListVBComponentCode()
    Dim strVBC As String
    Dim strTyp As String
    Set vbp = VBE.ActiveVBProject
```



Listing 19.4 Ausgabe des gesamten VBA-Codes eines Code-Moduls (Fortsetzung)

```

Set vbc = vbp.VBComponents("modReadVBComponents")
With vbc.CodeModule
    MsgBox .Lines(1, .CountOfLines)
End With
End Sub

```

Als Ergebnis erhalten Sie die Übersicht über den VBA-Code in Abbildung 19.9.

**WICHTIG**

Da die MsgBox-Funktion nur Zeichenfolgenlängen von 1024 Zeichen ausgeben kann, wird in den Fällen, wo diese Grenze überschritten wird, die Ausgabe abgebrochen. Bei umfangreicheren Code-Modulen kann es daher passieren, dass nicht alle Code-Zeilen angezeigt werden.

Abbildg. 19.9 Anzeige des VBA-Codes eines Code-Moduls



## Zugriff auf den Deklarationsbereich

Haben Sie im Deklarationsbereich (das ist der gesamte Bereich vor der ersten Prozedur) einige globale Variablen deklariert, können Sie über die CountOfDeclarationLines-Eigenschaft nur auf diese Zeilen zugreifen und z.B. auslesen, wie Listing 19.5 veranschaulicht.

Listing 19.5 Anzeigen des Deklarationsbereiches

```

Sub subListVBComponentCodeDec()
    Dim strVBC As String
    Dim strTyp As String
    Set vbp = VBE.ActiveVBProject
    Set vbc = vbp.VBComponents("modReadVBComponents")
    With vbc.CodeModule
        MsgBox .Lines(1, .CountOfDeclarationLines)
    End With
End Sub

```

Als Ergebnis werden in diesem Beispiel nur die ersten vier Zeilen (somit auch die Leerzeile) aus der Abbildung 19.9 in Abbildung 19.10 angezeigt.

**Abbildg. 19.10** Anzeige der Zeilen des Deklarationsbereiches eines Code-Moduls



## Zugriff auf die Code-Zeilen einzelner Prozeduren

Innerhalb einer Komponente befinden sich neben allgemeinen Deklarationen häufig auch mehrere Prozeduren: z.B. in Abbildung 19.9 die beiden Prozeduren `subListVBComponentCode` und `subListVBComponentCodeDec`.

Die Prozeduren selbst lassen sich nur dann ansprechen, wenn die Prozedurnamen bekannt sind. Leider steht keine Methode oder Eigenschaft zur Verfügung, die diese Informationen direkt liefert. Sie können jedoch die Eigenschaft `ProcOfLine` verwenden, die den Prozedurnamen der aktuellen Zeile zurückliefert. Somit kann man mit einer Schleife über alle Zeilen eines Code-Moduls für jede Zeile prüfen, ob diese zu einer Prozedur gehört und, wenn dies der Fall ist, wie der Name der jeweiligen Prozedur lautet.

Das Beispiel in Listing 19.5 durchläuft zeilenweise den Code der Komponente *modReadVBComponents* und liefert nur die Prozedurnamen zurück. Das Ergebnis ist in Abbildung 19.11 ersichtlich.

**Listing 19.6** Ausgabe der Prozedurnamen einer Komponente

```
Sub subListVBComponentCodeProc()
    Dim strVBC As String
    Dim strTyp As String
    Dim intLine As Integer
    Dim strProc As String, strOldProc As String
    Dim strMsg As String
    Set vbp = VBE.ActiveVBProject
    Set vbc = vbp.VBComponents("modReadVBComponents")
    With vbc.CodeModule
        For intLine = 1 To .CountOfLines
            strProc = .ProcOfLine(intLine, vbext_pk_Proc)
            If strProc <> strOldProc Then
                strOldProc = strProc
                strMsg = strMsg & strProc & vbCrLf
            End If
        Next intLine
    End With
    MsgBox strMsg
End Sub
```

Abbildg. 19.11 Ermittlung und Ausgabe der Prozedurnamen in einer Komponente



Mit diesen Informationen können Sie nun zusammen mit den Eigenschaften `ProcBodyLine` und `ProcCountLines` den Code-Bereich einer Prozedur ermitteln.

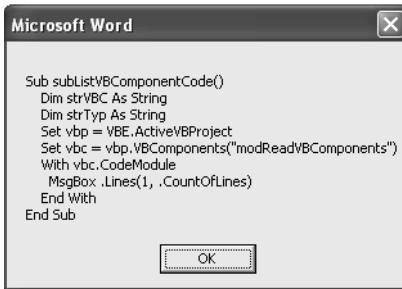
**ACHTUNG** Die `ProcBodyLine`-Eigenschaft liefert die erste Zeile einer Prozedur zurück, in der die Sub-, Function- oder Property-Anweisung enthalten ist. Allerdings können vor der Anweisung auch noch Leerzeilen und Kommentarzeilen stehen, die ebenfalls zu dieser Prozedur gehören.

Aus diesem Grund würde in diesen Fällen die `ProcBodyLine`-Eigenschaft nicht die tatsächliche Anfangszeile der Prozedur zurückliefern. Verwenden Sie daher die `ProcStartLine`-Eigenschaft, um die erste Zeile der ermittelten Prozedur zu erhalten.

Die Verwendung dieser Eigenschaften veranschaulicht Listing 19.7; das Ergebnis befindet sich in Abbildung 19.12.

Listing 19.7 Ausgabe des VBA-Codes für einzelne Prozeduren

```
Sub subListVBComponentCodeProcLines()
    Dim strVBC As String
    Dim strTyp As String
    Dim intLine As Integer
    Dim strProc As String, strOldProc As String
    Dim strMsg As String
    Dim intProcStart As Integer, intProcLines As Integer
    Set vbp = VBE.ActiveVBProject
    Set vbc = vbp.VBComponents("modReadVBComponents")
    With vbc.CodeModule
        For intLine = 1 To .CountOfLines
            strProc = .ProcOfLine(intLine, vbext_pk_Proc)
            If strProc <> strOldProc Then
                strOldProc = strProc
                intProcStart = .ProcStartLine(strProc, vbext_pk_Proc)
                MsgBox .Lines(intProcStart, .ProcCountLines(strProc, vbext_pk_Proc))
                strMsg = strMsg & strProc & vbCrLf
            End If
        Next intLine
    End With
End Sub
```

**Abbildg. 19.12** Prozedurweise Ausgabe des VBA-Codes


Für den direkten Zugriff auf die Prozeduren in Modulen und Benutzerformularen wird als Prozedurart ProzArt die Konstante vbext\_pk\_Proc verwendet. Bei Eigenschaftenprozeduren muss hingegen erst die Art der Prozedur geprüft werden (siehe nachfolgender Kasten).

### Eigenschaftenprozeduren (*Property-Prozeduren*)

Bei diesem direkten Zugriff auf den Prozedur-Code wird eine Besonderheit deutlich, die im Abschnitt »Zugriff auf die in einem Projekt enthaltenen Komponenten« in diesem Kapitel angesprochen wurde: Die unterschiedlichen Typen von Komponenten, besonders dabei die besondere Stellung der Klassenmodule.

So werden in Benutzerformularen und Modulen meistens nur die Sub- und Function-Prozeduren verwendet, während in Klassenmodulen überwiegend Eigenschaftenprozeduren (Property-Prozeduren) verwendet werden. Diese Eigenschaftenprozeduren können mehrere Darstellungen im Modul haben, je nachdem ob Werte festgelegt oder zurückgegeben werden (Property Get, Property Let und Property Set).

Daher müssen Sie beim Zugriff auf eine solche Eigenschaftenprozedur prüfen, von welcher Art die jeweilige Eigenschaftenprozedur ist. Die folgenden Konstanten repräsentieren die verschiedenen Eigenschaftenprozeduren und müssen korrekt beim Zugriff auf eine Prozedur angewendet werden:

- vbext\_pk\_Get
- vbext\_pk\_Let
- vbext\_pk\_Set

So schlägt der Zugriff auf eine Property Get-Prozedur fehl, wenn sie als Zugriffskonstante vbext\_pk\_Let verwenden. Zur Prüfung der Prozeduren muss in diesem Fall entweder der Inhalt zeilenweise ausgelesen und geprüft werden, oder Sie werten den zurückgegebenen Fehler aus.

Das folgende Beispiel Listing 19.8 überprüft zeilenweise auf die Prozedurtypen und liest dann über die passende Konstante die Prozedur aus.

**Listing 19.8** Zugriffsermittlung für Eigenschaftenprozeduren und Ausgabe der Prozeduren (*Fortsetzung*)

```

With vbc.CodeModule
    For intLine = 1 To .CountOfLines
        strProc = .Lines(intLine, 1)
        If InStr(1, strProc, "Property Get") > 0 Then
            strProc = .ProcOfLine(intLine, vbext_pk_Proc)
            intProcStart = .ProcStartLine(strProc, vbext_pk_Get)
            intProcLines = .ProcCountLines(strProc, vbext_pk_Get)
            MsgBox .Lines(intProcStart, intProcLines)
        ElseIf InStr(1, strProc, "Property Let") > 0 Then
            strProc = .ProcOfLine(intLine, vbext_pk_Let)
            intProcStart = .ProcStartLine(strProc, vbext_pk_Let)
            MsgBox .Lines(intProcStart, .ProcCountLines(strProc, vbext_pk_Let))
        ElseIf InStr(1, strProc, "Property Set") > 0 Then
            strProc = .ProcOfLine(intLine, vbext_pk_Set)
            intProcStart = .ProcStartLine(strProc, vbext_pk_Set)
            MsgBox .Lines(intProcStart, .ProcCountLines(strProc, vbext_pk_Set))
        End If
    Next intLine
End With

```

## Auflisten und Durchsuchen aller Projekte

Zum Abschluss dieses Abschnitts über den lesenden Zugriff auf den VBA-Code möchten wir Ihnen in Listing 19.9 zeigen, wie Sie alle ungeschützten Projekte nach einer Prozedur durchsuchen können. Wird diese gefunden, wird die gesamte Prozedur angezeigt.



Die Beispiel-Prozedur zum Durchsuchen aller Projekte finden Sie auf der CD-ROM in der Datei `\Beispiele\Kap19\Bsp19_01.doc` im Modul `modSearchAndReplaceLines`.

**Listing 19.9**

Durchsuchen aller Projekte nach einer bestimmten Prozedur

```

Sub subFindProzName()
    Dim intLine As Integer
    Dim intStartProc As Integer, intCountLines As Integer
    Dim strSearch As String
    Dim strMsg As String
    strSearch = "subFindProzName"
    For Each vbp In VBE.VBProjects
        If vbp.Protection = vbext_pp_locked Then GoTo p_next
        For Each vbc In vbp.VBComponents
            Set vbCode = vbc.CodeModule
            With vbCode
                For intLine = 1 To .CountOfLines
                    If .ProcOfLine(intLine, vbext_pk_Proc) = strSearch Then
                        intStartProc = .ProcStartLine(strSearch, vbext_pk_Proc)
                        intCountLines = .ProcCountLines(strSearch, vbext_pk_Proc)
                        strMsg = .Lines(intStartProc, intCountLines)
                        MsgBox strMsg, vbInformation, vbc.Name
                    End If
                Next intLine
            End With
        Next vbc
    Next vbp
End Sub

```

**Listing 19.9** Durchsuchen aller Projekte nach einer bestimmten Prozedur (*Fortsetzung*)

```

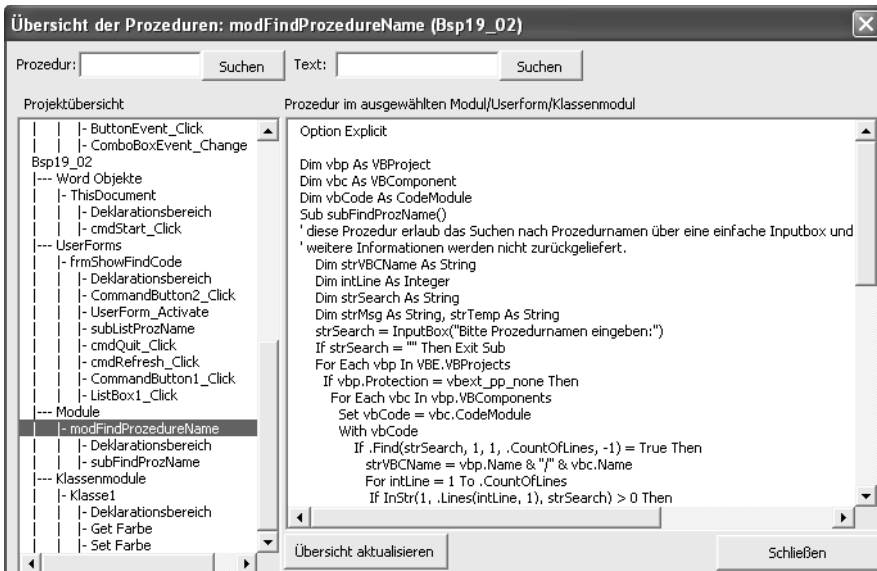
        End If
    Next intLine
End With
Next vbc
p_next:
Next vbp
Set vbCode = Nothing
End Sub

```



In der Beispieldatei *Bsp19\_02.doc*, die Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap19* finden, finden Sie ein zusätzliches Beispiel, wie Sie den gesamten VBA-Code in allen ungeschützten Projekten in einer Art Baumstruktur anzeigen lassen können, und wie Sie den gesamten VBA-Code nach einer Prozedur und nach einem beliebigen String durchsuchen können.

Nach dem Öffnen dieser Datei rufen Sie das UserForm zur Prozedur und Text-Suche über die angezeigte Schaltfläche »VBA-Code/-Prozeduren suchen«. Es öffnet sich daraufhin das in Abbildung 19.13 gezeigte Benutzerformular.

**Abbildg. 19.13** Benutzerformular zum Durchsuchen aller Projekte nach einem Prozedurnamen oder einem beliebigen Text


# Ersetzen und Entfernen von VBA-Code-Zeilen

In den vorangegangenen Abschnitten wurde das Auslesen von VBA-Code aus Komponenten und die Erstellung von neuen Komponenten behandelt. Unter den in der obigen Tabelle 19.2 aufgelisteten Zugriffsmethoden sind aber auch Methoden zum Ersetzen und Entfernen von VBA-Code-Zeilen enthalten.

**Tabelle 19.3** Methoden zum Suchen und Entfernen von Code-Zeilen

Zugriffsmethode	Beschreibung
DeleteLines	Löscht eine oder mehrere Zeilen.
ReplaceLine	Ersetzt eine vorhandene Code-Zeile durch eine angegebene Code-Zeile.

Die ReplaceLine-Methode erwartet als Parameter neben der genauen Zeile im CodeModule-Objekt der Komponente den einzufügenden Text. Beachten Sie dabei, dass einzufügende Anführungszeichen mit Hilfe der Chr(34)-Funktion maskiert werden müssen.

Das Makro in Listing 19.10 sucht in allen Projekten nach der Prozedur »subFindProzName« (siehe Listing 19.9) und führt diese aus, sodass der Inhalt der Prozedur angezeigt wird. Anschließend wird die Prozedur in »subFindNewProzName« umbenannt und die Zeile mit dem Suchtext durch den Namen einer neuen Prozedur »subFindNewProzName« ersetzt. Danach wird diese umbenannte und geänderte Prozedur erneut ausgeführt, sodass der Inhalt dieser Prozedur »subFindNewProzName« angezeigt wird. Das Ergebnis ist in Abbildung 19.14 ersichtlich; beachten Sie die beiden markierten Bereiche, wo die Änderungen vorgenommen wurden.



Die Beispiel-Prozeduren finden Sie auf der CD-ROM in der Datei \Beispiele\Kap19\Bsp19\_01.doc im Modul *modSearchAndReplaceLines*.

**Listing 19.10** Umbenennen und Ersetzen einer Zeile in einer Prozedur und Ausführen der geänderten Prozedur

```
Sub subReplaceProcedureName()
    Dim intLine As Integer
    Dim intStartProc As Integer, intCountLines As Integer
    Dim strSearch As String, strReplace As String
    Dim strMsg As String
    Dim bFound As Boolean
    strSearch = "subFindProzName"
    strReplace = "subFindNewProzName"
    Set vbc = VBE.ActiveVBProject.VBComponents("modSearchAndReplaceLines")
    Set vbCode = vbc.CodeModule
    With vbCode
        For intLine = 1 To .CountOfLines
            If .ProcOfLine(intLine, vbext_pk_Proc) = strSearch Then
                bFound = True
                Application.Run strSearch
            Exit For
        
```

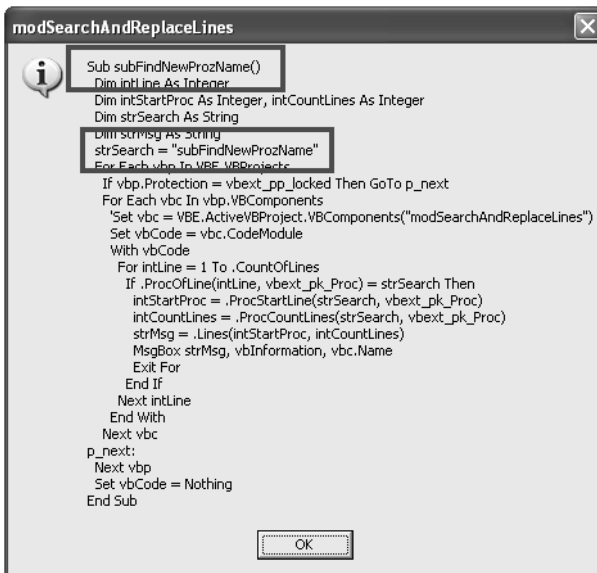
**Listing 19.10** Umbenennen und Ersetzen einer Zeile in einer Prozedur und Ausführen der geänderten Prozedur (Fortsetzung)

```

End If
Next intLine
If bFound = True Then
    For intLine = 1 To .CountOfLines
        If Trim(.Lines(intLine, 1)) = "Sub " & strSearch & "(" Then
            .ReplaceLine intLine, "Sub " & strReplace & "("
        ElseIf Trim(.Lines(intLine, 1)) = "strSearch = " & Chr(34) & _
            strSearch & Chr(34) Then
            .ReplaceLine intLine, " strSearch = " & Chr(34) & strReplace & Chr(34)
        Exit For
    End If
Next intLine
End If
End With
Set vbCode = Nothing
Application.OnTime Now, strReplace
End Sub

```

**Abbildg. 19.14** Ergebnis nach der Ausführung von Listing 19.10





**WICHTIG** Wenn sich die aufzurufende geänderte Prozedur im **selben** Code-Modul befindet wie die Prozedur, aus der heraus die Änderung vorgenommen wird, dürfen Sie diese nicht mittels

```
Application.Run "Prozedurname"
```

ausführen. Denn durch das Ausführen einer Prozedur wird das Code-Modul von Word (intern) kompiliert und im Speicher gehalten. Dazu werden alle Variablen initialisiert und, sofern sie global sind, mit Werten gefüllt. Änderungen an einer Prozedur in diesem Code-Modul werden zwar vorgenommen, aber beim Ausführen einer Prozedur wird die kompilierte Version aus dem Speicher verwendet. Damit werden Änderungen an der Prozedur nicht berücksichtigt.

Verwenden Sie in diesem Fall die `OnTime`-Methode des `Application`-Objektes. Diese Methode startet zu einem bestimmten Zeitpunkt das angegebene Makro:

```
Application.OnTime Now, "Prozedurname",1
```

Als Zeitpunkt können Sie auf die `Now`-, `TimeValue`- oder `TimeSerial`-Funktion verwenden und kombinieren. Als dritten Parameter können Sie eine Toleranzzeit angeben, zu dem das angegebene Makro abgebrochen wird, wenn es zum angegebenen Zeitpunkt nicht ausgeführt werden konnte (z.B. wenn noch ein Dialogfeld geöffnet ist).

Um beispielsweise eine Prozedur in fünf Sekunden zu starten, können Sie den folgenden Aufruf verwenden:

```
Application.OnTime Now + TimeSerial(0, 0, 5), "Prozedurname", 1
```

Mit der `DeleteLines`-Methode können Sie eine oder mehrere Zeilen löschen.

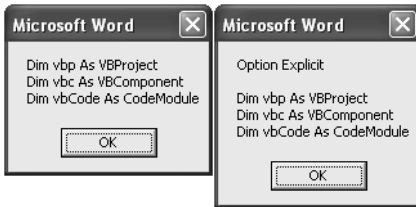
Als Parameter erwartet diese Methode die absolute Zeilenzahl im Code-Modul und die Anzahl der Zeilen. Das Makro in Listing 19.11 löscht die ersten beiden Zeilen im Code-Modul, wenn in der ersten Zeile

```
Option Explicit
```

gefolgt von einer Leerzeile steht. Steht diese Anweisung nicht in der ersten Zeile, wird sie eingefügt. Am Ende des Makros wird der Deklarationsbereich angezeigt, wie in Abbildung 19.15 ersichtlich.

**Listing 19.11** Löschen der Zeile *Option Explicit* bzw. Einfügen dieser Zeile in den Deklarationsbereich

```
Sub subDeleteLines()
    Set vbc = VBE.ActiveVBProject.VBComponents("modSearchAndReplaceLines")
    Set vbCode = vbc.CodeModule
    With vbCode
        If .Lines(1, 2) = "Option Explicit" & vbCrLf Then
            .DeleteLines 1, 2
        Else
            .InsertLines 1, "Option Explicit" & Chr(13)
        End If
        MsgBox .Lines(1, .CountOfDeclarationLines)
    End With
End Sub
```

**Abbildg. 19.15** Löschen und Einfügen von Text am Anfang des Deklarationsbereiches


## Hinzufügen von Komponenten zu einem Projekt



Die in diesem Abschnitt verwendeten Beispiel-Prozeduren finden Sie auf der CD-ROM in der Datei `\Beispiele\Kap19\Bsp19_01.doc` im Modul `modImportVBComponent`.

Einem Projekt können Sie auf zwei Arten Komponenten hinzufügen (eine Übersicht über die Komponenten finden Sie im Abschnitt »Zugriff auf die in einem Projekt enthaltenen Komponenten« in diesem Kapitel):

- Sie importieren eine als Datei (siehe Kapitel 2) vorliegende Komponente (Benutzerformular, Modul oder Klassenmodul) mit dem VBA-Code, oder
- Sie erstellen eine neue Komponente per VBA-Code und füllen diese per VBA mit Code-Zeilen.

**ACHTUNG** Sie können allerdings weder ein Word-Objekt neu anlegen noch ein solches importieren, da für jedes Projekt nur ein Word-Objekt und in diesem nur ein fest benanntes Objekt `ThisDocument` existieren kann.

Wenn Sie ein vorher exportiertes Word-Objekt `ThisDocument` importieren möchten, wird dieses als neues Klassenmodul bei diesen Komponenten eingefügt, da das Word-Objekt als Klassenmodul mit der Endung `.cls` exportiert wird.

Die Methoden in Tabelle 19.4 stehen Ihnen zum Erstellen und Bearbeiten von Komponenten, Ereignissen und Code-Zeilen zur Verfügung:

**Tabelle 19.4** Übersicht über die Methoden zum Erstellen und Bearbeiten von Komponenten per VBA-Code

Zugriffsmethode	Beschreibung
AddFromFile	Fügt den Inhalt einer angegebenen Datei in ein Code-Modul ein (siehe den Abschnitt »Makro-Anweisungen per VBA-Code importieren«).
AddFromString	Fügt einen angegebenen Text in das Code-Modul in die erste Zeile ein (siehe den Abschnitt »Makro-Anweisungen per VBA-Code einfügen«).
CreateEventProc	Erstellt für ein angegebenes Objekt eine Ereignisprozedur (siehe den Abschnitt »Makro-Anweisungen per VBA-Code einfügen«).

**Tabelle 19.4** Übersicht über die Methoden zum Erstellen und Bearbeiten von Komponenten per VBA-Code (Fortsetzung)

Zugriffsmethode	Beschreibung
DeleteLines	Löscht eine oder mehrere Zeilen (siehe den Abschnitt »Ersetzen und Entfernen von VBA-Code-Zeilen«).
InsertLines	Fügt eine oder mehrere Zeilen an einer bestimmten Stelle in das Code-Modul ein (siehe den Abschnitt »Makro-Anweisungen per VBA-Code einfügen«).
ReplaceLine	Ersetzt eine vorhandene Code-Zeile durch eine angegebene Code-Zeile (siehe den Abschnitt »Ersetzen und Entfernen von VBA-Code-Zeilen«).

## Eine vorhanden Komponente in ein Projekt importieren

Für den Import einer Komponente zu einem Projekt steht Ihnen die Import-Methode des VBProject-Objektes zur Verfügung.

Über die Import-Methode können Sie dem aktuellen Projekt einen der drei genannten Komponenten-Typen (UserForm, Standardmodul, Klassenmodul) hinzufügen. Als Parameter müssen Sie dazu den vollständigen Pfad zu der zu importierenden Datei angeben.

### TIPP

Diese Methode erwartet standardmäßig den vollständigen Pfad zur Datei. Wenn die Komponente jedoch in einem Unterverzeichnis des aktuellen Projektes liegt, können Sie auch den relativen Pfad von der Projektdatei aus, in dem sich die aufrufende Prozedur befindet, angeben. Den vollständigen Pfad erhalten Sie dann, indem Sie über die MacroContainer-Eigenschaft den absoluten Pfad der Projektdatei voranstellen (weitere Informationen zur MacroContainer-Eigenschaft finden Sie im Abschnitt »Das aktive VBA-Projekt«):

```
VBE.ActiveVBProject.VBComponents.Import MacroContainer.Path & _
  "\Beispiele\modImportModul.bas"
```

Diese MacroContainer-Eigenschaft bietet sich immer dann an, wenn der absolute Pfad zur Projektdatei (Dokument oder Dokumentvorlage), in der die aktuelle Prozedur enthalten ist, benötigt wird. Auch zur Ermittlung des aktuellen Pfades zur Projektdatei bietet sich diese Eigenschaft an.

Als Komponenten-Name wird der normalerweise in der importierten Datei als Attribut vermerkte Name verwendet. Sie können diesen aber nach dem Import über die Name-Eigenschaft ändern, wie Listing 19.12 veranschaulicht. Das Ergebnis ist in Abbildung 19.16 ersichtlich.

**Listing 19.12** Eine vorhandene Komponentendatei importieren und umbenennen

```
Sub subImportVBComponent()
    Dim vbp As VBProject
    Dim vbc As VBComponent
    Dim strVBC As String
    Dim strTyp As String
    Set vbp = VBE.ActiveVBProject
```

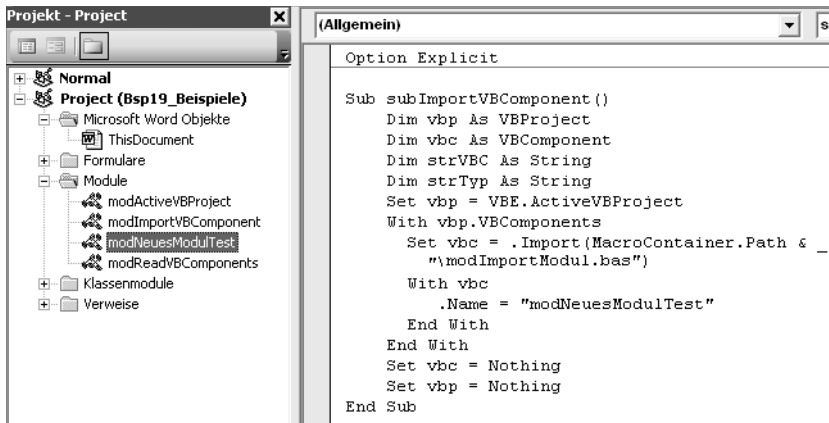
**Listing 19.12** Eine vorhandene Komponentendatei importieren und umbenennen (*Fortsetzung*)

```
With vbp.VBComponents
    Set vbc = .Import(MacroContainer.Path & "\modImportModul.bas")
    With vbc
        .Name = "modNeuesModulTest"
    End With
End With
Set vbc = Nothing
Set vbp = Nothing
End Sub
```



Sie finden die Dateien *Bsp19\_01.doc* und *modImportModul.bas* auf der CD-ROM im Verzeichnis *\Beispiele\Kap19*.

**Abbildg. 19.16** Anzeige des importierten Moduls im Projekt-Explorer



**WICHTIG** Existiert bereits eine Komponente mit dem intern vermerkten Komponenten-Namen, wird beim Importieren der Komponenten-Name mit einer fortlaufenden Nummer erweitert.

Wenn Sie nach dem Import als Namen für die Komponente einen bereits vergebenen Namen zuweisen möchten, erhalten Sie eine Fehlermeldung. Dies ist beispielsweise der Fall, wenn Sie die Prozedur *subImportVBComponent* aus Listing 19.12 ein zweites Mal ausführen.

Prüfen Sie daher vor dem Festlegen des Namens, ob es diesen bereits gibt. Im Abschnitt »Zugriff auf die in einem Projekt enthaltenen Komponenten« in diesem Kapitel wird beschrieben, wie Sie die Namen aller Komponenten ermitteln können.

## Eine neue Komponente einem Projekt hinzufügen

Um eine neue Komponente dem aktiven Projekt hinzuzufügen, verwenden Sie die Add-Methode. Diese Methode erwartet als Parameter einen der in Tabelle 19.1 vbext-Konstantwerte.

Weitere Informationen zu den verschiedenen VBComponents-Typen finden Sie im Abschnitt »Zugriff auf die in einem Projekt enthaltenen Komponenten« in diesem Kapitel.

Jede neu hinzugefügte Komponente erhält standardmäßig als Namen erstmals eine Kombination aus Typ und fortlaufender Nummer, z.B. »Modul1«. Um dieser Komponente direkt einen aussagekräftigen Namen zu geben, können Sie diesen über die Name-Eigenschaft bei der Erstellung festlegen.

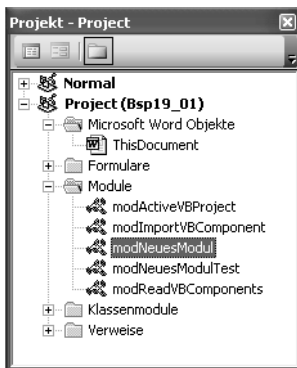
Das Makro in Listing 19.13 erstellt ein neues leeres Standardmodul mit dem Namen *modNeuesModul*.

**Listing 19.13** Hinzufügen eines neuen Standardmoduls zum aktiven Projekt

```
Sub subAddVBComponent()
    Dim vbp As VBProject
    Dim vbc As VBComponent
    Dim strVBC As String
    Dim strTyp As String
    Set vbp = VBE.ActiveVBProject
    With vbp.VBComponents
        Set vbc = .Add(vbext_ct_StdModule)
        With vbc
            .Name = "modNeuesModul"
        End With
    End With
    Set vbc = Nothing
    Set vbp = Nothing
End Sub
```

Nach Ausführen der Prozedur *subAddVBComponent* wird die neue Komponente im Projekt-Explorer unter den Komponenten »Module« einsortiert angezeigt, wie die Abbildung 19.17 veranschaulicht.

**Abbildg. 19.17** Das hinzugefügte Modul wird im Projekt-Explorer angezeigt.



Dieses Modul ist jedoch noch leer und muss mit Code-Zeilen gefüllt werden.

**HINWEIS**

Abhängig von der Einstellung im Menü des Visual Basic-Editors unter *Extras/Optionen/Editor: Variablendeklaration erforderlich* ist das Modul bzw. die Komponente entweder leer oder mit dem Ausdruck `Option Explicit` versehen, wodurch eine Deklaration aller Variablen in dieser Komponente zwingend erforderlich ist. Weitere Informationen zu den Einstellungen des Visual Basic-Editors finden Sie in Kapitel 2.

Diese Option sollte unbedingt verwendet werden, da sich dadurch Fehler, die durch falsche Variablen-Deklaration entstehen, besser erkennen und vermeiden lassen.

Gleichzeitig funktioniert die automatische Eigenschafts- und Methoden-Einblendung im Visual Basic-Editor nur für deklarierte Variablen und Objektverweise.

Nachdem Sie eine neue Komponente in dem Projekt erstellt haben, müssen Sie diese noch mit Inhalt füllen. Dazu haben Sie wieder die beiden Möglichkeiten den Text zu importieren oder per VBA-Code anzulegen.

## Makro-Anweisungen per VBA-Code importieren

Alle Prozeduren und Makro-Anweisungen werden im CodeModule der jeweiligen Komponente erfasst. Das CodeModule-Objekt enthält somit den gesamten VBA-Code der Formulare, Standardmodule und Klassenmodule, wie er im Code-Bereich der jeweiligen Komponente angezeigt wird.

Zum Importieren von Makro-Anweisungen in eine Komponente steht Ihnen die `AddFromFile`-Methode des CodeModule-Objektes zur Verfügung. Über diese Methode wird der Inhalt der angegebenen Datei der angegebenen Komponente hinzugefügt. Als Parameter müssen Sie wieder den vollständigen Pfad zur Datei angeben. Das Listing 19.14 zeigt ein Beispiel hierfür, wenn sich die einzufügende Datei im selben Ordner wie die Beispieldatei *Bsp19\_01.doc* befindet.

**Listing 19.14** VBA-Code aus einer Datei in eine Komponente hinzufügen

```
Sub subImportCodeToVBComponent()
    Dim vbp As VBProject
    Dim vbc As VBComponent
    Set vbp = VBE.ActiveVBProject
    Set vbc = vbp.VBComponents.Add(vbext_ct_StdModule)
    With vbc.CodeModule
        .AddFromFile MacroContainer.Path & _
            "\modSearchProz.bas"
        .Name = "modNeuesModulTest"
    End With
    Set vbc = Nothing
    Set vbp = Nothing
End Sub
```



Sie finden diese Dateien *Bsp19\_01.doc* und *modSearchProz.bas* auf der CDROM im Verzeichnis *\Beispiele\Kap19*.

**ACHTUNG**

Die `AddFromFile`-Methode fügt den Inhalt der angegebenen Datei der Komponente hinzu, ohne den Komponententyp zu prüfen. So können Sie den Inhalt eines Benutzerformulars durchaus einem Standardmodul hinzufügen. Allerdings werden die Attribute am Anfang einer importierten Datei, die diese normalerweise beim Import in das Projekt identifizieren, als Klartext mit in die Komponente eingefügt.

Prüfen Sie nach dem Import von Anweisungsdateien unbedingt den VBA-Code, um Fehler bei der Ausführung zu vermeiden!

## Makro-Anweisungen per VBA-Code einfügen

Zum Erzeugen von neuen Anweisungen steht Ihnen die `AddFromString`-Methode zur Verfügung. Mit dieser Methode werden Textzeilen der Komponente hinzugefügt. Dabei wird der Text in die Zeile **vor der ersten Prozedur** eingefügt. Besitzt die Komponente keine Prozedur, wird der Text hinter die letzte Zeile eingefügt.

Diese Methode besitzt daher den großen Nachteil, dass Sie damit keine Prozedur anlegen können. Denn sobald Sie mit dieser Methode den Prozedurnamen einfügen, werden die nachstehenden Texte vor diesen Namen eingefügt.

**Listing 19.15** Versuch, per `AddFromString`-Methode eine Prozedur zu erstellen

```
Sub subAddStringToModule()
    Dim vbp As VBProject
    Dim vbc As VBComponent
    Set vbp = VBE.ActiveVBProject
    Set vbc = vbp.VBComponents.Add(vbext_ct_StdModule)
    vbc.Name = "modTestModule"
    With vbc.CodeModule
        .AddFromString "Sub Main"
        .AddFromString "MsgBox Me.Name" & Chr(13) & "' Ein Kommentar"
        .AddFromString "End Sub"
    End With
End Sub
```

Das Makro in Listing 19.15 erzeugt nun nicht wie erhofft im Modul *modTestModule* eine korrekte Prozedur *Main*, sondern die nachstehenden Zeilen:

```
Option Explicit
MsgBox Me.Name
' Ein Kommentar
End Sub
Sub Main()
```

Daher bietet sich diese Methode eigentlich nur an, um allgemein gültige Variablen oder Typen zu deklarieren.

Besser ist da die `InsertLines`-Methode geeignet. Denn diese Methode erwartet neben dem Text die Zeile, in der der Text eingefügt werden soll. Ändern Sie das Beispiel aus Listing 19.15 dahingehend um, indem Sie die `InsertLines`-Methode verwenden:

**Listing 19.16** Erstellen einer Prozedur mittels der *InsertLines*-Methode

```
Sub subAddStringToModule()
    Dim vbp As VBProject
    Dim vbc As VBComponent
    Set vbp = VBE.ActiveVBProject
    Set vbc = vbp.VBComponents.Add(vbext_ct_StdModule)
    vbc.Name = "modTestModule"
    With vbc.CodeModule
        .InsertLines .CountOfLines, "Sub Main"
        .InsertLines .CountOfLines, "    MsgBox Me.Name" & Chr(13) & _
            "    ' Ein Kommentar"
        .InsertLines .CountOfLines, "End Sub"
    End With
End Sub
```

Durch Verwendung der *CountOfLines*-Eigenschaft werden die einzelnen Zeilen nacheinander ans Ende des Moduls eingefügt. Als Ergebnis erhalten Sie eine korrekte Prozedurdarstellung:

```
Option Explicit
Sub Main()
    MsgBox Me.Name
    ' Ein Kommentar
End Sub
```

Da alle Code-Zeilen als in Anführungszeichen eingeschlossen Text angeben müssen, müssen Sie alle syntaktisch notwendigen Anführungszeichen durch ihren Zeichencode maskieren. Dazu können Sie die *Chr()*-Funktion mit dem Zeichencode für die Anführungszeichen (34) verwenden: *Chr(34)* liefert somit als String das Anführungszeichen zurück, das Sie dann in die Textzeilen einfügen können.

Für Ereignisse, wie sie z.B. in Benutzerformularen ausgeführt werden, steht Ihnen eine besondere Methode zur Verfügung: Die *CreateEventProc*-Methode.

Mit dieser Methode erstellen Sie ein Ereignis für ein Objekt, z.B. das *Click*-Ereignis für eine Schaltfläche. Als Rückgabewert erhalten Sie die Nummer der ersten Zeilen, in dem dieses Ereignis beginnt.

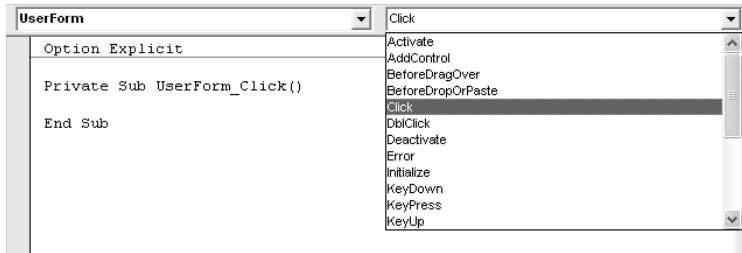
Um aber ein Objekt-Ereignis erstellen zu können, muss das entsprechende Objekt auch existieren. So können Sie für eine Schaltfläche erst dann das *Click*-Ereignis erstellen, wenn Sie vorher auf dem UserForm eine Schaltfläche mit dem Objekt-Namen angelegt haben. Wie Sie auf dem UserForm Steuerelemente (Controls) per VBA-Code erstellen, wird im Abschnitt »Anzeigen von dynamisch erzeugten UserForms« beschrieben.

Allerdings können Sie für das UserForm selbst bereits Ereignisse erstellen. Welche Ereignisse Sie für ein Objekt erstellen können, wird Ihnen in der Ereignis-Auswahlliste des Visual Basic-Editors angezeigt (Abbildung 19.18), wenn Sie sich in der Code-Anzeige des UserForms befinden.

Für ein UserForm werden meistens die *Initialize*-, *Activate*-, *Terminate*- und *QueryClose*-Ereignisse verwendet. Alle ereignisabhängigen Parameter werden dann automatisch von der *CreateEventProc*-Methode miterstellt, sodass Sie sich darum nicht kümmern müssen.



Abbildg. 19.18 Übersicht über die möglichen Ereignisse für Benutzerformulare



Das Beispiel in Listing 19.17 erstellt für ein UserForm das *QueryClose*-Ereignis, das ausgeführt wird, wenn das Benutzerformular über die *Schließen*-Schaltfläche in der rechten oberen Ecke geschlossen wird.

Listing 19.17 Erstellen des *QueryClose*-Ereignisses für ein Benutzerformular

```
Sub subAddEventToUserForm()
    Dim strQ As String
    strQ = Chr(34)
    Dim vbp As VBProject
    Dim vbc As VBComponent
    Dim intZeile As Integer
    Set vbp = VBE.ActiveVBProject
    Set vbc = vbp.VBComponents.Add(vbext_ct_MSForm)
    vbc.Name = "frmTestModule"
    With vbc.CodeModule
        intZeile = .CreateEventProc("QueryClose", "UserForm")
        .InsertLines intZeile + 1, "'Die Prozedur beginnt in Zeile: " & _
            & intZeile
        .InsertLines intZeile + 2, "MsgBox " & strQ & _
            "QueryClose-Ereignis wurde ausgelöst." & _
            strQ & ",vbInformation," & strQ & "Ereignis"
    End With
    Set vbc = Nothing
    Set vbp = Nothing
End Sub
```

Mit diesem Makro wird ein neues UserForm *frmTestModule* erstellt und das *QueryClose*-Ereignis hinzugefügt. Damit Sie auch sehen, dass das Ereignis ausgelöst wird, wenn Sie das UserForm schließen, wird über die *InsertLines*-Methode eine kurze Meldung eingefügt (Listing 19.18), die dann ausgegeben wird.

Listing 19.18 Mittels der *CreateEventProc*-Methode erzeugtes Ereignis

```
Private Sub UserForm_QueryClose(Cancel As Integer, CloseMode As Integer)
    'Die Prozedur beginnt in Zeile: 3
    MsgBox "QueryClose-Ereignis wurde ausgelöst.", vbInformation, "Ereignis"
End Sub
```

Wenn Sie anschließend das Benutzerformular starten und über die Schließen-Schaltfläche beenden, wird die Meldung in Abbildung 19.19 ausgegeben.

**Abbildg. 19.19** Ausgabe des ausgelösten *QueryClose*-Ereignisses


## Steuerelemente auf einem UserForm erzeugen

Um ein Objekt-Ereignis anlegen zu können, benötigen Sie, mit Ausnahme der UserForm selbst, ein entsprechendes Objekt (Control) in Form eines Steuerelementes auf der UserForm.

Während die Ereignis-Prozeduren im Code-Modul der jeweiligen Komponente angelegt werden, werden die Objekte im Designer-Bereich des UserForms erstellt.

### Die grafische Benutzeroberfläche *Designer*

Der Designer ist eine Eigenschaft des VBComponent-Objektes und gibt ein Objekt zurück, das das Entwurfsfenster mit einer grafischen Oberfläche im Visual Basic-Editor darstellt. Sie können den Designer standardmäßig zum grafischen Entwurf von UserForms verwenden.

Die Professional Edition und die Enterprise Edition von Microsoft Visual Basic enthalten weitere Designer für ActiveX-Steuerelemente und ActiveX-Dokumente.

Über die Designer-Eigenschaft eines UserForms haben Sie Zugriff auf dessen Eigenschaften und Methoden. Dazu gehören die direkten Eigenschaften des UserForms selbst (z.B. Aussehen und Größe) und alle Steuerelemente (Controls) mit ihren Eigenschaften.



Die in diesem Abschnitt verwendeten Beispiel-Prozeduren finden Sie auf der CD-ROM in der Datei `\Beispiele\Kap19\Bsp19_01.doc` im Modul *modCreateNewUserForm*.

### WICHTIG

Damit Ihnen alle Eigenschaften und Methoden während der Entwicklung zur Verfügung stehen, müssen Sie den Objekt-Verweis auf das UserForm auch vom Typ UserForm deklarieren:















```
Dim vbp As VBProject
Dim vbcUF As UserForm
Dim vbc As VBComponent
Set vbp = VBE.ActiveVBProject
Set vbc = vbp.VBComponents.Add(vbext_ct_MSForm)
Set vbcUF = vbc.Designer
```

Alle Standard-Steuerelemente auf dem UserForm werden über die Methoden der Controls-Eigenschaft des Benutzerformular-Designers erzeugt und konfiguriert. Die Controls-Eigenschaft liefert dazu einen Objekt-Verweis auf das jeweilige Steuerelement zurück.

Um ein Steuerelement dem Benutzerformular hinzuzufügen, verwenden Sie die Add-Methode des angegebenen Control-Objektes. Diese Methode erwartet neben dem internen Namen des Steuerelementes auch eine Ressource-ID (Programmatic ID – ProgID), die das jeweilige Steuerelement-Objekt bezeichnet.

Die Ressource-IDs für die Standard-Steuerelemente sind in der Tabelle 19.5 aufgelistet.

**Tabelle 19.5** Übersicht über die Ressource-IDs der Standard-Steuerelemente

Bezeichnung	Ressource-ID (ProgID)	Abbildung
Kontrollkästchen	Forms.CheckBox.1	
Kombinationsfeld	Forms.ComboBox.1	
Befehlsschaltfläche	Forms.CommandButton.1	
Rahmen	Forms.Frame.1	
Abbildung	Forms.Image.1	
Bezeichnungsfeld	Forms.Label.1	
Listenfeld	Forms.ListBox.1	
Multiseiten	Forms.MultiPage.1	
Optionsfeld	Forms.OptionButton.1	
Bildlaufleiste	Forms.ScrollBar.1	
Drehfeld	Forms.SpinButton.1	
Register	Forms.TabStrip.1	
Textfeld	Forms.TextBox.1	
Umschaltfeld	Forms.ToggleButton.1	

Über die weiteren Methoden des jeweiligen Controls-Objektes können Sie alle Eigenschaften des Steuerelementes konfigurieren, wie Sie es auch im Visual Basic-Editor über das Eigenschaftsfenster des jeweiligen Steuerelementes vornehmen können.

Das Makro in Listing 19.19 erstellt auf einem neuen UserForm zwei Schaltflächen mit Namen *cmdQuit* und *cmdMsg*, positioniert und beschriftet sie und weist ihnen Ereignisse zu. Über die Schaltfläche *cmdQuit* wird das UserForm geschlossen und über die Schaltfläche *cmdMsg* wird eine Meldung ausgegeben. Zum Schluss wird das Benutzerformular angezeigt.

**Listing 19.19** Dynamisch erstelltes Benutzerformular mit Steuerelementen

```

Sub subAddControlsToUserForm()
    Dim vbp As VBProject
    Dim vbcUF As UserForm, frm As Object
    Dim ctlUF1 As MSForms.Label
    Dim ctlUF2 As MSForms.CommandButton
    Dim ctlUF3 As MSForms.CommandButton
    Dim vbc As VBComponent
    Dim strVBC As String
    Dim strTyp As String
    Dim intZeile As Integer
    Set vbp = VBE.ActiveVBProject
    Set vbc = vbp.VBComponents.Add(vbext_ct_MSForm)
    With vbc
        .Properties("Width") = 300
        .Properties("Height") = 200
        .Properties("Caption") = "Per Code erstellte UserForm"
        On Error Resume Next
        .Name = "frmUserFormPerCode"
        On Error GoTo 0
        Set vbcUF = vbc.Designer
        Set ctlUF1 = vbcUF.Controls.Add("Forms.Label.1", "lblText", True)
        With ctlUF1
            .Left = 2
            .Height = 14
            .Top = .Height + 5
            .Width = vbcUF.InsideWidth - 5
            .BorderStyle = 1
            .TextAlign = fmTextAlignCenter
            .Caption = "Dieses ist ein UserForm, das komplett per VBA-Code erzeugt wurde."
        End With
        Set ctlUF2 = vbcUF.Controls.Add("Forms.CommandButton.1", "cmdQuit", True)
        With ctlUF2
            .Left = vbcUF.InsideWidth - .Width - 5
            .Height = 30
            .Top = vbcUF.InsideHeight - .Height - 5
            .Caption = "Close"
        End With
        intZeile = .CodeModule.CreateEventProc("Click", "cmdQuit")
        .CodeModule.InsertLines intZeile + 1, "Unload Me"
        Set ctlUF3 = vbcUF.Controls.Add("Forms.CommandButton.1", "cmdMsg", True)
        With ctlUF3
            .Left = 10
            .Height = 30
            .Top = vbcUF.InsideHeight - .Height - 5
            .Caption = "Zeige MsgBox"
        End With
        intZeile = .CodeModule.CreateEventProc("Click", "cmdMsg")
        .CodeModule.InsertLines intZeile + 1, "MsgBox " & Chr(34)
        & "Hello World!" & Chr(34) & ",vbInformation, " & Chr(34) & "Meldung" & Chr(34)
        End With
    Set vbc = Nothing
    Set vbcUF = Nothing
    Set vbp = Nothing
End Sub

```

Wenn Sie das Benutzerformular starten, sehen Sie das erzeugte UserForm in Abbildung 19.20 mit dem Bezeichnungsfeld und den beiden Schaltflächen. Wenn Sie auf die Schaltfläche *Zeige MsgBox* klicken, werden Ihnen die wohl bekannten Worte »Hello World« angezeigt.

Abbildg. 19.20 Anzeige des dynamisch erstellten Benutzerformulars.



## Entfernen von Komponenten aus einem Projekt

Zum Entfernen einzelner Komponenten aus einem nicht gesperrten Projekt steht Ihnen die *Remove*-Methode des *VBComponents*-Objektes zur Verfügung. Diese Methode erwartet als Parameter ein *VBComponents*-Objekt, das ein vorhandenes Modul, Benutzerformular oder Klassenmodul darstellt (z.B. über einen Objektverweis):

```
vbp.VBComponents.Remove vbc
```

Das Makro in Listing 19.20 erstellt ein neues Modul im aktuellen Projekt, gibt den (automatisch vergebenen) Namen aus und entfernt die Komponente wieder aus dem Projekt. Zur Kontrolle, dass das Modul wieder entfernt wurde, werden dann die Namen aller vorhandenen Module angezeigt.



Die verwendete Beispiel-Prozedur finden Sie auf der CD-ROM in der Datei *\Beispiele\Kap19\Bsp19\_01.doc* im Modul *modRemoveVBComponent*.

Listing 19.20 Entfernen einer neu angelegten Komponente

```
Sub subRemoveVBComponent()
    Dim vbp As VBProject
    Dim vbc As VBComponent
    Dim strVBC As String
    Set vbp = VBE.ActiveVBProject
    With vbp.VBComponents
        Set vbc = .Add(vbext_ct_StdModule)
        MsgBox vbc.Name
        vbp.VBComponents.Remove vbc
    End With
End Sub
```

**Listing 19.20** Entfernen einer neu angelegten Komponente (*Fortsetzung*)

```
For Each vbc In vbp.VBComponents
    If vbc.Type = vbext_ct_StdModule Then
        strVBC = strVBC & vbc.Name & vbCrLf
    End If
Next vbc
MsgBox strVBC, vbInformation, "VBComponents"
Set vbc = Nothing
Set vbp = Nothing
End Sub
```

**WICHTIG** Die Remove-Methode akzeptiert nur ein VBComponent-Objekt der VBComponents-Auflistung. Wenn Sie versuchen, das VBComponents-Objekt über den Namen anzugeben, erhalten Sie die Fehlermeldung »Typen unverträglich«.

## Anzeigen von dynamisch erzeugten UserForms

Mit den bisherigen Eigenschaften und Methoden können Sie jetzt zwar ein UserForm erstellen, mit Steuerelementen gestalten und Ereignisse erstellen, es steht Ihnen aber kein direkter Befehl zum Anzeigen des UserForms zur Verfügung.

Bei UserForms, die Sie direkt im Visual Basic-Editor erstellen, können Sie diese über ihren Namen und der Show-Eigenschaft aufrufen. Dazu wird über den Namen auf das entsprechende UserForm-Objekt referenziert.

### PROFITIPP

Im Gegensatz zu den im Visual Basic-Editor erstellten UserForms stehen Ihnen bei dynamisch erstellten UserForms diese nicht als Referenz zur Verfügung, da das entsprechende UserForm-Objekt vor dem Ausführen des Makros noch nicht bekannt ist und somit vom Visual Basic-Editor auch nicht referenziert wurde. Da Sie auch (mit Einschränkungen) den Namen des UserForms zur Laufzeit ändern können, wodurch sich auch das UserForm-Objekt selbst ändern würde, müssen Sie für dynamisch erstellte UserForms die UserForms-Auflistung verwenden.

Über diese UserForms-Auflistung erhalten Sie Zugriff auf alle **geladenen** UserForms. Zugriff auf ein einzelnes UserForm erhalten Sie durch das UserForm-Objekt, das Ihnen die UserForms-Auflistung zurückliefert.

Um nun einen Zugriff auf eine bestimmte *UserForm*-Komponente in einem Projekt zu erhalten, müssen Sie das UserForm erst laden und somit der UserForms-Auflistung hinzufügen. Hierzu steht Ihnen die Add-Methode zur Verfügung, der Sie entweder den Index oder den Namen des UserForms, wie er im Projekt-Explorer angezeigt wird, als Parameter mitgeben müssen.

Anschließend stehen Ihnen alle Eigenschaften eines UserForm-Objektes zur Verfügung. Da es sich dabei aber um ein Objekt vom Typ *Object* handelt und nicht um ein Objekt vom Typ *UserForm*, stehen Ihnen die Eigenschaften und Methoden des UserForm **nicht** zur Verfügung.

Damit Ihnen während der Entwicklung trotzdem die Eigenschaften und Methoden des UserForm-Objektes zur Verfügung stehen, können Sie das hinzugefügte UserForm-Objekt für die Zeit der Entwicklung auch vom Typ *UserForm* deklarieren. ►

## PROFITIPP

Wenn Sie allerdings mit dieser »Falschdeklaration« versuchen das Makro auszuführen, werden Sie bestimmte Fehlermeldungen erhalten, da nicht alle Eigenschaften und Methoden, die für ein UserForm gültig sind, auch für ein Objekt zutreffen.

Vergessen Sie auf jeden Fall nicht, die Deklaration zu Testzwecken und vor der Freigabe des Codes wieder auf `Object` zu ändern!

Das Makro in Listing 19.21 durchläuft im aktuellen Projekt alle Komponenten. Handelt es sich um ein UserForm, wird dieses der UserForms-Auflistung hinzugefügt und anschließend über die `Show`-Methode angezeigt.

Durch das anschließende Löschen/Zurücksetzen des `objFrm`-Objektes über die Zeile

```
Set objFrm = Nothing
```

wird das UserForm wieder aus der UserForms-Auflistung entfernt.



Die verwendeten Beispiel-Prozeduren finden Sie auf der CD-ROM in der Datei `\Beispiele\Kap19\Bsp19_01.doc` im Modul `modShowAllUserForms`.

**Listing 19.21** Anzeige aller Benutzerformulare im aktuellen Projekt

```
Sub subShowAllUserForms()
    Dim vbc As VBComponent
    Dim objFrm As Object
    For Each vbc In VBE.ActiveVBProject.VBComponents
        If vbc.Type = vbext_ct_MSForm Then
            Set objFrm = VBA.UserForms.Add(vbc.Name)
            objFrm.Caption = vbc.Name
            objFrm.Show
            Set objFrm = Nothing
        End If
    Next vbc
End Sub
```

Mit Listing 19.21 erhalten Sie Zugriff auf das UserForm mit normalerweise einer Reihe von Steuerelementen. In Listing 19.19 haben Sie gesehen, dass neue Steuerelemente (Controls-Objekte) über die Designer-Eigenschaft des `VBComponents`-Objektes dem UserForm hinzugefügt werden können. Sie erhalten allerdings auch über das UserForm-Objekt die Möglichkeit, auf die Steuerelemente zuzugreifen, da dieses Objekt viele Eigenschaften und Methoden eines als UserForm deklarierten Objektes besitzt. So ist eine Eigenschaft die `Controls`-Eigenschaft mit der bekannten `Name`-Methode. Zur Ermittlung des Steuerelement-Typs steht Ihnen entweder die `If...Then...Else`-Anweisung mit der `TypeOf`-Anweisung zur Verfügung, oder Sie verwenden die `CallByName`-Funktion. Diese `CallByName`-Funktion können Sie verwenden, um zur Laufzeit eine Eigenschaft eines Steuerelement-Objektes einzustellen bzw. zu erhalten oder eine Methode dieses Objektes aufzurufen. So liefert folgende Zeile die Beschriftung des `CommandButton`-Steuerelementes `CommandButton1`:

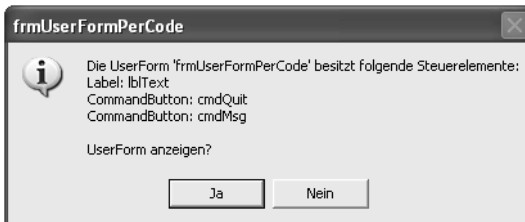
```
MsgBox CallByName(CommandButton1,"Caption",VbGet)
```

Die Prozedur *subShowUserFormControlInfos* in Listing 19.22 benutzt diese Funktion, um für alle UserForms die enthaltenen Steuerelemente mit Typ und Namen anzuzeigen (Abbildung 19.21), bevor das jeweilige UserForm angezeigt werden kann.

**Listing 19.22** Ermittlung und Ausgabe aller Steuerelemente eines Benutzerformulars mit Typ und Namen

```
Sub subShowUserFormControlInfos()
    Dim vbc As VBComponent
    Dim objFrm As Object
    Dim strMSG As String
    Dim intCtl As Integer, intRet As Integer
    For Each vbc In VBE.ActiveVBProject.VBComponents
        If vbc.Type = vbext_ct_MSForm Then
            Set objFrm = VBA.UserForms.Add(vbc.Name)
            strMSG = "Die UserForm '" & objFrm.Name & _
                "' besitzt folgende Steuerelemente:" & vbCrLf
            For intCtl = 0 To objFrm.Controls.Count - 1
                strMSG = strMSG & TypeName(objFrm.Controls(intCtl)) & ": " & _
                    CallByName(objFrm.Controls(intCtl), "Name", VbGet) & vbCrLf
            Next intCtl
            intRet = MsgBox(strMSG & vbCrLf & "UserForm anzeigen?", _
                vbInformation + vbYesNo, objFrm.Name)
            objFrm.Caption = vbc.Name
            If intRet = vbYes Then
                objFrm.Show
            End If
            Set objFrm = Nothing
        End If
    Next vbc
End Sub
```

**Abbildg. 19.21** Auflistung der Steuerelemente eines Benutzerformulars und Anzeige des *UserForm*-Objektes



Wenn Sie auf die Schaltfläche *Ja* klicken, wird das UserForm-Objekt geladen und angezeigt.



Abbildg. 19.22 Anzeige der UserForm *frmUserFormPerCode*

## Verweise auf Bibliotheken und Dateien ermitteln und zur Laufzeit setzen

Voraussetzung für die Verwendung der VBA-Funktionen und Befehle ist die Bereitstellung entsprechender Bibliotheken (.dll), Objektbibliotheken (.olb) oder Objekten (.ocx) durch das Betriebssystem oder ein anderes Programm. So sind bereits ein paar Verweise notwendig, um überhaupt in den Visual Basic-Editor zu gelangen oder im Visual Basic-Editor ein UserForm zu erstellen. Diese Verweise werden automatisch gesetzt und lassen sich auch nicht entfernen.

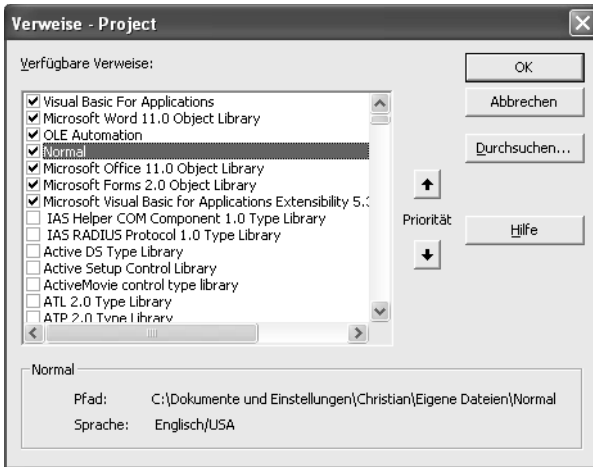


Die in diesem Abschnitt verwendeten Beispiel-Prozeduren finden Sie auf der CD-ROM in der Datei `\Beispiele\Kap19\Bsp19_01.doc` im Modul *modReferences*.

## Übersicht über die gesetzten Verweise

Welche Verweise gerade im Visual Basic-Editor für ein Projekt gesetzt sind, können Sie entweder über den Menübefehl *Extras/Verweise* im zugehörigen Dialogfeld ablesen oder über die References-Auflistung des Projektes ermitteln.

Das Listing 19.23 zeigt alle gesetzten Verweise (Abbildung 19.24), wie sie auch über den entsprechenden Menüpunkt zu sehen sind, für das aktive Projekt an, indem es die Count-Eigenschaft der References-Auflistung auswertet. Neben dem Namen wird, sofern verfügbar, die Beschreibung des jeweiligen Verweises ausgegeben.

**Abbildg. 19.23** Übersicht über die gesetzten und verfügbaren Verweise eines Projektes

**Listing 19.23** Auflisten aller VBE-Verweise

```
Sub subVBEVerweise()
    Dim strMSG As String
    Dim intRef As Integer
    With VBE.ActiveVBProject.References
        For intRef = 1 To .Count
            strMSG = strMSG & .Item(intRef).Name & ": " & .Item(intRef).Description & vbCrLf
        Next intRef
    End With
    MsgBox strMSG, vbInformation, "Auflistung der Verweise"
End Sub
```

**Abbildg. 19.24** Auflistung der gesetzten Verweise mit Beschreibung


Über die References-Eigenschaft eines Projektes lassen sich aber nicht nur die gesetzten Verweise ermitteln, sondern Sie können darüber auch zur Laufzeit neue Bibliotheken laden und Verweise auf diese setzen. Auch können Sie die Gültigkeit der Verweise überprüfen und diese ggf. neu setzen.

## Neuen Verweis setzen

Zum Hinzufügen eines neuen Verweises stehen Ihnen die zwei Methoden `AddFromFile` und `AddFromGuid` zur Verfügung. Während die Methode `AddFromFile` als Parameter den vollständigen Pfad zur Verweisdatei erwartet, müssen Sie bei der Methode `AddFromGuid` den eindeutigen Bezeichner (GUID) des Verweises kennen und angeben.

### HINWEIS

Die `AddFromGuid`-Methode durchsucht die Registrierung, um den hinzuzufügenden Verweis zu ermitteln. Der global eindeutige Bezeichner (GUID) kann eine Klassenbibliothek, ein Steuerelement, ein Klassenbezeichner usw. sein.

Die Tabelle 19.6 listet die GUID für die wichtigsten Verweis-Bibliotheken auf. Das Listing 19.24 veranschaulicht, wie diese Angaben programmtechnisch zu ermitteln sind; das Ergebnis ist in Abbildung 19.25 ersichtlich.

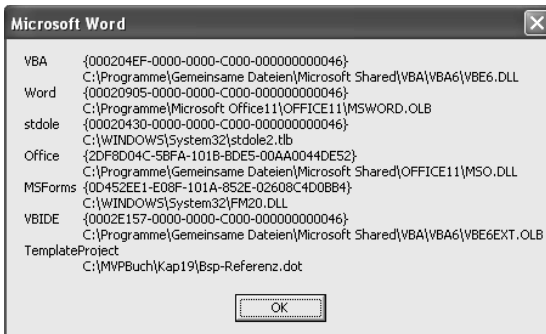
**Tabelle 19.6** GUIDs für die Standardverweise in Office 11 (Office 2003)

Name	GUID/Standardpfad/Beschreibung
VBA	{000204EF-0000-0000-C000-000000000046}
	C:\Programme\Gemeinsame Dateien\Microsoft Shared\VBA\VBA6\VBE6.DLL
	Visual Basic For Applications
Word	{00020905-0000-0000-C000-000000000046}
	C:\Programme\Microsoft Office\OFFICE11\MSWORD.OLB
	Microsoft Word 11.0 Object Library
stdole	{00020430-0000-0000-C000-000000000046}
	C:\WINDOWS\system32\Stdole2.tlb
	OLE Automation
Office	{2DF8D04C-5BFA-101B-BDE5-00AA0044DE52}
	C:\Programme\Gemeinsame Dateien\Microsoft Shared\OFFICE11\MSO.DLL
	Microsoft Office 11.0 Object Library
MSForms	{0D452EE1-E08F-101A-852E-02608C4D0BB4}
	C:\WINDOWS\System32\FM20.DLL
	Microsoft Forms 2.0 Object Library
VBEIDE	{0002E157-0000-0000-C000-000000000046}
	C:\Programme\Gemeinsame Dateien\Microsoft Shared\VBA\VBA6\VBE6EXT.OLB
	Microsoft Visual Basic for Applications Extensibility 5.3

Für die Office-Versionen »Office 2000 (Office 9)« und »Office 2002 (Office XP/Office 10)« ändern sich nur die Versionsnummern und Dateinamen der jeweiligen Versionsdateien.

**Listing 19.24** Ermitteln der gesetzten Verweise mit Pfad und GUID

```
Sub subListGUIDReferences()
    Dim strMSG As String
    Dim intRef As Integer
    With VBE.ActiveVBProject.References
        For intRef = 1 To .Count
            strMSG = strMSG & .Item(intRef).Name & vbTab & _
                .Item(intRef).GUID & vbCrLf & vbTab & _
                .Item(intRef).FullPath & vbCrLf
        Next intRef
    End With
    MsgBox strMSG
End Sub
```

**Abbildg. 19.25** Ausgabe der gesetzten Verweise mit Pfad und GUID


Das Listing 19.25 setzt einen neuen Verweis auf die Datei *Bsp19-Referenz.dot*, sofern dieser noch nicht in der Liste der gesetzten Verweise aufgeführt ist. Damit das Beispiel funktioniert, kopieren Sie bitte die Datei *Bsp19-Referenz.dot* von der CD-ROM aus dem Verzeichnis *\Beispiele\Kap19\* in das lokale Verzeichnis *C:\MVPBuch\Kap19\Bsp19-Referenz.dot*. Wenn Sie die Datei an einen anderen Ort speichern, passen Sie bitte den vollständigen Pfad in Listing 19.25 an den gewählten Speicherort an.

**Listing 19.25** Setzen eines Verweises auf eine Dokumentvorlage

```
Sub subSetReference()
    fktSetReferences "C:\MVPBuch\Kap19\Bsp19-Referenz.dot"
End Sub

Function fktSetReferences(strRefPath As String)
    Const c_Ref As String = "Verweis setzen"
    Dim strGUID As String
    Dim strFile As String
    Dim oRef As Reference
    Dim bFound As Boolean: bFound = False
    If Dir(strRefPath) = "" Then
        MsgBox "Die Verweisdatei existiert nicht!", vbCritical, c_Ref
        Exit Function
    End If
    With VBE.ActiveVBProject.References
        For Each oRef In VBE.ActiveVBProject.References
            If oRef.FullPath = strRefPath Then
```

Listing 19.25 Setzen eines Verweises auf eine Dokumentvorlage (Fortsetzung)

```

        bFound = True
        Exit For
    End If
Next oRef
If bFound = False Then
    Set oRef = .AddFromFile(strRefPath)
    MsgBox "Verweis auf" & vbCrLf & strRefPath & vbCrLf & _
        "erfolgreich gesetzt", vbInformation, c_Ref
Else
    MsgBox "Verweis auf" & vbCrLf & strRefPath & vbCrLf & _
        "war bereits gesetzt", vbInformation, c_Ref
End If
End With
End Function

```

Wenn Sie die AddFromGuid-Methode verwenden möchten, müssen Sie diese GUID kennen, da diese eindeutig und unabhängig von der verwendeten Office-Version ist (siehe Tabelle 19.6).

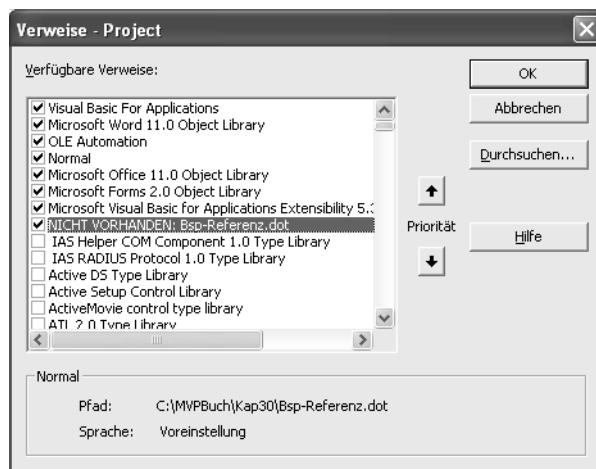
## Ungültige Verweise korrigieren bzw. entfernen

Normalerweise behalten die Standardverweise auf die Office/VBA-Bibliotheken ihre Gültigkeit, wenn Sie beispielsweise die Datei verschieben oder auf einen anderen Rechner kopieren. Auch wenn Sie die Datei mit einer älteren Word-Version öffnen, werden die Verweise durch Verweise durch die entsprechenden Versionen der Bibliotheken ersetzt.

Wenn Sie aber Verweise auf eigene Dokumentvorlagen setzen und dann die Datei auf einen anderen Rechner kopieren oder die Verweis-Dokumentvorlage entfernen, erhalten Sie ungültige Verweise in der Übersicht.

Diese ungültigen Verweise können zu Fehlern in den Makros führen und sogar die Stabilität von Word beeinträchtigen, wie beispielsweise die Abbildung 19.26 zeigt.

Abbildg. 19.26 Fehlerhafte Einträge in der Verweisübersicht



Die Gültigkeit der gesetzten Verweise können Sie mit der `IsBroken`-Eigenschaft prüfen. Diese Eigenschaft vergleicht einen Verweis mit den Verweis-Einträgen in der Registry und liefert den Status der Gültigkeit zurück.

Mit dem Makro in Listing 19.26 überprüfen Sie die gesetzten Verweise. Wenn Sie die im Listing 19.25 hinzugefügte Verweisdatei *Bsp-Referenz.dot* entfernen oder umbenennen und das Makro ausführen, erhalten Sie den Hinweis in Abbildung 19.27, dass der Verweis ungültig sei.

**Listing 19.26** Prüfen der Verweise auf Gültigkeit

```
Sub subProofReferences()
    Const c_Ref As String = "Verweis prüfen"
    Dim ret As Integer
    Dim oRef As Reference
    With VBE.ActiveVBPProject.References
        For Each oRef In VBE.ActiveVBPProject.References
            If oRef.IsBroken Then
                ret = MsgBox("Der Verweis" & vbCrLf & oRef.Name & vbCrLf & _
                    "ist nicht gültig." & vbCrLf & _
                    "Verweis entfernen?", vbCritical + vbYesNo, c_Ref)
                If ret = vbYes Then
                    .Remove oRef
                End If
            End If
        Next oRef
    End With
End Sub
```

Um Fehler bezüglich fehlerhafter Verweise zu vermeiden, können Sie diese anschließend entfernen oder neu setzen.

**Abbildg. 19.27** Hinweismeldung auf fehlerhafte Verweise



Um fehlerhafte Verweise neu zu setzen, können Sie diese aber nicht direkt wieder hinzufügen, sondern Sie müssen zuerst den fehlerhaften Eintrag entfernen, d.h. den Haken vor dem Eintrag entfernen, und anschließend den Verweis entweder mittels der `AddFromFile`- oder `AddFromGuid`-Methode wieder hinzufügen. Zum Entfernen eines Verweises steht Ihnen die `Remove`-Methode der `References`-Eigenschaft zur Verfügung.

Die Prozedur in Listing 19.27 überprüft alle gesetzten Verweise und versucht diese entweder anhand des Dateipfades oder der GUID wieder zu setzen.

**Listing 19.27** Reparieren ungültiger Verweise

```
Sub subReSetReferences()
    Const c_Ref As String = "Verweis prüfen/setzen"
    Dim strGUID As String
```

Listing 19.27 Reparieren ungültiger Verweise (Fortsetzung)

```

Dim strFile As String
Dim oRef As Reference
With VBE.ActiveVBProject.References
    For Each oRef In VBE.ActiveVBProject.References
        If oRef.IsBroken Then
            If strGUID = "" Then
                strFile = oRef.FullPath
                .Remove oRef
                If Dir(strFile) = "" Then
                    MsgBox "Der Verweis ist nicht vorhanden:" & vbCrLf & _
                        oRef.Name, vbCritical, c_Ref
                End If
                If strFile <> "" Then
                    .AddFromFile strFile
                    MsgBox "Der Verweis wurde neu hinzugefügt:" & vbCrLf & _
                        strFile, vbInformation, c_Ref
                End If
            Else
                strGUID = oRef.GUID
                strFile = oRef.Name
                .Remove oRef
                .AddFromGuid strGUID, 0, 0
                MsgBox "Der Verweis '" & strFile & "' wurde neu gesetzt:" & _
                    vbCrLf & strGUID, vbCritical, c_Ref
            End If
        End If
    Next oRef
End With
End Sub

```

Problematisch wird das Reparieren bei Verweisdateien (z.B. Dokumentvorlagen), wenn diese verschoben wurden. Für diesen Fall können Sie das Listing 19.27 dahingehend modifizieren, dass Sie optional den neuen Pfad zur Datei mit angeben. In Listing 19.28 werden alle ungültigen Verweise dahingehend geprüft, ob die angegebene Verweisdatei *Bsp-Referenz.dot* betroffen ist. Ist dies der Fall, wird nach Bestätigung (Abbildung 19.28) der ungültige Verweis durch den neuen Verweis ersetzt. Handelt es sich bei dem Verweis um eine Bibliothek mit GUID, wird diese ermittelt und ein Verweis mit dieser GUID neu gesetzt.

Listing 19.28 Ungültige Verweise ersetzen

```

Sub subTestAndReSetReference()
    fktReSetReferences "G:\MVPBuch\Kap19\Bsp19-Referenz.dot"
End Sub
Function fktReSetReferences(Optional strRefPath)
    Const c_Ref As String = "Verweis prüfen/setzen"
    Dim strGUID As String
    Dim strFile As String
    Dim strRef As String
    Dim oRef As Reference
    Dim bExists As Boolean: bExists = True
    Dim ret As Integer
    If IsMissing(strRefPath) = False Then
        If Dir(strRefPath) = "" Then
            MsgBox "Die angegebene Verweisdatei existiert nicht:" & _

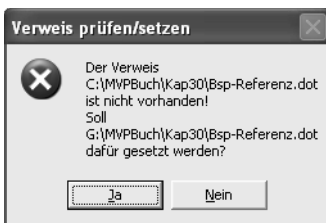
```

**Listing 19.28** Ungültige Verweise ersetzen (Fortsetzung)

```

        vbCrLf & strRefPath, vbCritical, c_Ref
        bExists = False
    End If
End If
If bExists = True Then
    strRef = Right(strRefPath, InStrRev(strRefPath, "\", -1) - 1)
End If
With VBE.ActiveVBProject.References
    For Each oRef In VBE.ActiveVBProject.References
        If oRef.IsBroken Then
            If strGUID = "" Then
                strFile = oRef.FullPath
                .Remove oRef
                If Dir(strFile) = "" And InStr(1, strRefPath, strRef) > 0 And bExists = True Then
                    ret = MsgBox("Der Verweis" & vbCrLf & strFile & vbCrLf & _
                        "ist nicht vorhanden!" & vbCrLf & "Soll " & vbCrLf & strRefPath & vbCrLf & _
                        "dafür gesetzt werden?", vbCritical + vbYesNo, c_Ref)
                    If ret = vbYes Then
                        .AddFromFile strRefPath
                        MsgBox "Der Verweis wurde neu hinzugefügt:" & vbCrLf & _
                            strRefPath, vbInformation, c_Ref
                    End If
                ElseIf Dir(strFile) = "" Then
                    MsgBox "Der Verweis ist nicht vorhanden und wurde entfernt:" & vbCrLf & _
                        strFile, vbCritical, c_Ref
                End If
            Else
                strGUID = oRef.GUID
                strFile = oRef.Name
                .Remove oRef
                .AddFromGuid strGUID, 0, 0
                MsgBox "Der Verweis '" & strFile & "' wurde neu gesetzt:" & _
                    vbCrLf & strGUID, vbCritical, c_Ref
            End If
        End If
    Next oRef
End With
End Function

```

**Abbildg. 19.28** Ungültigen Verweis ersetzen




# Zusammenfassung

In diesem Kapitel wurde Ihnen ein Überblick über einige Funktionen, Befehle und Möglichkeiten gegeben, um auf den Visual Basic-Editor und dort die Projekte, die verschiedenen Module, UserForms und Objekt-Verweise zuzugreifen:

- Sie haben gesehen, wie Sie eine Übersicht über alle VBA-Projekte erhalten (Seite 718) und auf das aktive VBA-Projekt (Seite 719) mit seinen darin enthaltenen Modulen und Benutzerformularen (Seite 721) zugreifen können
- Ein weiterer Schwerpunkt war der Zugriff auf den VBA-Code eines Projektes (Seite 724) und das Auslesen bestimmter Zeilen (Seite 725 und Seite 726).
- Nach dem lesenden Zugriff auf den VBA-Code wurde Ihnen gezeigt, wie Sie in einem Projekt gezielt einzelne VBA-Code-Zeilen suchen, ersetzen und entfernen können (Seite 731).
- Der nächste Schwerpunkt betraf die Behandlung von Komponenten in einem Projekt. So wurde Ihnen gezeigt, welche Möglichkeiten der Visual Basic-Editor bietet, um gesamte Komponenten, (Seite 734) aber auch einzelne Code-Zeilen in ein Projekt einzufügen (Seite 738 und Seite 739).
- Neben dem Einfügen von VBA-Code haben Sie des Weiteren gesehen, wie Sie Steuerelemente in eine UserForm einfügen (Seite 742) und eine so erstellte UserForm zur Laufzeit anzeigen lassen können (Seite 746).
- Abschließend wurde Ihnen in diesem Kapitel gezeigt, wie Sie während der Bearbeitung von Makros benötigte Verweise auf Bibliotheken ermitteln (Seite 749) und ggf. neu setzen können (Seite 751).

Anhand der aufgeführten Listings und den Beispieldateien auf der CD-ROM zum Buch sollten Sie nun in der Lage sein, diese Funktionen in eigene Anwendungen oder Makros einzubinden.

Natürlich können in diesem relativ kurzen Kapitel nicht alle Möglichkeiten und Funktionen/Eigenschaften, die das VBE-Objekt des Visual Basic-Editors zur Verfügung stellt, angesprochen und bis zur letzten Methode behandelt werden, da dies den Rahmen eines eigenen Buches sprengen würde. Weitere hilfreiche Informationen finden Sie sowohl in der Online-Hilfe des Visual Basic-Editors als auch im Internet auf der MSDN-Homepage von Microsoft (<http://msdn.microsoft.com/library/default.asp>) im Bereich »Office Solutions Development«.



# Teil E

## Praktische Anwendungsbeispiele

### In diesem Teil:

<b>Kapitel 20</b>	Kalenderblatt erstellen	761
<b>Kapitel 21</b>	Aktive Symbolleisten	771
<b>Kapitel 22</b>	Aktualisierung der Felder mit einem Fortschrittsbalken visualisieren	779
<b>Kapitel 23</b>	Allerlei in Kopf- und Fußzeilen	799
<b>Kapitel 24</b>	Grafiken mit Beschriftungen	821
<b>Kapitel 25</b>	Rechtschreibfehler für den Ausdruck formatieren	831
<b>Kapitel 26</b>	Rechnung erstellen mit Formularfeldern	839
<b>Kapitel 27</b>	Attraktive Tischkarten im Nu	849



## Kapitel 20

# Kalenderblatt erstellen

### In diesem Kapitel:

Die Aufgabenstellung	762
Diskussion zum Code	764
Zusammenfassung	769

In Kapitel 6 wurden die Grundlagen der Arbeit mit dem Word-Objektmodell vorgestellt. Die vorliegende Lösung veranschaulicht die unten stehenden Objekte, Eigenschaften und Methoden, wobei der Schwerpunkt auf der Arbeit mit Tabellen liegt. Zudem wird deutlich, wie flexibel es sich mit Datumsangaben in VBA arbeiten lässt, im Gegensatz zu den Feldfunktionen von Word (siehe den Abschnitt »Feldfunktionen« in Kapitel 6).

---

**Im Blickpunkt:**

## VB-Funktionen

CDate

DateAdd, DateDiff

Format

## Word-Objekte

Application.CentimetersToPoints

Document.PageSetup.Zoom

FormFields.Add, FormField.Name

Table.Add, Table.Borders, Table.Shading, Table.Style, Table.Condition

Row.HeightRule, Row.Height

Cell.Merge, Cell.VerticalAlignment

---

## Die Aufgabenstellung

Ein oft geäußelter Wunsch ist ein Werkzeug, um Monats-Kalenderblätter zu erstellen. Als Anregung stellen wir hier eine Lösung vor, die auf einer Tabelle basiert. Das Blatt aus Abbildung 20.1 eignet sich sowohl für den Online-Gebrauch wie auch für den Ausdruck auf Papier. Für die Dateneingabe online (in Word) enthält jede nicht dunkel schattierte Zelle ein Formularfeld, um zu gewährleisten, dass der Text an der richtigen Stelle eingegeben wird. Der Dokumentschutz stellt sicher, dass die Kalenderstrukturen nicht unbeabsichtigt geändert werden. Für die Ausgabe auf einen Drucker kann die Feldschattierung ausgeschaltet werden.

Als Benutzerschnittstelle für die Eingabe der Monats- und Jahresangaben dient entweder eine InputBox oder eine UserForm mit Kalender-Steuerelement, wie in Abbildung 20.2 dargestellt. Nach Bestätigung durch den Benutzer wird die Eingabe auf Gültigkeit geprüft. Weiter rechnet der Code aus, an welchem Wochentag der Monat beginnt und wie viele Tage er hat. Darauf basierend wird eine Tabelle erstellt und mit Tagesnamen und -daten gefüllt.

Abbildg. 20.1 Ein Kalenderblatt mit Textformularfeldern zur Eingabe von Terminen und Notizen

Sonntag	Montag	Dienstag	Mittwoch	Donnerstag	Freitag	Samstag
1 Neujahrstfest in der Stadt mit Max und Moritz	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				
Notizen:				Januar 2006		

Abbildg. 20.2 Eine InputBox oder eine UserForm mit Kalender-Steuerelement nimmt die Benutzereingaben auf.

Microsoft Word

Datum im Format 'Monat Jahr' eingeben:

OK

Abbrechen

Jan-2006

Kalender erstellen

Jan 2006

Jan 2006

Datum übernehmen

Abbrechen

Leere Zellen am Monatsanfang oder -ende werden verbunden, um eine Fläche für allgemeine Notizen bereitzustellen. Zuletzt werden die Schrift-, Rahmen und Schattierungsformatierungen zugewiesen.

**HINWEIS**

Das in diesem Beispiel verwendete Kalender-Steuerelement, *MSCAL.ocx*, gehört zum Lieferumfang von Microsoft Access aus Office Professional und muss explizit installiert werden. Sie dürfen es nur dort einsetzen, wo Office Professional installiert ist; es darf nicht frei weitergereicht werden. Wo dieses Steuerelement nicht vorhanden ist, müssen Sie eine InputBox oder andere Methode verwenden.

## Diskussion zum Code

Die Prozedur *KalenderErstellen* blendet die Schnittstelle für die Benutzereingabe ein. Die Zeichenkette aus dem Eingabefeld (InputBox) wird mit der Funktion *CVDate* in ein Datum umgewandelt (Date ist ein Datentyp wie Long oder String), woraus der Wochentag für den ersten Tag des Monats ermittelt wird. Dabei entspricht »1« dem Sonntag – erster Tag bzw. erste Spalte im Kalenderblatt. Eine Kombination der Funktionen *DateDiff* und *DateAdd* berechnet die Anzahl der Tage eines Monats. *DateDiff* kalkuliert die Anzahl der Tage (»d«), die zwischen dem eingegebenen Datum und diesem Datum plus einem Monat liegen.

Danach wird für den Kalender ein neues Dokument erstellt und im DIN A4-Querformat mit ein Zentimeter breiten Seitenrändern eingerichtet. Sie können auch ein beliebiges anderes Papierformat wählen. Im Beispielmakro passt sich die Tabelle dem Papier an. Wird sie dadurch für das von Ihnen gewählte Papier zu klein, müssen Sie unter Umständen die Schriftgröße der ersten Zeile etwas verringern.

Die Funktion *KalenderTabelleEinfügen* wird nun aufgerufen, um die Grundtabelle samt Wochentagen einzufügen. Sie gibt diese Tabelle als Objektvariable zurück; die Formatierung wird damit getrennt vorgenommen. In *KalenderTabelleEinfügen* sehen Sie, wie die angesprochene Anpassung an die Papiergröße erfolgt. Der Parameter *DefaultTableBehavior:=wdWord8TableBehavior* ist unabdingbar, um automatisch gleich breite Spalten über die ganze Seitenbreite zu erhalten, die sich in der Breite dem Inhalt nicht anpassen. Um die passende Zeilenhöhe zu bekommen, wird die Papierhöhe minus der Randbreite durch die Anzahl der Zeilen dividiert. Achten Sie auf die *HeightRule*-Eigenschaft für Tabellenzeilen: Sie müssen diese auf *wdRowHeightExactly* oder *wdRowHeightAtLeast* setzen, um die Einstellung für die Zeilenhöhe wirksam zu machen.

**HINWEIS**

Die Wochentage werden automatisch in der Standardsprache von Windows in den Kalender eingefügt.

Zurück in *KalenderErstellen*, werden die Monattage nummeriert. Die Prozedur fängt mit dem Wochentag an, der dem ersten des Monats entspricht, und schleift für jeden Tag im Monat durch die Tabellenzellen. In jeder Zelle wird die Tageszahl fett formatiert, gefolgt von einem Formularfeld in einer neuen Zeile. Das Formularfeld wird umbenannt, so dass es mit dem Datum bezeichnet ist (MMM\_DD\_YYYY). Dies erleichtert die Datenübertragung zwischen Word und anderen Anwendungen wie Outlook, falls Sie die Funktionalität ausbauen möchten.

Wenn alle Monate 28 Tage hätten, wäre die Erstellung eines Kalenders direkter: Alles würde immer in fünf Zeilen passen. Es gibt aber immer Ausreißer, die uns das Leben schwer machen: Monate mit 30 oder 31 Tagen, die an einem Freitag oder Samstag beginnen. Um in solchen Fällen Platz in der letzten Zeile zu schaffen, wird die Notizfläche in die zweite Tabellenzeile versetzt. Egal wo sie sich befindet, werden vier Zellen für diese Fläche verbunden – ebenso für den Monatsnamen, unten



rechts. Außerdem werden Textmarken gesetzt, damit das Makro später auf diese Zellen zurückgreifen kann.

Jetzt kommt die Formatierung dran. Wenn die Tabellenformatvorlage in Ihrer Dokumentvorlage bereits vorhanden wäre, könnten Sie diese Aufgabe mit einer einzigen Befehlszeile erledigen. Sie müssten lediglich die Anweisung

```
tbl.Style = TabelleFormatvorlage(doc)
```

mit

```
tbl.Style = "Name der Tabellenformatvorlage"
```

ersetzen. Nun soll dieses Beispiel auch die Erstellung einer Tabellenformatvorlage veranschaulichen, also ...

Die Tabellenformatvorlage wird von der Prozedur *TabelleFormatvorlage* erstellt. Um eine neue Formatvorlage des Typs »Tabelle« zu bestimmen, genügt es, dem Parameter *Type* der *Add*-Methode den Konstantenwert *wdStyleTypeTable* zu übergeben. Die allgemeinen Attribute wie Schriftart und -größe werden wie für jede Formatvorlage oder wie für einen Bereich festgelegt. Auch die Tabelleneigenschaften, wie Rahmen oder Schattierungen, werden nicht anders gehandhabt als für die Formatierung einer Tabelle.

Unterschiedlich ist jedoch die Definition der AutoFormat-spezifischen Teile: oberste und unterste Zeile (*wdFirstRow* bzw. *wdLastRow*), Eckzellen (*wdCellSE*) und Streifen (*wdEvenRowBanding*). Diese werden durch die neue Eigenschaft *Condition* bestimmt.

#### WICHTIG

Die Streifen kommen in VBA erst zum Vorschein, wenn Sie einen Wert für die Eigenschaft *RowStripe* bzw. *ColumnStripe* setzen. In der Benutzeroberfläche übernimmt Word diese Aufgabe automatisch, sobald eine Formatierung für ungerade oder gerade Zeilen oder Spalten festgelegt wurde.

Die Festlegung der Rahmenlinien ist der heikelste Teil der Formatvorlagendefinition, egal ob sie in der Benutzeroberfläche oder über VBA erfolgt. Fangen Sie unbedingt mit den Einstellungen für die ganze Tabelle an, gefolgt von den Randreihen und -spalten und schließlich den Eckzellen. Sobald ein Element angesprochen wird, überschreiben seine Attribute die geerbten der »höheren« Stufe. Deshalb werden alle *.Borders* für jeden Tabellenteil einzeln und ausdrücklich gesetzt.

Formatierungen des Layouts können nicht über eine Tabellenformatvorlage bestimmt werden, weshalb wir sie anschließend in *KalenderErstellen* vornehmen. Zum Schluss folgen Feinarbeiten am eigentlichen Dokument. Die letzte Absatzmarke, die einen Seitenumbruch verursacht, kann nicht gelöscht werden, weil sie seit Word 2000 Informationen über die Tabellen-Positionierung speichert. Formatieren wir sie jedoch mit der Schriftgröße »1 Punkt«, bleibt sie auf der gleichen Seite mit der Tabelle.

Falls Sie es vorziehen, keine Formularfelder in die Tabelle einzufügen, können Sie die vorletzte Zeile des Makros löschen, da der Dokumentschutz nicht benötigt wird.

**Listing 20.1** Den Kalender erstellen

```

Option Explicit

Sub KalenderErstellen()
    Dim doc As Word.Document
    Dim tbl As Word.Table
    Dim ffld As Word.FormField
    Dim rng As Word.Range
    Dim cel As Word.Cell
    Dim strDatum As String
    Dim lWochenTag As Long
    Dim lAnzTage As Long
    Dim dat As Date
    Dim sngCM1 As Single
    Dim lZaehler As Long
    Dim frm As frmKalender

    sngCM1 = CentimetersToPoints(1)
    Application.ScreenUpdating = False

    'Mit UserForm und Kalender-Steuerelement
    ' Set frm = New frmKalender
    ' frm.Show
    ' If frm.Tag = "Abbrechen" Then
    '     Unload frm
    '     Exit Sub
    ' End If
    ' strDatum = CStr(frm.cal.Value)
    ' Debug.Print strDatum
    ' Unload frm

    'Schleifen, bis die Eingabe ein gültiges Datum ist.
    Do
        strDatum = InputBox("Datum im Format 'Monat Jahr' eingeben:")
        If strDatum = "" Then Exit Sub
    Loop While Not IsDate(strDatum)

    dat = CDate(strDatum)
    lWochenTag = Weekday(dat)
    lAnzTage = DateDiff("d", dat, DateAdd("m", 1, dat))

    'Dokument einfügen und formatieren.
    Set doc = Documents.Add
    With doc.PageSetup
        .PaperSize = wdPaperA4
        .Orientation = wdOrientLandscape
        .RightMargin = sngCM1
        .LeftMargin = sngCM1
        .TopMargin = sngCM1
        .BottomMargin = sngCM1
    End With

    Set tbl = KalenderTabelleEinfügen(doc)
    'In der Zelle anfangen, die dem Wochentag des ersten Tags des Monats entspricht.
    tbl.Rows(2).Cells(lWochenTag).Select
    'Die Tage nummerieren und ein Formularfeld für die Benutzereingabe einfügen.
    For lZaehler = 1 To lAnzTage

```

Listing 20.1 Den Kalender erstellen (Fortsetzung)

```

Selection.Collapse wdCollapseStart
Selection.Font.Bold = True
Selection.TypeText lZaehler
Selection.Font.Bold = False
Selection.TypeText Chr$(11)
Set fflD = Selection.FormFields.Add(
    Range:=Selection.Range, Type:=wdFieldFormTextInput)
'Formularfeld mit dem Datum benennen.
fflD.Name = Format(dat, "MMM_DD_YYYY")
Selection.Cells(1).Range.Next(wdCell, 1).Select
Next lZaehler

'Wenn ein Monat (außer Feb) an einem Freitag oder Samstag beginnt,
'stehen die Notizen oben statt unten links.
If (lWochenTag = 6 Or lWochenTag = 7) And lAnzTage >= 30 Then
    'Zelle verbinden, um Platz für Notizen zu schaffen.
    Set cel = tbl.Cell(2, 1)
    cel.Merge MergeTo:=tbl.Cell(2, 5)
    cel.Range.Bookmarks.Add Name:="Notizen", Range:=cel.Range
    'Monatsbezeichnung steht unten rechts.
    Set cel = tbl.Cell(tbl.Rows.Count, 5)
    cel.Merge MergeTo:=tbl.Cell(tbl.Rows.Count, 7)
    doc.Bookmarks.Add Name:="Monat", Range:=cel.Range
Else
    'Zelle verbinden, um Platz für Notizen zu schaffen.
    Set cel = tbl.Cell(tbl.Rows.Count, 1)
    cel.Merge MergeTo:=tbl.Cell(tbl.Rows.Count, 4)
    cel.Range.Bookmarks.Add Name:="Notizen", Range:=cel.Range
    'Monatsbezeichnung steht unten rechts.
    Set cel = tbl.Cell(tbl.Rows.Count, 2)
    cel.Merge MergeTo:=tbl.Cell(tbl.Rows.Count, 4)
    doc.Bookmarks.Add Name:="Monat", Range:=cel.Range
End If

'Formatvorlage im Dokument erstellen und der Tabelle zuweisen.
tbl.Style = TabelleFormatvorlage(doc)
'Monat und Jahr in die Zelle unten rechts einfügen.
With doc.Range.Bookmarks("Monat").Range
    .Text = Format(dat, "MMMM") & " " & Format(dat, "yyyy")
    'Die Formatvorlage kann die Eckzellen-Formatierung nicht anzeigen.
    'Weil die erste und letzte Spalte keine besondere Formatierung hat,
    'müssen wir die Zelle ausdrücklich formatieren.
    .Cells(1).Shading.BackgroundPatternColor = wdColorGray70
    .Font.Size = 24
    .Font.ColorIndex = wdWhite
    .ParagraphFormat.Alignment = wdAlignParagraphRight
    .Cells.VerticalAlignment = wdCellAlignVerticalBottom
End With

'Notizenzelle bezeichnen und Formularfeld einfügen.
Set rng = doc.Bookmarks("Notizen").Range
rng.Text = "Notizen: "
rng.Collapse wdCollapseEnd
rng.MoveEnd wdCharacter, -1
doc.FormFields.Add Range:=rng, Type:=wdFieldFormTextInput

```

**Listing 20.1** Den Kalender erstellen *(Fortsetzung)*

```

'Erste Reihe mit den Tagesnamen formatieren.
'Über einer Formatvorlage können wir die vertikale Ausrichtung nicht bestimmen.
tbl.Rows(1).Cells.VerticalAlignment = wdCellAlignVerticalCenter
'Die Höhe etwas verringern, sodass die Tabelle nicht über die Seite bricht.
tbl.Rows(1).Height = tbl.Rows(1).Height - 10

'Aus dem gleichen Grunde die letzte Absatzmarke klein formatieren.
doc.Paragraphs.Last.Range.Font.Size = 1
'Die ganze Seite anzeigen.
doc.ActiveWindow.Panes(1).Zooms(wdPrintView).PageFit = wdPageFitFullPage
'An Dokumentanfang springen.
Selection.HomeKey wdStory
'Das Dokument schützen, sodass die Formularfelder aktiv sind.
doc.Protect wdAllowOnlyFormFields
End Sub

Function KalenderTabelleEinfügen(doc As Word.Document) As Word.Table
    Dim tbl As Word.Table
    Dim cel As Word.Cell
    Dim lAnzZeilen As Long
    Dim lAnzSpalten As Long
    Dim lZaehler As Long

    lAnzZeilen = 7
    lAnzSpalten = 7
    Set tbl = doc.Tables.Add(Range:=doc.Range, NumRows:=lAnzZeilen, _
        NumColumns:=lAnzSpalten, DefaultTableBehavior:=wdWord8TableBehavior)
    tbl.Rows.HeightRule = wdRowHeightExactly
    With doc.PageSetup
        tbl.Rows.Height = (.PageHeight - .TopMargin - .BottomMargin) / lAnzZeilen
    End With
    For Each cel In tbl.Rows(1).Cells
        lZaehler = lZaehler + 1
        cel.Range.Text = Format(Weekday(lZaehler), "dddd")
    Next cel
    Set KalenderTabelleEinfügen = tbl
End Function

Function TabelleFormatvorlage(doc As Word.Document) As String
    Dim sty As Word.Style
    Dim strTFV As String, strFVFont

    strTFV = "KalenderFV"
    strFVFont = "Arial"
    Set sty = doc.Styles.Add(Name:=strTFV, Type:=wdStyleTypeTable)
    sty.Font.Name = strFVFont
    sty.Font.Size = 9
    With sty.Table
        .Borders.OutsideLineStyle = wdLineStyleDouble
        .Borders.OutsideLineStyle = wdLineStyleSingle
        With .Condition(wdEvenRowBanding)
            .Shading.Texture = wdTextureNone
            .Shading.BackgroundPatternColor = wdColorGray10
            .Borders(wdBorderLeft).LineStyle = wdLineStyleDouble
            .Borders(wdBorderRight).LineStyle = wdLineStyleDouble
        End With
    End With
End Function

```

Listing 20.1 Den Kalender erstellen (Fortsetzung)

```

        .Borders(wdBorderBottom).LineStyle = wdLineStyleSingle
        .Borders(wdBorderTop).LineStyle = wdLineStyleSingle
    End With
    With .Condition(wdFirstRow)
        .Shading.BackgroundPatternColor = wdColorGray70
        .Font.Size = 16
        .Font.ColorIndex = wdWhite
        .Font.Bold = True
        .Borders(wdBorderBottom).LineStyle = wdLineStyleSingle
        .Borders(wdBorderLeft).LineStyle = wdLineStyleDouble
        .Borders(wdBorderRight).LineStyle = wdLineStyleDouble
        .Borders(wdBorderTop).LineStyle = wdLineStyleDouble
    End With
    With .Condition(wdFirstColumn)
        .Font.Size = 9
    End With
    With .Condition(wdLastColumn)
    End With
    With .Condition(wdLastRow)
        .Shading.BackgroundPatternColor = wdColorAutomatic
        .Borders(wdBorderBottom).LineStyle = wdLineStyleDouble
    End With
    With .Condition(wdSECell)
        .Shading.BackgroundPatternColor = wdColorGray70
        .Font.Size = 24
        .Font.ColorIndex = wdWhite
        .ParagraphFormat.Alignment = wdAlignParagraphRight
        .Borders(wdBorderBottom).LineStyle = wdLineStyleDouble
        .Borders(wdBorderRight).LineStyle = wdLineStyleDouble
    End With
    .RowStripe = 1
End With

TabelleFormatvorlage = strTFV
End Function

```



Die Beispieldatei *Bsp20\_01.dot* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap20*.

## Zusammenfassung

Anhand des Lösungsvorschlags, wie ein Online-Kalenderblatt zu erstellen ist, wurden einige Aspekte des Word-Objektmodells näher vorgestellt. Unter anderen wurden hervorgehoben:

- Tabellen und Tabellenformatvorlagen
- Der Umgang mit einigen Date-Funktionen der VB-Sprache
- Formularfelder und der Dokumentschutz



## Kapitel 21

# Aktive Symbolleisten

### In diesem Kapitel:

Symbolleisten mit dynamischen Symbolleistenschaltflächen	772
Zusammenfassung	777

In Kapitel 6 werden die Grundlagen zum Word-Objektmodell vorgestellt. Das Kapitel 7 befasst sich unter anderem mit dem Einbinden von Applikationsereignissen in das Projekt. Dieses Lösungsbeispiel baut auf den nachfolgenden Objekten und Ereignissen auf.

---

**Im Blickpunkt:**

API-Funktion

GetKeyState

Word-Ereignisse

DocumentChange, WindowSelectionChange

Word-Objekte

CommandBars.Controls

ActiveDocument.PrintOut, ActiveDocument.AutoHyphenation

Application.StatusBar

Selection.Type, Selection.ShapeRange, Selection.InlineShapes, Selection.Tables

---

Im Gegensatz zu den integrierten Symbolleisten von Word sind die benutzerdefinierten Symbolleisten passiv. Dies bedeutet, dass die einzelnen Symbolleistenschaltflächen, ohne zutun des Entwicklers, stets aktiviert sind. Dies ist auch dann der Fall, wenn die hinterlegten Funktionen gar nicht zum Kontext der Einfügemarke passen und somit eigentlich nicht gestartet werden können.

Dieses Lösungsbeispiel zeigt, wie aus einer passiven Symbolleiste eine dem Kontext entsprechende Symbolleiste aufgebaut wird, deren Status der einzelnen Symbolleistenschaltflächen sich dynamisch verändert. Die einzelnen Symbolleistenschaltflächen werden in Abhängigkeit von der Position der Einfügemarke aktiviert bzw. deaktiviert.

## Symbolleisten mit dynamischen Schaltflächen

Das vorliegende Beispiel baut auf einer statischen Symbolleiste auf, die in eine Dokumentvorlage eingebunden wurde. Die Dokumentvorlage wird anschließend als Add-In geladen. Bei Bedarf kann die Symbolleiste auch dynamisch beim Starten von Word angelegt werden. Dies ist in Kapitel 15 beschrieben. Das Einbinden einer Dokumentvorlage als Add-In wird in Kapitel 1 näher erläutert.

In der Datei *Bsp21\_Symbolleiste.dot* ist neben der eigentlichen Symbolleiste die Logik zur Steuerung der kontextabhängigen Symbolleistenschaltflächen und die eigentlichen Makros hinterlegt.

**Abbildg. 21.1** Symbolleiste *DasHandbuch\_Bsp21* mit aktivierten bzw. deaktivierten Symbolleistenschaltflächen





**HINWEIS**

Damit die integrierte Symbolleiste in allen Dokumenten zur Verfügung steht, muss die Datei *Bsp21\_Symbolleiste.dot* in den *Startup*-Ordner von Word kopiert werden. Nach dem erneuten Aufruf der Applikation steht die Symbolleiste automatisch am rechten Rand des Programmfensters zur Verfügung.

## Zusammenspiel der einzelnen Module

Um auf Ereignisse reagieren zu können, muss eine entsprechende Klasse in das Projekt eingebunden werden. Eine neue Instanz dieser Klasse wird dann beim Starten von Word angelegt. In dieser Klasse wird die »Überwachung« der Ereignisse bearbeitet.

### Klassenmodul

Damit die Symbolleiste mit aktiven Symbolleistenschaltflächen versehen werden kann, muss das Add-In auf die Ereignisse der Applikation reagieren können. In Kapitel 7 wird gezeigt, wie diese aktiviert werden können.

### Das Programmmodul


Innerhalb des Programmmoduls muss an geeigneter Stelle sichergestellt werden, dass eine neue Instanz der entsprechenden Klasse mit den hinterlegten Applikationsereignissen angelegt wird. Wie eine neue Instanz auf diese Klassen erzeugt wird, ist in Kapitel 7 detailliert beschrieben.

Um beim Laden des Add-Ins die automatische Überwachung der Ereignisse zu gewährleisten, ist ein Makro mit der Bezeichnung *AutoExec* implementiert. Dieses wird automatisch von Word abgearbeitet, sobald das Add-In aktiviert wird. Die Funktionsweise der Auto-Makros ist in Kapitel 5 behandelt.

### Die Ereignisse

Durch das Einbinden des Schlüsselworts *WithEvents* in die Deklaration der globalen Variable innerhalb des Klassenmoduls werden die Ereignisse aus einer ActiveX-Komponente überwacht. In unserem Fall ist dies die Komponente *Word.Application*.

Damit eine aktive Symbolleiste erzeugt werden kann, sind in erster Line zwei Ereignisse von Bedeutung:

- Das *DocumentChange*-Ereignis wird ausgelöst, wenn eine Datei geöffnet oder geschlossen wird. Es wird auch ausgelöst, wenn zwischen zwei verschiedenen Dokumenten hin und her gewechselt wird.
- Das *WindowSelectionChange*-Ereignis wird ausgelöst, wenn die Einfügemarke verschoben wird. Dies ist in erster Linie dann der Fall, wenn die Einfügemarke mittels Pfeiltasten oder eines Mausklicks verschoben wird. Ein Wechseln der Zellen innerhalb einer Tabelle mittels der -Taste löst das Ereignis ebenfalls aus.

Das Ereignis wird jedoch nicht ausgelöst, wenn Text erfasst wird. Aus Sicht von Word wird in diesem Fall die Einfügemarke nicht verschoben, da diese immer *vor* dem gleichen Zeichen steht.

Diese beiden Ereignisse reichen aus, um die Symbolleiste »zum Leben zu erwecken« und die einzelnen Symbolleistenschaltflächen dem Kontext der Einfügemarke entsprechend zu aktivieren bzw. zu deaktivieren.

## Diskussion zum Code

Die eigentliche Funktionalität hinter den einzelnen Symbolleistenschaltflächen wurde dreistufig aufgebaut:

- Die erste Stufe ist die allgemeine Prozedur, die den aktuellen Kontext der Einfügemarke auswertet und die Symbolleistenschaltflächen entsprechend aktiviert bzw. deaktiviert.
- Die zweite Stufe dient der Fehlerbehandlung. Hier wird überprüft, ob die entsprechende Symbolleistenschaltfläche zu Recht aktiviert war. Wurden alle Prüfungen erfolgreich bestanden, wird das eigentliche Makro aufgerufen.
- In der dritten Stufe ist das eigentliche Makro, also die gewünschte Funktionalität, der Symbolleistenschaltfläche hinterlegt.

Durch das dreistufige Konzept können die einzelnen Stufen problemlos ergänzt oder angepasst werden, ohne dass dies einen Einfluss auf die restliche Funktionalität des Add-Ins hat.

So kann beispielsweise die Funktion zum Ausdrucken eines Entwurfs erweitert werden, indem die Papierzufuhr beim Ausdruck aus einem bestimmten Papierschacht erfolgt. Auf die vorherigen Prüfungen, ob überhaupt ein Ausdruck erfolgen kann, hat dies keinen Einfluss.

---

**HINWEIS** Die Fehlerbehandlung in der zweiten Stufe wird bewusst eingebaut, da die einzelnen Ereignisse nicht in jedem Fall so ausgeführt werden, wie sich dies der Programmierer wünscht.

Als besonderer Stolperstein kann sich ab und zu das `WindowSelectionChange`-Ereignis erweisen, denn dieses wird beim Erfassen des Textes nicht ausgelöst.

---

### Kontext der Einfügemarke auswerten – Stufe 1

In Listing 21.1 wird der Kontext der Einfügemarke analysiert. Für jeden benötigten Fall wird eine eigene Variable gesetzt. Am Schluss der Prozedur wird den einzelnen Symbolleistenschaltflächen die zugehörige Kombination von Variablen zugewiesen.

**Listing 21.1** Prozedur zum Auswerten des Kontextes der Einfügemarke und Definieren der Symbolleistenschaltflächen

```
Private Sub procSchaltflächeAktivieren()
    Dim objCContext As Object
    Dim bDocumentOffen As Boolean
    Dim bDocumentOhneSchutz As Boolean
    Dim bTabelleMarkiert As Boolean
    Dim bShapeMarkiert As Boolean
    Dim bInlineShapeMarkiert As Boolean

    'Ist ein Document geöffnet?
    If (Application.Windows.Count > 0) Then
        bDocumentOffen = True

    'Ist Document geschützt?
    If (ActiveDocument.ProtectionType = wdNoProtection) Then
        bDocumentOhneSchutz = True
    End If

    'Ist eine Tabelle markiert?
```

**Listing 21.1** Prozedur zum Auswerten des Kontextes der Einfügemarke und Definieren der Symbolleistenschaltflächen (Fortsetzung)

```

    If (Selection.Tables.Count > 0) Then
        bTabelleMarkiert = True
    End If

    'Ist ein Shape markiert?
    If Selection.Type = wdSelectionShape Then
        bShapeMarkiert = True
    End If

    'Ist ein InlineShape markiert?
    If Selection.Type = wdSelectionInlineShape Then
        bInlineShapeMarkiert = True
    End If
End If

'Schaltflächen ein-/ausschalten
Set objCContext = Application.CustomizationContext
Application.CustomizationContext = ThisDocument

With CommandBars(cbrSYMBOLLEISTE).Controls
    .Item(cbbDOKUMENT_DRAFT_DRUCK).Enabled = bDocumentOffen
    .Item(cbbDOKUMENT_FELDFUNKTION).Enabled = bDocumentOffen And bDocumentOhneSchutz
    .Item(cbbDOKUMENT_SILBENTRENNUNG).Enabled = bDocumentOffen And bDocumentOhneSchutz
    .Item(cbbTABELLE_FORMATIEREN).Enabled = bDocumentOffen And bDocumentOhneSchutz _
        And bTabelleMarkiert
    .Item(cbbSHAPE_BREITE_SETZEN).Enabled = bDocumentOffen And bDocumentOhneSchutz _
        And (bShapeMarkiert Or bInlineShapeMarkiert)
    .Item(cbbSHAPE_VERANKERN).Enabled = bDocumentOffen And bDocumentOhneSchutz _
        And bShapeMarkiert
End With
Application.CustomizationContext = objCContext

'Add-In als gespeichert kennzeichnen
ThisDocument.Saved = True
End Sub

```

**PROFITIPP**

Ist in einem Add-In eine Symbolleiste eingebunden, wird manchmal beim Beenden von Word die Rückfrage gestellt, ob das entsprechende Add-In gespeichert werden soll. Dies geschieht vor allem dann, wenn, wie in unserem Beispiel, durch das Programm selber oder durch den Anwender die Symbolleiste geändert wurde.

Um diese Rückfrage zu verhindern, wird nach erfolgter Manipulation der Symbolleiste das Add-In als bereits gespeichert gekennzeichnet:

```
ThisDocument.Saved = True
```

Um sicherzustellen dass die besagte Rückfrage unter keinen Umständen gestellt wird, wurde zusätzlich ein Makro mit der Bezeichnung *AutoExit* eingebaut. Dieses Makro wird automatisch abgearbeitet, sobald Word beendet wird.

Wird das Add-In vom Entwickler geändert und er beendet Word, ohne das Add-In zuvor manuell gespeichert zu haben, gehen sämtliche Änderungen verloren, da das erwähnte Makro die erwartete Rückfrage unterdrückt und die Datei geschlossen wird, ohne es zu speichern.

**Mögliche Fehlerbehandlung – Stufe 2**

In Listing 21.2 ist eine Programmsequenz aufgezeigt, die für die Fehlerbehandlung einer einzelnen Symbolleistenschaltfläche zuständig ist. Das Makro *ShapeVerankern* wurde der Symbolleistenschaltfläche zugewiesen. Für jede Symbolleistenschaltfläche wurde eine eigene Prozedur, mit ähnlicher Funktionalität, aufgebaut.

**Listing 21.2** Fehlerprüfung für die Symbolleistenschaltfläche *Autoform verankern ein/aus*

```
Public Sub ShapeVerankern()
    Dim cbtn As Office.CommandBarButton

    Set cbtn = CommandBars(cbrSYMBOLLEISTE).Controls.Item(cbbSHAPE_VERANKERN)
    cbtn.Enabled = False

    If Windows.Count = 0 Then
        Beep
    ElseIf Not (ActiveDocument.ProtectionType = wdNoProtection) Then
        MsgBox "Aktuelles Dokument ist geschützt." & vbCr & _
            "Verankerung der Shape-Objekte kann nicht geändert werden.", _
            vbOKOnly + vbExclamation, "Warnung"
    ElseIf (Not Selection.Type = wdSelectionShape) Then
        Beep
    Else
        fktShapeVerankern
        cbtn.Enabled = True
    End If
End Sub
```

Nur wenn die Prüfung auf mögliche Fehler erfolgreich war, wird die eigentliche Funktion, welche die markierten Shape-Objekte verankert, aufgerufen und die zugehörige Symbolleistenschaltfläche wieder aktiviert:

```
fktShapeVerankern
cbtn.Enabled = True
```

**Ausführen der gewünschten Funktion – Stufe 3**

In Listing 21.3 wird der letzte Schritt ausgeführt. In dieser Funktion wurde die eigentliche Programmsequenz für die Symbolleistenschaltfläche hinterlegt.

**Listing 21.3** Die Programmzeilen zur Symbolleistenschaltfläche *Autoform verankern ein/aus*

```
Public Function fktShapeVerankern()
    Dim shp As Word.Shape
    Dim bFlag As Boolean

    bFlag = True
    For Each shp In Selection.ShapeRange
        bFlag = bFlag And shp.LockAnchor
    Next shp

    Selection.ShapeRange.LockAnchor = Not bFlag
```

Listing 21.3 Die Programmzeilen zur Symbolleistenschaltfläche *Autoform verankern ein/aus* (Fortsetzung)

```
Application.StatusBar = "Verankerung bei " & CStr(fktShapesInShapeRangeZählen) & _
    " Shape-Objekten auf '" & CStr(Not bFlag) & "' gesetzt."
End Function
```

Die Funktion *fktShapeVerankern* verhält sich ähnlich wie andere Standardfunktionen von Word (beispielsweise markierten Text auf Fett setzen). Mittels einer Schleife wird zuerst der Status aller markierten Shape-Objekte ermittelt und in der Variable *bFlag* zwischengespeichert. Solange bei allen Objekten die Verankerung aktiviert ist, bleibt *bFlag* auf *True*. Ist jedoch nur bei einem einzelnen Objekt die Verankerung nicht aktiviert, so wird *bFlag* auf *False* gesetzt:

```
For Each shp In Selection.ShapeRange
    bFlag = bFlag And shp.LockAnchor
Next shp
```

In einem zweiten Schritt wird der umgekehrte Wert der *LockAnchor*-Eigenschaft zugewiesen. Auf diese Weise ist sichergestellt, dass allen markierten Shape-Objekten der gleiche Status der *LockAnchor*-Eigenschaft zugewiesen wird:

```
Selection.ShapeRange.LockAnchor = Not bFlag
```

**HINWEIS**

Bei den aufgeführten Listings handelt es sich nur um eine Auswahl von Programmsequenzen, die benötigt werden, um die Funktionalität der sechs Symbolleistenschaltflächen zu ermöglichen.

Es würde jedoch den Rahmen dieses Buches sprengen, wenn jede Prozedur abgebildet und detailliert erläutert würde. Die restlichen Programmzeilen können direkt in der Beispieldatei *Bsp21\_Symbolleiste.dot* eingesehen werden.



Das dargestellte Beispiel mit der aktiven Symbolleiste finden Sie in der Beispieldatei *Bsp21\_Symbolleiste.dot* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap21*.

## Zusammenfassung

In diesem Lösungsbeispiel wurde gezeigt, wie Ereignisse genutzt werden können, um die Schaltflächen einer Symbolleiste dem Kontext entsprechend aktivieren bzw. deaktivieren zu können:

- Es wurde dargestellt, wie ein Klassenmodul aufgebaut sein muss, um auf die Ereignisse reagieren zu können, und wie eine Instanz dieser Klasse in das Projekt eingebunden wird (Seite 773).
- Außerdem wurde erklärt, weshalb es sinnvoll ist, die eigentliche Funktionalität von einer zusätzlichen Fehlerbehandlung zu entkoppeln (Seite 774).



## Kapitel 22

# Aktualisierung der Felder mit einem Fortschrittsbalken visualisieren

### In diesem Kapitel:

Aktualisieren aller Felder eines Dokumentes	780
Erstellen und Einbinden eines Fortschrittsbalkens	791
Einbinden des Microsoft ProgressBar-Controls	795
Zusammenfassung	796

Dieses Kapitel zeigt Ihnen, wie Sie das Aktualisieren aller Felder in einem Dokument mit Hilfe eines Fortschrittsbalkens und des *ProgressBar*-Steuerelementes synchronisiert visualisieren können.

---

**Im Blickpunkt:**

Word-Objekte, Eigenschaften und Methoden

ActiveDocument.StoryRanges-Objekt

Range.StoryType-Eigenschaft

Range.NextStoryRange-Eigenschaft

Range.Fields-Eigenschaft

Fields.Update-Methode

Fields.Locked-Methode

ActiveDocument.Sections-Eigenschaft

Section.Headers-Eigenschaft

Section.Footers-Eigenschaft

ActiveDocument.Shapes-Objekt

Shape.TextFrame-Objekt

Benutzerformular-Steuerelemente, Ereignisse, Eigenschaften und Methoden

Label-Steuerelement

Label.With-Eigenschaft

Label.SpecialEffect-Eigenschaft

Label.MouseUp-Ereignis

Label.MouseDown-Ereignis

ListBox-Steuerelement

ListBox.AddItem-Methode

ProgressBar-Steuerelement

---

## Aktualisieren aller Felder eines Dokumentes

In einem Word-Dokument lässt sich mittels Felder eine Vielzahl von Informationen erfassen und darstellen. Ändern sich die Informationen, die in den Feldern angezeigt werden, müssen auch die Felder aktualisiert werden. Dabei wird zwischen den Feldern unterschieden, die sich automatisch aktualisieren, wie z.B. Querverweis-Felder, und Feldern, die manuell aktualisiert werden müssen, wie das Inhaltsverzeichnis.



Es ist durchaus sinnvoll, dass nicht alle Felder immer automatisch von Word aktualisiert werden, da bei umfangreichen Dokumenten mit Querverweisen und Verknüpfungen zu Grafiken das Arbeiten ansonsten fast unmöglich würde, wenn wirklich alle Felder jedes mal überprüft werden würden.

Das Aktualisieren der Felder, die sich nicht automatisch aktualisieren, muss vom Anwender initiiert werden. Dies kann z.B. durch das Speichern und wieder Öffnen des Dokumentes erfolgen, wenn im Menü *Extras/Optionen* auf der Registerkarte *Allgemein* die Option *Automatische Verknüpfungen beim Öffnen aktualisieren* aktiviert ist. Eine andere Möglichkeit besteht darin, in die Dokumentansicht *Seitenlayout* zu wechseln. Oder Sie aktualisieren die Felder manuell mittels der Taste **[F9]** für ein einzelnes Feld bzw. für alle Felder, wenn Sie vorher das gesamte Dokument markiert haben (z.B. mittels **[Strg]+[A]**).

Leider haben alle Möglichkeiten ihre Nachteile: Entweder werden nicht alle Felder wirklich aktualisiert oder das Aktualisieren ist zu umständlich (wenn das Dokument immer erst neu geöffnet werden muss).

Die häufigste verwendete Möglichkeit, das gesamte markierte Dokument mittels **[F9]** zu aktualisieren, hat den großen Nachteil, dass dabei nur die Felder im normalen Textbereich (dem Hauptdokumentbereich) berücksichtigt werden, die auch in der Normalansicht angezeigt werden. Alle Felder in Kopf- und Fußzeilen, Fußnoten oder in Textrahmen werden mit dieser Methode nicht angesprochen und somit aktualisiert.

## Das StoryRange-Objekt

Word bietet für den Zugriff auf die verschiedenen Bereiche eines Dokumentes (den Dokumentkomponenten: Kopf- und Fußzeilen, Fußnoten- und Endnotenbereich, Kommentarbereich und Hauptdokumentbereich) die StoryRanges-Eigenschaft eines Dokumentes. Diese liefert für das jeweilige Dokument eine Auflistung aller Dokumentkomponenten in Form einer StoryRanges-Auflistung zurück.



Die in diesem Abschnitt verwendeten Beispiele finden Sie auf der CD-ROM zum Buch in der Datei `\Beispiele\Kap22\Bsp22_01.doc` im Modul `modStoryRanges`.

In Word 2003 stehen Ihnen insgesamt folgende Dokumentkomponentenbereiche zur Verfügung:

**Tabelle 22.1** Übersicht über die Komponentenbereiche (StoryRanges) eines Dokumentes

StoryRanges-Typ	Komponentenbereich	Max. Anzahl	Beschreibung
wdMainTextStory	Haupttextbereich	1 pro Dokument	Der normale Textbereich. Dieser Bereich ist immer vorhanden.
wdFootnotesStory	Fußnotenbereich	1 pro Dokument	Gehört zum Haupttextbereich. Wird erstellt, sobald Sie eine Fußnote erstellen.
wdEndnotesStory	Endnotenbereich	1 pro Dokument	Gehört zum Haupttextbereich. Wird erstellt, sobald Sie eine Endnote erstellen.
wdCommentsStory	Kommentarbereich	1 pro Dokument	Gehört zum Haupttextbereich. Wird erstellt, sobald Sie einen Kommentar erstellen.

**Tabelle 22.1** Übersicht über die Komponentenbereiche (*StoryRanges*) eines Dokumentes (*Fortsetzung*)

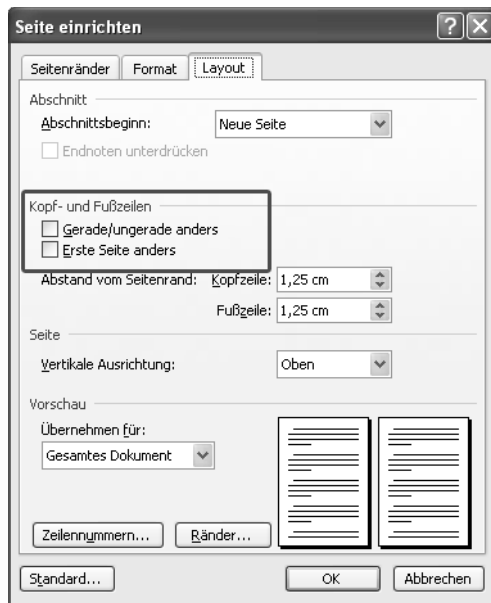
StoryRanges-Typ	Komponenten-bereich	Max. Anzahl	Beschreibung
wdTextFrameStory	Textrahmenbereich	n pro StoryRange	Textrahmen gehören zu den Shape-Objekten und können in allen StoryRange-Objekten vorkommen.
wdPrimaryHeaderStory	Hauptkopfzeilenbereich/Kopfzeilenbereich ungerade Seiten	1 pro Abschnitt	Wird erstellt, sobald Sie Text in die erste Kopfzeile eintragen.
wdEvenPagesHeaderStory	Kopfzeilenbereich gerade Seiten	1 pro Abschnitt	Wird erstellt, wenn Sie das Seitenlayout auf <i>Gerade/ungerade anders</i> einstellen und Text in die Kopfzeile einer geraden Seitenzahl eintragen.
wdPrimaryFooterStory	Hauptfußzeilenbereich/Fußzeilenbereich ungerade Seiten	1 pro Abschnitt	Wird erstellt, sobald Sie Text in die erste Fußzeile eintragen.
wdEvenPagesFooterStory	Fußzeile gerade Seiten	1 pro Abschnitt	Wird erstellt, wenn Sie das Seitenlayout auf <i>Gerade/ungerade anders</i> einstellen und Text in die Fußzeile einer geraden Seitenzahl eintragen.
wdFirstPageHeaderStory	Kopfzeile erste Seite	1 pro Abschnitt	Wird erstellt, sobald Sie das Seitenlayout auf <i>Erste Seite anders</i> ändern und Text in die Kopfzeile der ersten Seiten eintragen.
wdFirstPageFooterStory	Fußzeile erste Seite	1 pro Abschnitt	Wird erstellt, sobald Sie das Seitenlayout auf <i>Erste Seite anders</i> ändern und Text in die Fußzeile der ersten Seiten eintragen
wdFootnoteSeparatorStory	Fußnoten-Trennlinienbereich	1 pro Dokument	Gehört zum Haupttextbereich. Bei neu erstellten Dokumenten existiert dieser Bereich, sobald Sie entweder eine Fußnote einfügen oder in die Kopf- oder Fußzeilenansicht wechseln.
wdFootnoteContinuationSeparatorStory	Fußnoten-Fortsetzungstrennlinienbereich	1 pro Dokument	Gehört zum Fußnotenbereich. Dieser Bereich wird erstellt, sobald Sie entweder eine Fußnote einfügen oder in die Kopf- oder Fußzeilenansicht wechseln.
wdFootnoteContinuationNoticeStory	Fußnoten-Fortsetzungshinweisbereich	1 pro Dokument	Gehört zum Fußnotenbereich. Dieser Bereich wird erstellt, sobald Sie einen Fußnoten-Fortsetzungshinweis erfassen.
wdEndnoteSeparatorStory	Endnoten-Trennlinienbereich	1 pro Dokument	Gehört zum Haupttextbereich. Bei neu erstellten Dokumenten existiert dieser Bereich, sobald Sie entweder eine Fußnote einfügen oder in die Kopf- oder Fußzeilenansicht wechseln.

Tabelle 22.1 Übersicht über die Komponentenbereiche (*StoryRanges*) eines Dokumentes (*Fortsetzung*)

StoryRanges-Typ	Komponenten-bereich	Max. Anzahl	Beschreibung
wdEndnoteContinuationSeparatorStory	Endnoten-Fortsetzungstrennlinienbereich	1 pro Dokument	Gehört zum Haupttextbereich. Bei neu erstellten Dokumenten existiert dieser Bereich, sobald Sie entweder eine Fußnote einfügen oder in die Kopf- oder Fußzeilenansicht wechseln.
wdEndnoteContinuationNoticeStory	Endnoten-Fortsetzungshinweisbereich	1 pro Dokument	Gehört zum Endnotenbereich. Dieser Bereich wird erstellt, sobald Sie einen Endnoten-Fortsetzungshinweis erfassen.

Der Zugriff bzw. die Verfügbarkeit einzelner StoryRanges hängt davon ab, wie das Dokument eingerichtet ist. Speziell sind damit die Einstellungen gemeint, die Sie über den Menüpunkt *Datei/Seite einrichten* im gleichnamigen Dialogfeld *Seite einrichten* auf der Registerkarte *Layout* im Bereich *Kopf- und Fußzeilen* einstellen können (Abbildung 22.1).

Abbildg. 22.1 Einrichten eines Dokumentes mit unterschiedlichen Kopf- und Fußzeilen



Ein neues leeres Dokument, wie Sie es über den Menüpunkt *Datei/Neu* und dann im Aufgabenbereich über den Link *Leeres Dokument* erzeugen können, besitzt nur einen Komponentenbereich: den Haupttextbereich.

Sobald Sie das Seitenlayout auf *Gerade/ungerade anders* ändern, stehen Ihnen die StoryRange-Objekte wdPrimaryHeaderStory, wdEvenPagesHeaderStory, wdPrimaryFooterStory und wdEvenPagesFooterStory zur Verfügung. Sobald Sie in die Kopf- oder Fußzeilen Text eintragen, werden diese Bereiche im Dokument erstellt.

Wenn Sie das Seitenlayout auf *Erste Seite anders* ändern, stehen Ihnen die StoryRange-Objekte `wdFirstPageHeaderStory` und `wdFirstPageFooterStory` zur Verfügung und werden erstellt, sobald Sie in die entsprechenden Kopf-/Fußzeile Text eintragen.

Bei diesen StoryRange-Objekten handelt es sich um Dokumentkomponenten, die für jeden Abschnitt festgelegt werden können. Um nun alle Dokumentkomponenten eines bestimmten Typs in allen Abschnitten zu durchlaufen, steht Ihnen die Eigenschaft `NextStoryRange` der StoryRanges-Auflistung zur Verfügung. Mit dieser Eigenschaft können Sie prüfen, ob es weitere StoryRange-Objekte des jeweiligen Typs im Dokument gibt.

Das Beispiel in Listing 22.1 zählt alle Textfelder im Hauptdokumentbereich, indem es prüft, ob die `NextStoryRange`-Eigenschaft einen Verweis auf ein weiteres Textfeld zurückliefert.

**Listing 22.1** Durchlaufen aller Textfelder im Hauptdokumentbereich

```
Sub subNextTextFrameStory()
    Dim rngStory As Range
    Dim intTFStory As Integer: intTFStory = 0
    For Each rngStory In ActiveDocument.StoryRanges
        If rngStory.StoryType = wdTextFrameStory Then
            intTFStory = intTFStory + 1
            While Not (rngStory.NextStoryRange Is Nothing)
                Set rngStory = rngStory.NextStoryRange
                intTFStory = intTFStory + 1
            Wend
        End If
    Next rngStory
    MsgBox "Es gibt " & intTFStory & " Textfeld" & _
        IIf(intTFStory = 1, "", "er") & " im Hauptdokumentbereich", _
        vbInformation, "Anzahl Textfelder"
End Sub
```



Wenn Sie die Prozedur `subNextTextFrameStory` auf das Beispieldokument *Bsp22\_02.doc* auf der CD-ROM zum Buch im Ordner `\Beispiele\Kap22` anwenden, erhalten Sie die in Abbildung 22.2 gezeigte Ausgabe.

**Abbildg. 22.2** Anzeige der Textfeldanzahl im Dokument *Bsp22\_02.doc*



### WICHTIG

Das Ermitteln aller Textfelder in einem Bereich liefert nur die Anzahl der eigenständigen Textfelder zurück. Miteinander verknüpfte Textfelder (wie in der Beispieldatei *Bsp22\_02.doc*) werden dabei nur als ein einziges Textfeld berücksichtigt.

Die Besonderheit beim Umgang mit Textfeldern wird im Abschnitt »Textfelder« behandelt.

Tabelle 22.2 Übersicht über die von der *NextStoryRange*-Eigenschaft zurückgegebenen Dokumentkomponenten

Art der Dokumentkomponente	Dokumentkomponente, die von der <i>NextStoryRange</i> -Eigenschaft zurückgegeben wird
wdMainTextStory	Gibt immer <b>Nothing</b> zurück, da nur eine Komponente pro Dokument existiert.
wdFootnotesStory	Gibt immer <b>Nothing</b> zurück, da nur eine Komponente pro Dokument existiert.
wdEndnotesStory	Gibt immer <b>Nothing</b> zurück, da nur eine Komponente pro Dokument existiert.
wdCommentsStory	Gibt immer <b>Nothing</b> zurück, da nur eine Komponente pro Dokument existiert.
wdEvenPagesHeaderStory	Gibt die entsprechende Dokumentkomponente im nächsten Abschnitt zurück, sofern ein weiterer Abschnitt im Dokument existiert.
wdPrimaryHeaderStory	Gibt die entsprechende Dokumentkomponente im nächsten Abschnitt zurück, sofern ein weiterer Abschnitt im Dokument existiert.
wdEvenPagesFooterStory	Gibt die entsprechende Dokumentkomponente im nächsten Abschnitt zurück, sofern ein weiterer Abschnitt im Dokument existiert.
wdPrimaryFooterStory	Gibt die entsprechende Dokumentkomponente im nächsten Abschnitt zurück, sofern ein weiterer Abschnitt im Dokument existiert.
wdFirstPageHeaderStory	Gibt die entsprechende Dokumentkomponente im nächsten Abschnitt zurück, sofern ein weiterer Abschnitt im Dokument existiert.
wdFirstPageFooterStory	Gibt die entsprechende Dokumentkomponente im nächsten Abschnitt zurück, sofern ein weiterer Abschnitt im Dokument existiert.
wdTextFrameStory	Gibt den nächsten Textrahmen im Hauptdokumentbereich zurück, sofern ein weiterer im Dokument existiert.

Mit dieser Eigenschaft können Sie nun erstmals alle Dokumentkomponenten durchlaufen und Aktionen in den einzelnen Bereichen ausführen.

Das Listing 22.2 zeigt Ihnen für das aktuelle Dokument genau an, welche StoryRanges im Dokument vorhanden sind.



Das vollständige Listing mit den notwendigen Konstanten-Deklarationen finden Sie auf der CD-ROM zum Buch in der Datei `\Beispiele\Kap22\Bsp22_01.doc` im Modul `modStoryRanges`.

Listing 22.2 Anzeigen aller Dokumentkomponenten eines Dokumentes

```

Sub subListStoryRanges()
    Dim rngStory As Range
    Dim strStoryName As String
    Dim strMSG As String
    For Each rngStory In ActiveDocument.StoryRanges
        Select Case rngStory.StoryType
            Case wdMainTextStory
                strStoryName = "Haupttextbereich"
            Case wdFootnotesStory
                strStoryName = "Fußnotenbereich"
        End Select
    Next rngStory
End Sub

```

**Listing 22.2** Anzeigen aller Dokumentkomponenten eines Dokumentes (Fortsetzung)

```

Case wdFootnoteSeparatorStory
    strStoryName = "Fußnoten-Trennlinienbereich"
Case wdFootnoteContinuationSeparatorStory
    strStoryName = "Fußnoten-Fortsetzungstrennlinienbereich"
Case wdFootnoteContinuationNoticeStory
    strStoryName = "Fußnoten-Fortsetzungshinweisbereich"
Case wdEndnotesStory
    strStoryName = "Endnotenbereich"
Case wdEndnoteSeparatorStory
    strStoryName = "Endnoten-Trennlinienbereich"
Case wdEndnoteContinuationSeparatorStory
    strStoryName = "Endnoten-Fortsetzungstrennlinienbereich"
Case wdEndnoteContinuationNoticeStory
    strStoryName = "Endnoten-Fortsetzungshinweisbereich"
Case wdCommentsStory
    strStoryName = "Kommentarbereich"
Case wdTextFrameStory
    strStoryName = "Textrahmenbereich"
Case wdEvenPagesHeaderStory
    strStoryName = "Kopfzeilenbereich gerade Seiten"
Case wdPrimaryHeaderStory
    strStoryName = "Hauptkopfzeilenbereich/Kopfzeilenbereich ungerade Seiten"
Case wdEvenPagesFooterStory
    strStoryName = "Fußzeile gerade Seiten"
Case wdPrimaryFooterStory
    strStoryName = "Hauptfußzeilenbereich/Fußzeilenbereich ungerade Seiten"
Case wdFirstPageHeaderStory
    strStoryName = "Kopfzeile erste Seite"
Case wdFirstPageFooterStory
    strStoryName = "Fußzeile erste Seite"
End Select
strMSG = strMSG & strStoryName & vbCrLf
While Not (rngStory.NextStoryRange Is Nothing)
    Set rngStory = rngStory.NextStoryRange
    Select Case rngStory.StoryType
    Case wdMainTextStory
        strStoryName = "Haupttextbereich"
    Case wdFootnotesStory
        strStoryName = "Fußnotenbereich"
    Case wdFootnoteSeparatorStory
        strStoryName = "Fußnoten-Trennlinienbereich"
    Case wdFootnoteContinuationSeparatorStory
        strStoryName = "Fußnoten-Fortsetzungstrennlinienbereich"
    Case wdFootnoteContinuationNoticeStory
        strStoryName = "Fußnoten-Fortsetzungshinweisbereich"
    Case wdEndnotesStory
        strStoryName = "Endnotenbereich"
    Case wdEndnoteSeparatorStory
        strStoryName = "Endnoten-Trennlinienbereich"
    Case wdEndnoteContinuationSeparatorStory
        strStoryName = "Endnoten-Fortsetzungstrennlinienbereich"
    Case wdEndnoteContinuationNoticeStory
        strStoryName = "Endnoten-Fortsetzungshinweisbereich"
    Case wdCommentsStory
        strStoryName = "Kommentarbereich"
    
```

Listing 22.2 Anzeigen aller Dokumentkomponenten eines Dokumentes (Fortsetzung)

```

Case wdTextFrameStory
    strStoryName = "Textrahmenbereich"
Case wdEvenPagesHeaderStory
    strStoryName = "Kopfzeilenbereich gerade Seiten"
Case wdPrimaryHeaderStory
    strStoryName = "Hauptkopfzeilenbereich/Kopfzeilenbereich ungerade Seiten"
Case wdEvenPagesFooterStory
    strStoryName = "Fußzeile gerade Seiten"
Case wdPrimaryFooterStory
    strStoryName = "Hauptfußzeilenbereich/Fußzeilenbereich ungerade Seiten"
Case wdFirstPageHeaderStory
    strStoryName = "Kopfzeile erste Seite"
Case wdFirstPageFooterStory
    strStoryName = "Fußzeile erste Seite"
End Select
strMSG = strMSG & strStoryName & vbCrLf
Wend
Next rngStory
MsgBox strMSG
End Sub

```

In diesem Dokument, das dieses Kapitel beinhaltet, sind z.B. die in Abbildung 22.3 abgebildeten Dokumentkomponenten enthalten:

Abbildg. 22.3 Beispiel für die in einem Dokument enthaltenen Dokumentkomponentenbereiche



## Felder in den *StoryRange*-Objekten aktualisieren

In diesem Kapitel soll ja die Aktualisierung aller Felder in einem Dokument behandelt werden, sodass wir uns die Möglichkeiten der StoryRange-Objekte für diese Aufgabe zu Nutzen machen können.

Um nun gezielt die Felder in einem Bereich anzusprechen, verwenden Sie die `Fields`-Eigenschaft des jeweiligen Bereiches. Diese Eigenschaft gehört zu den Objekten `Document`, `Range` oder `Selection` und erwartet ein entsprechendes Objekt im Aufruf. Die `Fields`-Eigenschaft liefert eine `Fields`-Auflistung aller Felder in dem angegebenen Bereich zurück. Über die `Update`-Methode der `Fields`-Eigenschaft können Sie die Felder der jeweiligen `Fields`-Auflistung aktualisieren:

```
ActiveDocument.Fields.Update
```

Wenn Sie nun als Objekt ein StoryRange-Objekt angeben, können Sie mit der Update-Methode alle Felder in diesem Bereich aktualisieren. In Listing 22.3 werden als Beispiel die Hauptkopfzeilen in allen Abschnitten des aktuellen Dokumentes aktualisiert.

**Listing 22.3** Aktualisieren aller Felder in den Hauptkopfzeilen eines Dokumentes

```
Sub subHauptkopfzeilenfelderAktualisieren()
    Dim rngStory As Range
    For Each rngStory In ActiveDocument.StoryRanges
        If rngStory.StoryType = wdPrimaryHeaderStory Then
            rngStory.Fields.Update
            While Not (rngStory.NextStoryRange Is Nothing)
                Set rngStory = rngStory.NextStoryRange
                rngStory.Fields.Update
            Wend
        End If
    Next rngStory
End Sub
```

#### HINWEIS

Mit der Update-Methode werden im Haupttextbereich auch die Verzeichnis- und Indexfelder aktualisiert. Bei Inhaltsverzeichnissen werden aber dabei nur die Seitenzahlen aktualisiert; neu hinzugekommene Verzeichniseinträge werden nicht berücksichtigt. Um bei diesen Feldern ein vollständig neues Verzeichnis zu erstellen, müssen Sie gezielt auf das TablesOfContents-Objekt (oder ein anderes Verzeichnisfeld-Objekt) zugreifen und über die Update-Methode dieses Objektes das Verzeichnis erneuern:

```
ActiveDocument.TablesOfContents(1).Update
```

## Textfelder in Kopf- und Fußzeilen ansprechen

Textfelder (Dokumentkomponente wdTextFrameStory) gehören zu den Shape-Objekten eines Dokumentes. Die Shapes-Eigenschaft eines Dokumentes liefert dazu eine Auflistung aller Shape-Objekte zurück, die sich im Dokument, mit Ausnahme der Kopf- und Fußzeilen, befinden. Diese Shapes-Auflistung kann Zeichnungen, Formen, Bilder, OLE-Objekte, ActiveX-Steuerelemente, Textobjekte und Legenden enthalten. Textfelder stellen dabei eine besondere Gruppe innerhalb der Shape-Objekte dar, da sie Text enthalten können, der wiederum z.B. ein Feld beinhalten kann. Zur Überprüfung von Textfeldern auf Textinhalte können Sie die HasText-Eigenschaft des Textfeldes (TextFrame-Eigenschaft des Shape-Objektes) auswerten. Diese liefert True zurück, wenn das Textfeld Text beinhaltet.

Textfelder im Haupttextbereich können über die NextStoryRange-Eigenschaft durchlaufen werden. Dabei werden miteinander verknüpfte Textfelder aber nur als ein einziges Textfeld behandelt.

Um nun auch die Textfelder in Kopf- oder Fußzeilen zu erreichen, benötigen Sie ein Objekt, das die Kopf- bzw. Fußzeile repräsentiert. Hierzu können Sie die HeaderFooter-Objekte verwenden, die in jedem Abschnitt eines Dokumentes alle Kopf- und Fußzeilen zurückliefern. Da diese Objekte nur abschnittsweise die Kopf- und Fußzeilen zurückliefern, müssen Sie das Dokument auch abschnittsweise durchlaufen: z.B. in einer For Each...Next-Anweisung über alle Abschnitte:



```
For Each oSections In Activedocument.Sections
```

Anschließend müssen Sie in jedem Abschnitt getrennt voneinander die Kopf- und Fußzeilen ansprechen. Dieses erreichen Sie über die Headers- und Footers-Auflistungen der gleichnamigen Sections-Eigenschaften Headers und Footers:

```
For Each oHeaderFooter In oSections.Headers
```

und

```
For Each oHeaderFooter In oSections.Footers
```

Nachdem Sie so die Kopf- und Fußzeilen referenziert haben und ansprechen können, müssen Sie als Nächstes alle Shapes dahingehend überprüfen, ob das Shape-Objekt ein Textfeld ist und, wenn dies der Fall ist, ob das Textfeld auch Text beinhaltet.

In Listing 22.4 werden nacheinander alle Abschnitte durchlaufen und in diesen zuerst die Kopfzeilen auf Textfelder überprüft und anschließend die Fußzeilen. Für den Fall, dass die Textfelder auch Text beinhalten, werden alle evtl. vorhandenen Felder aktualisiert.

**Listing 22.4** Aktualisieren aller Textfelder in den Kopf- und Fußzeilen aller Abschnitte eines Dokumentes

```
Sub subKopfFußzeilenTextfelderAktualisieren()
    Dim oSections As Section
    Dim oHeaderFooter As HeaderFooter
    Dim shpHeaderFooter As Shape
    For Each oSections In ActiveDocument.Sections
        For Each oHeaderFooter In oSections.Headers
            For Each shpHeaderFooter In oHeaderFooter.Shapes
                If shpHeaderFooter.TextFrame.HasText Then
                    shpHeaderFooter.TextFrame.TextRange.Fields.Update
                End If
            Next shpHeaderFooter
        Next oHeaderFooter
        For Each oHeaderFooter In oSections.Footers
            For Each shpHeaderFooter In oHeaderFooter.Shapes
                If shpHeaderFooter.TextFrame.HasText Then
                    shpHeaderFooter.TextFrame.TextRange.Fields.Update
                End If
            Next shpHeaderFooter
        Next oHeaderFooter
    Next oSections
End Sub
```

## Durchlaufen aller Dokumentkomponenten

Nachdem Sie nun in den vorherigen Abschnitten erfahren haben, wie Sie Felder in den einzelnen Dokumentkomponentenbereichen aktualisieren können und wie Sie Felder in Textfeldern aktualisieren können, die sich in den Kopf- und Fußzeilen befinden, werden diese Informationen nun so in dem Listing 22.5 zusammengefasst, dass Sie alle Felder in allen Bereichen eines Dokumentes errei-

chen und aktualisieren können. Zusätzlich werden auch die besonderen Felder wie die Verzeichnissfelder berücksichtigt.

**HINWEIS** Sie können einzelne Felder über die Locked-Eigenschaft auch für die Aktualisierung sperren. Um diese Felder nicht unnötigerweise zu aktualisieren, werden nur die Felder, die nicht gesperrt sind, berücksichtigt.

**Listing 22.5** Aktualisieren aller Felder (inkl. Verzeichnis- und Indexfelder) eines Dokumentes

```
Sub subAlleFelderAktualisieren()
    Dim oDoc As Document
    Dim rngStory As Range
    Dim fldStory As Field
    Dim oSections As Section
    Dim oHeaderFooter As HeaderFooter
    Dim shpHeaderFooter As Shape
    Dim tocCont As TableOfContents
    Dim tocFig As TableOfFigures
    Dim tocAuth As TableOfAuthorities
    Set oDoc = ActiveDocument
    ' Alle Felder im Haupttextbereich, die nicht gesperrt sind, aktualisieren
    For Each rngDoc In ActiveDocument.StoryRanges
        For Each fldStory In rngDoc.Fields
            If fldStory.Locked = False Then
                fldStory.Update
            End If
        Next fldStory
        While Not (rngDoc.NextStoryRange Is Nothing)
            Set rngDoc = rngDoc.NextStoryRange
            For Each fldStory In rngDoc.Fields
                If fldStory.Locked = False Then
                    fldStory.Update
                End If
            Next fldStory
        Wend
    Next rngDoc
    ' Alle Textfelder in Shapes in allen Kopf- und Fußzeilen
    ' aller Abschnitten aktualisieren
    For Each oSections In oDoc.Sections
        ' Shapes in den Kopfzeilen überprüfen
        For Each oHeaderFooter In oSections.Headers
            For Each shpHeaderFooter In oHeaderFooter.Shapes
                If shpHeaderFooter.TextFrame.HasText Then
                    shpHeaderFooter.TextFrame.TextRange.Fields.Update
                End If
            Next shpHeaderFooter
        Next oHeaderFooter
        ' Shapes in den Fußzeilen überprüfen
        For Each oHeaderFooter In oSections.Footers
            For Each shpHeaderFooter In oHeaderFooter.Shapes
                If shpHeaderFooter.TextFrame.HasText Then
                    shpHeaderFooter.TextFrame.TextRange.Fields.Update
                End If
            Next shpHeaderFooter
        Next oHeaderFooter
    Next oSections
End Sub
```

Listing 22.5 Aktualisieren aller Felder (inkl. Verzeichnis- und Indexfelder) eines Dokumentes (Fortsetzung)

```

For Each rngStory In ActiveDocument.StoryRanges
    If rngStory.StoryType = wdTextFrameStory Then
        intTFStory = intTFStory + 1
        While Not (rngStory.NextStoryRange Is Nothing)
            Set rngStory = rngStory.NextStoryRange
            intTFStory = intTFStory + 1
        Wend
    End If
Next rngStory
' Alle Rechtsgrundlagenverzeichnisse aktualisieren
For Each tocAuth In oDoc.TablesOfAuthorities
    tocAuth.Update
Next tocAuth
' Alle Abbildungsverzeichnisse aktualisieren
For Each tocFig In oDoc.TablesOfFigures
    tocFig.Update
Next tocFig
' Alle Inhaltsverzeichnisse aktualisieren
For Each tocCont In oDoc.TablesOfContents
    tocCont.Update
Next tocCont
Set oDoc = Nothing
End Sub

```

## Erstellen und Einbinden eines Fortschrittsbalkens

Ein Fortschrittsbalken ist ein grafisches Element, das den aktuellen Fortschritt einer Verarbeitung in Bezug auf die Gesamtbearbeitung anzeigt. Beim Aktualisieren der Felder eines Dokumentes würde ein Fortschrittsbalken die zu einem bestimmten Zeitpunkt bereits aktualisierten Felder in Bezug zur Gesamtanzahl aller Felder anzeigen.

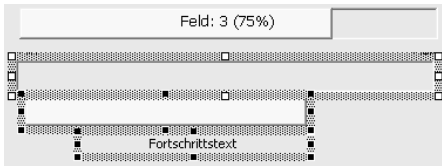
Ein Fortschrittsbalken setzt sich im Prinzip aus drei Unterelementen zusammen:

1. Dem Fortschrittsbalken-Hintergrund, der die Gesamtzahl aller Aktionen darstellt.
2. Dem Laufbalken, der die aktuelle Anzahl der bereits erfolgten Aktionen in Relation zur Gesamtzahl anzeigt.
3. Einem Anzeigetext, der die grafische Anzeige des Laufbalkens in Prozent und/oder Zahlen anzeigt.

Der Anzeigetext ist dabei nur eine weitere Ablesehilfe und kein unabdingbarer Bestandteil eines Fortschrittsbalkens.

Zusätzlich können noch Textfelder zur Anzeige der Minimum- und Maximumwerte verwendet werden.

Die Abbildung 22.4 zeigt die drei Elemente anhand des Beispiels aus dem UserForm *frmFortschrittsbalkenCtrl* in der Datei *Bsp22\_01.doc* noch einmal nebeneinander und als Ergebnis.

**Abbildg. 22.4** Bestandteile eines Fortschrittsbalkens und in der Gesamtdarstellung


**HINWEIS** Die Funktion *fkt\_setStartPos* im UserForm *frmFortschrittsBalkenCtrl*, die im Initialize-Ereignis des UserForms aufgerufen wird, sorgt für die korrekte Positionieren der drei Bestandteile und Layouteffekte.

Um einen Fortschrittsbalken zur Laufzeit anzeigen und aktualisieren zu können, muss die Gesamtzahl aller Aktionen bekannt sein.

Beim Aktualisieren der Felder können Sie die Feldanzahl in einer Dokumentkomponente über die Count-Eigenschaft der Fields-Auflistung ermitteln. Beim Aktualisieren der Felder können Sie dann aber nicht mehr mit der *For Each...Next*-Anweisung arbeiten, da diese für das verwendete Objekt nicht immer einen Laufindex anbietet, sondern müssen mit der *For...Next*-Anweisung alle Elemente bis zum Maximumwert Count durchlaufen. Dieses ist deshalb notwendig, um den Fortschritt der Aktualisierung für jedes Feld darstellen zu können. Dieser Wert muss dann an das UserForm mit dem Fortschrittsbalken weitergegeben werden, wo mit Hilfe des Dreisatzes der Laufbalken grafisch an die Gesamtzahl und somit Breite des Fortschrittsbalkens-Hintergrundes angepasst wird.

```
lblLaufbalken.Width = lblHintergrund.Width / intGesamtzahlFelder * intAktuellesFeld
```

Zum Zurücksetzen wird dann die Breite des Laufbalkens einfach auf »0« gesetzt.

Damit die Darstellung des Fortschritts einfacher mit dem Fortschritt im Dokument synchronisiert werden kann, wird die Verarbeitung nach der Anzeige des UserForms gestartet. Da das UserForm während der gesamten Aktualisierung angezeigt bleibt, stehen die Controls auf dem UserForm auch in den Prozeduren mit ihren Eigenschaften zur Verfügung.

In Listing 22.6 wird der aktuelle Aktualisierungsstand (*intFld*) der Felder im Haupttextbereich zusammen mit der Gesamtanzahl aller Felder an die Funktion *fkt\_setPos* weitergegeben, in der die Breite des Laufbalkens auf dem UserForm berechnet und angepasst wird.

**Listing 22.6** Aktualisieren der Felder im Haupttextbereich und Weitergabe der Feldnummer an das UserForm

```
For Each rngDoc In oDoc.StoryRanges
    For intFld = 1 To rngDoc.Fields.Count
        fkt_setPos intFld, rngDoc.Fields.Count
        If rngDoc.Fields(intFld).Locked = False Then
            rngDoc.Fields(intFld).Update
        End If
    Next intFld
    While Not (rngDoc.NextStoryRange Is Nothing)
        Set rngDoc = rngDoc.NextStoryRange
        For intFld = 1 To rngDoc.Fields.Count
            fkt_setPos intFld, rngDoc.Fields.Count
        Next intFld
    End While
End For
```

**Listing 22.6** Aktualisieren der Felder im Haupttextbereich und Weitergabe der Feldnummer an das UserForm (Fortsetzung)

```

    rngDoc.Fields(intFld).Update
    Next intFld
Wend
Next rngDoc

```



In der Beispieldatei *Bsp22\_01.doc* auf der CD-ROM zum Buch finden Sie im Ordner *\Beispiele\Kap22* die vollständigen Prozeduren *AlleFelderMitTextfeldernAktualisieren* und *fmt\_setPos*.

Da Felder in verschiedenen Dokumentkomponenten und in mehreren Abschnitten vorkommen können, wäre die einfache Anzeige des Fortschrittsbalkens nicht informativ genug. So fehlen Informationen über die Abschnittsnummer, ob das Feld in einem Textfeld in der Kopf- oder Fußzeile ist, ob das Feld ein Verzeichnisfeld ist und wie viele Felder am Ende insgesamt aktualisiert wurden.

Da diese Informationen mit Hilfe der StoryRanges-Auflistung ermittelt werden können und zur Ermittlung der Gesamtfeldzahl eine einfache Laufvariable ausreichend ist, steht einer ansprechenden Übersicht nichts mehr im Wege.

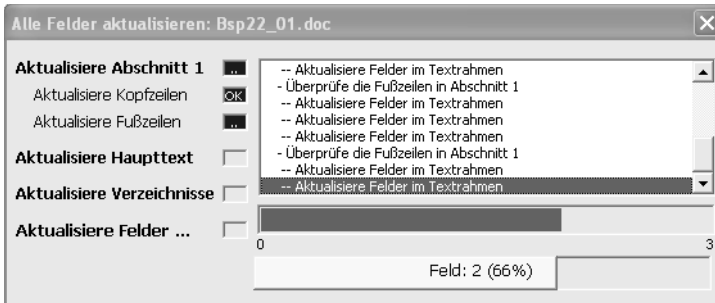
Wenn Sie in der Beispieldatei *Bsp22\_01.doc* im CD-ROM Ordner *\Beispiele\Kap22* auf die Schaltfläche *Felder aktualisieren* klicken, wird das UserForm *frmFortschrittsBalkenCtrl* angezeigt und die Aktualisierung durchläuft das aktuelle Dokument (Abbildung 22.5).

**Abbildg. 22.5** Erfolgte Aktualisierung aller Felder mit Informationen über den Aktualisierungsbereich

In dem UserForm werden die Informationen über die einzelnen durchlaufenen Dokumentkomponenten an zwei Stellen angezeigt:

- In der ListBox werden alle Einträge untereinander angezeigt und können über den vertikalen Rollbalken angezeigt werden.
- In der Textform wird der Verarbeitungszustand einzelner Dokumentkomponenten und Abschnitte angezeigt.

Während der Aktualisierung werden die Dokumentkomponenten, in denen gerade Felder aktualisiert werden, mit einem blauen Kästchen markiert. Bereiche (und somit Kästchen), in denen die Aktualisierung erfolgt ist, werden mit dem Text »OK« dargestellt, während noch nicht vollständig aktualisierte Bereiche mit zwei Punkten dargestellt werden.

**Abbildg. 22.6** Anzeige des Aktualisierungsfortschritts in den einzelnen Dokumentkomponenten


Nach erfolgter Aktualisierung aller Felder wird der Fortschrittsbalken zu einer Schaltfläche umgewandelt: Mit einem Klick auf die Schaltfläche *Schließen* in der Titelleiste wird das UserForm geschlossen.

**PROFITIPP**

Mit einer geschickten Wahl der Layouteffekte von Steuerelementen auf einem UserForm und den Steuerelement-Ereignissen `MouseDown` und `MouseUp` können Sie aus einem Textfeld-Steuerelement oder einem Bezeichnungsfeld-Steuerelement eine Schaltfläche simulieren. Dazu wird in den jeweiligen Ereignissen des Steuerelementes das Aussehen durch »umkippen« des 3D-Effektes (`SpecialEffect`-Eigenschaft) erreicht (Listing 22.7).

**Listing 22.7** Simulieren einer Schaltfläche mit einem Bezeichnungsfeld durch Änderung des Aussehens

```
Private Sub txtcounter_MouseDown(ByVal Button As Integer, ByVal Shift As Integer, _
    ByVal x As Single, ByVal y As Single)
    Me.lblBackground.SpecialEffect = fmSpecialEffectSunken
End Sub
Private Sub txtcounter_MouseUp(ByVal Button As Integer, ByVal Shift As Integer, _
    ByVal x As Single, ByVal y As Single)
    Me.lblBackground.SpecialEffect = fmSpecialEffectRaised
    Unload Me
End Sub
```

Durch Einbinden einer entsprechenden Aktion, in diesem Beispiel das Schließen des UserForms, erreichen Sie das Verhalten einer richtigen Schaltfläche.

Wenn Sie eine komplexere Schaltfläche mit Grafik und Text simulieren möchten, müssen Sie für alle Steuerelemente, die Bestandteil der Pseudo-Schaltfläche sind, die `MouseDown`- und `MouseUp`-Ereignisse mit den Effektänderungen und weiteren evtl. auszuführenden Aktionen definieren.

Eine Alternative dazu wäre das Erstellen einer richtigen Schaltfläche zur Laufzeit, was aber auch eine Größenänderung des UserForms oder Neupositionierung von Steuerelementen auf dem UserForm bedeuten würde.

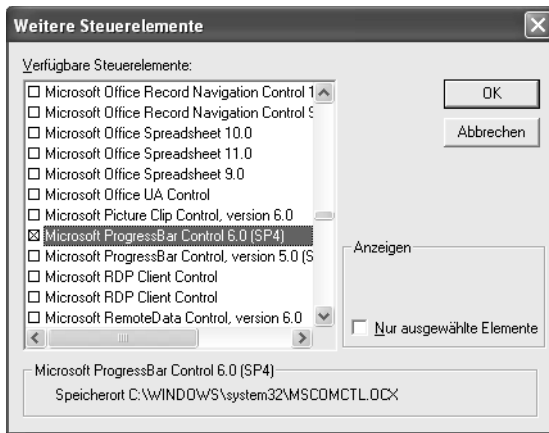
In Abbildung 22.6 sehen Sie neben dem selbst gebauten Fortschrittsbalken auch eine fertige Lösung in Form des Microsoft `ProgressBar`-Controls. Dieses wird im folgenden Abschnitt »Einbinden des Microsoft `ProgressBar`-Controls« etwas näher behandelt.

# Einbinden des Microsoft ProgressBar-Controls

Microsoft stellt mit dem Windows-Standardsteuerelement *MSCOMCTL.OCX* u.a. ein *ProgressBar*-Control zur Verfügung, das auf allen aktuellen Windows-Betriebssystemen verfügbar ist.

Sie können dieses Steuerelement in Ihre Werkzeugsammlung im Visual Basic-Editor einbinden, indem Sie mit der rechten Maustaste auf die Werkzeugsammlung klicken und im angezeigten Kontextmenü den Menüpunkt *Zusätzliche Steuerelemente* auswählen. In dem daraufhin angezeigten Dialogfeld suchen Sie den Eintrag *Microsoft ProgressBar Control 6.0 (SP4)*, markieren ihn und bestätigen die Auswahl über die *OK*-Schaltfläche.

Abbildg. 22.7 Hinzufügen des *ProgressBar*-Controls aus der Datei *MSCOMCTL.OCX*



Anschließend steht Ihnen auf der Werkzeugsammlung das neue Steuerelement *ProgressBar* zur Verfügung (Abbildung 22.8).

Abbildg. 22.8 Werkzeugsammlung mit zusätzlichem Steuerelement *ProgressBar*



Sie können dieses Steuerelement nun auf ein UserForm ziehen.

Die wichtigsten Eigenschaften dieses Steuerelementes sind die Minimum- und Maximumwert-Angaben (Eigenschaften *Min* und *Max*) und der anzuzeigende Wert (*Value*-Eigenschaft).

Analog zum Fortschrittsbalken werden dem Steuerelement während der Aktualisierung der Felder diese Informationen mitgeteilt.

Da dieses Steuerelement jedoch keine Caption- oder Text-Eigenschaft zur Anzeige eines Textes besitzt und Sie kein Bezeichnungsfeld über das Steuerelement legen können (es wird immer hinter dem Steuerelement angeordnet), wird in diesem Beispiel auf die Textanzeige des Fortschritts verzichtet.

Das Ansteuern des *ProgressBar*-Steuerelementes während der Aktualisierung erfolgt ebenfalls in der Prozedur *fkt\_setPos* (Listing 22.8). Nach dem Zurücksetzen des Laufbalkens werden erst für das *ProgressBar*-Steuerelement *pgbctrl* die Grenzen und der aktuelle Wert und anschließend für den Laufbalken der Fortschrittsanzeige die aktuelle Breite eingestellt. Im dem den Fortschrittsbalken überlagernden Textfeld wird zusätzlich zur aktuellen Feldzahl auch der prozentuale Wert in Bezug auf die Feldgesamtzahl berechnet und angezeigt.

**Listing 22.8** Festlegen der Werte für das *ProgressBar*-Steuerelement und den Fortschrittsbalken

```
Function fkt_setPos(intWert As Long, intMax As Long)
With frmFortschrittsBalkenCtrl
    .lblLeiste.Width = 0
    .lblBackground.SpecialEffect = fmSpecialEffectSunken
    .lblMax.Caption = intMax
    .lblMin.Caption = "0"
    If intWert <= intMax Then
        .pgbctrl.Min = 1
        .pgbctrl.Value = intWert * 100
        .pgbctrl.Max = intMax * 100
        .lblLeiste.Width = Int(.lblBackground.Width / intMax * intWert)
        .txtcounter.Caption = "Feld: " & intWert & " (" & Int(intWert / intMax * 100) & "%)"
        DoEvents
    Else
        .pgbctrl.Min = 0
        .pgbctrl.Value = 0
        .pgbctrl.Max = intMax * 100
    End If
End With
End Function
```

Bei Erreichen des Maximalwertes, also der Anzahl aller Felder in dem jeweiligen Bereich, wird die Anzeige der *ProgressBar* wieder auf »0« zurückgesetzt.

In Abbildung 22.6 sehen Sie als Ergebnis die synchrone Anzeige beider Darstellungen (*ProgressBar* und Fortschrittsbalken) untereinander mit identischen Werten.

Wenn Sie in der Beispieldatei *Bsp22\_01.doc* auf der Buch-CD im Ordner *\Beispiele\Kap22* auf die Schaltfläche *Felder aktualisieren* klicken, wird das UserForm *frmFortschrittsBalkenCtrl* angezeigt und die Aktualisierung wird in beiden Darstellungen (*ProgressBar* und Fortschrittsbalken) angezeigt.

## Zusammenfassung

In diesem Kapitel wurde Ihnen gezeigt, aus welchen Komponenten ein Dokument besteht, wo überall Felder eingefügt werden können und wie Sie diese ansprechen können;

- Mit Hilfe der *StoryRanges*-Auflistung und der *StoryRange*-Objekte haben Sie gesehen, wie Sie die verschiedenen Dokumentkomponenten gezielt ansprechen (Seite 781 ff. und Seite 787 ff.) und Felder in diesen Bereichen aktualisieren können.



- Anschließend wurde Ihnen die Besonderheit beim Umgang mit Feldern, die innerhalb von Textfeldern in Kopf- oder Fußzeilen eingefügt sind, gezeigt (Seite 788 ff.), bevor alle Informationen zusammengefasst wurden, um alle Felder in allen Dokumentkomponenten zu durchlaufen (Seite 789 ff.).
- Anhand eines Fortschrittsbalkens (Seite 791 ff.) und des *ProgressBar*-Steuerelementes (Seite 795 ff.) wurde Ihnen gezeigt, wie Sie den Aktualisierungsfortschritt beim Aktualisieren aller Felder in einem Dokument darstellen können.



## Kapitel 23

# Allerlei in Kopf- und Fußzeilen

### In diesem Kapitel:

Seitennummer eintragen	801
Firmenlogo einfügen	803
Wasserzeichen einfügen	806
Falz- und Lochmarke setzen	810
Absenderadresse eintragen	813
Dateiname und Dokumentdatum setzen	815
Zusammenfassung	819

Dieses Kapitel enthält Lösungsvorschläge zu regelmäßig auftauchenden Fragen im Zusammenhang mit Kopf- und Fußzeilen. Die Lösungen bauen auf den nachstehenden Objekten, Eigenschaften und Methoden auf.

---

**Im Blickpunkt:****Word-Objekte**

Application.CentimetersToPoints, Application.PointsToCentimeters  
Documents.Add  
Fields.Add  
HeaderFooter  
Shape.Delete, Shape.LockAnchor, Shape.RelativeHorizontalPosition, Shape.Rotation, Shape.ZOrder  
Shapes.AddLine, Shapes.AddPicture, Shapes.AddTextBox, Shapes.AddTextEffect  
Table.Add, Table.Borders, Table.Shading, Table.Style, Table.Condition  
Range.Collapse, Range.InsertAfter, Range.InsertBefore, Range.NextStoryRange, Range.ParagraphFormat, Range.Text

---

Bei den Kopf- und Fußzeilen handelt es sich um Bereiche eines Dokuments, deren Informationen auf jeder Seite des Dokuments ausgegeben werden.

Es werden Lösungsbeispiele zum Einfügen von Seitennummern, Firmenlogos, Wasserzeichen sowie Falz- und Lochmarken in der Kopfzeile aufgeführt. Weitere Beispiele behandeln das Eintragen von Absenderadresse, Dateiname und ähnlichen Informationen in die Fußzeile.

**HINWEIS**

Die Beispieldateien für dieses Kapitel sind gleich aufgebaut. Jede Datei enthält mindestens zwei VBA-Module. Davon trägt eines jeweils die Bezeichnung `vbmDemo` und beinhaltet, wie in Listing 23.1 dargestellt, die Programmsequenz, um ein neues Dokument zu erzeugen und dieses – soweit nötig – vorzubereiten.

Ein zweites VBA-Modul beinhaltet jeweils die Programmzeilen, die zum eigentlichen Beispiel gehören. Nur diese Prozedur wird im Beispiel erläutert.

Eventuelle weitere VBA-Module beinhalten in erster Linie Hilfsprozeduren, die für das Makro benötigt, aber nicht speziell erläutert werden.

**Listing 23.1**    Einstiegsprozedur für das Programmbeispiel in der Datei *Bsp23\_01.dot*

```
Sub Demo_Bsp23_01()  
    Dim docDemo As Word.Document  
  
    'Neues Dokument erstellen  
    Set docDemo = Documents.Add(Template:=ThisDocument.FullName)  
  
    'Kopfzeile mit Seitennummer einfügen
```

Listing 23.1 Einstiegsprozedur für das Programmbeispiel in der Datei *Bsp23\_01.dot* (Fortsetzung)

```

procKopfzeileSeitennummerEinfügen _
    doc:=docDemo, _
    intKopfzeile:=wdHeaderFooterPrimary, _
    intVariante:=eSeitenNrKeineXvonY

    Set docDemo = Nothing
End Sub

```

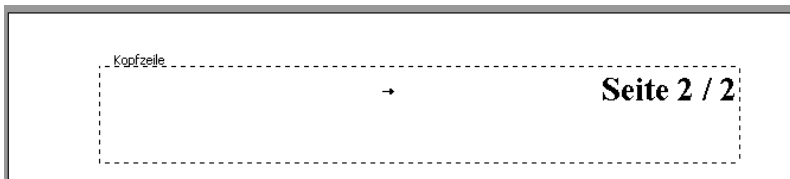
**HINWEIS** Für die nachstehenden Beispiele wurden die zugehörigen Formatvorlagen der Kopf- und Fußzeile auf die gewünschte Darstellung hin formatiert. Aus diesem Grunde konnte auf Formatierungsanweisungen innerhalb des Codes verzichtet werden.

## Seitennummer eintragen

In den meisten Dokumenten werden auch die Seitenzahlen ausgegeben, und zwar sinnvollerweise in der Kopf- oder Fußzeile. Im folgenden Beispiel wird die Seitennummer oben rechts im Dokument eingetragen.

Damit die Prozedur möglichst viele Anforderungen abdeckt, soll das Format der Seitennummer frei definiert werden können. Dies gilt für das Präfix und das Trennzeichen sowie die eingefügten Seitenzahlen. Zudem sollen sie mit (vgl. Abbildung 23.1) oder ohne Angabe der Gesamtseitenzahl dargestellt werden können.

Abbildg. 23.1 Eingefügte Seitenzahl in der Kopfzeile des Dokuments



In Listing 23.2 sind die Anforderungen an eine allgemein gültige Prozedur umgesetzt. Diese verfügt über fünf Argumente, die kurz vorgestellt werden.

Tabelle 23.1 Eingabeargumente der Prozedur *procKopfzeileSeitennummerEinfügen*

Argument	Bedeutung
doc	Objektvariable auf das zu bearbeitende Dokument.
intKopfzeile	Einen Wert aus der <code>WdHeaderFooterIndex</code> -Enumeration. Legt fest, welche Kopfzeile bearbeitet werden soll.
intVariante	Ein Wert aus der <code>EVarianteSeitennummer</code> -Enumeration. Legt das Darstellungsformat der Seitennummer fest. <code>eSeitenNrKeine</code> – keine Seitennummer wird eingefügt <code>eSeitenNrX</code> – nur die Seitennummer wird eingefügt <code>eSeitenNrXvonY</code> – die Seitennummer und die Gesamtanzahlseiten wird eingefügt

**Tabelle 23.1**    Eingabeargumente der Prozedur *procKopfzeileSeitennummerEinfügen* (Fortsetzung)

Argument	Bedeutung
strPräfix	Optional. Präfix vor der eigentlichen Seitennummer.
strTrennzeichen	Optional. Trennzeichen zwischen den beiden Feldern.

**Listing 23.2**    Prozedur zum Einfügen der Seitennummer mit frei definierbarem Format

```

Public Enum EVarianteSeitennummer
    eSeitenNrKeine = 0
    eSeitenNrX = 1
    eSeitenNrXvonY = 2
End Enum

Public Sub procKopfzeileSeitennummerEinfügen( _
    ByVal doc As Word.Document, _
    ByVal intKopfzeile As WdHeaderFooterIndex, _
    ByVal intVariante As EVarianteSeitennummer, _
    Optional strPräfix As String = "", _
    Optional strTrennzeichen As String = " / ")

    Const FORMAT As String = "\# ""#,##0"" \* Arabic"

    Dim rng As Word.Range

    'Kopfzeilen-Bereich als Range festlegen.
    Set rng = doc.Sections(1).Headers(intKopfzeile).Range

    'Inhalt der Kopfzeile sicherheitshalber löschen
    'und Formatvorlage »Kopfzeile« zuweisen
    With rng
        .Delete
        .Style = wdStyleHeader
    End With

    'Seitennummer in der Kopfzeile setzen.
    'Feld «AnzahlSeiten»
    If intVariante = eSeitenNrXvonY Then
        .Fields.Add Range:=rng, Type:=wdFieldNumPages, _
            Text:=FORMAT, PreserveFormatting:=True
        .Collapse (wdCollapseStart)
        .InsertBefore (strTrennzeichen)
    End If
    'Feld «Seiten»
    If Not intVariante = eSeitenNrKeine Then
        .Collapse (wdCollapseStart)
        .Fields.Add Range:=rng, Type:=wdFieldPage, _
            Text:=FORMAT, PreserveFormatting:=True
        .Collapse (wdCollapseStart)
        .InsertBefore (strPräfix)
    End If
    .Collapse (wdCollapseStart)
    .InsertBefore (vbTab)
End With
End Sub

```

Die Programmzeilen innerhalb der Prozedur müssen nicht einzeln erläutert werden. Speziell hingegen ist der eigentliche Aufbau der Kopfzeile, die bewusst von rechts nach links aufgebaut wird.

Der Grund dazu ist das Range-Objekt, das in diesem Fall ganz anders verwaltet werden kann, als wenn der Aufbau von links nach rechts erfolgen würde.

Würde der Aufbau, wie es eigentlich logisch wäre, von links gestartet, so müsste nach dem Einfügen einer Feldfunktion das Range-Objekt neu definiert werden, da das Feld außerhalb des Bereichs hinzugefügt und das Kollabieren zum Endpunkt dieses Feld noch nicht beinhalten würde (vgl. Abbildung 23.2).

Abbildg. 23.2 Die Einfügemarke steht vor der Feldfunktion.



Listing 23.3 Probleme entstehen, wenn das *Range*-Objekt von links her aufgebaut wird.

```
Sub Test()
    Dim doc As Word.Document
    Dim rng As Word.Range
    Set doc = Documents.Add
    Set rng = doc.Paragraphs(1).Range
    rng.InsertAfter vbTab & "Seite "
    rng.Collapse wdCollapseEnd
    rng.Fields.Add rng, wdFieldNumPages
    rng.Collapse wdCollapseEnd
    rng.Select
End Sub
```

Wird die Kopfzeile jedoch von rechts nach links aufgebaut, so kann dieses Problem elegant umgangen werden. Das Range-Objekt muss nie neu definiert werden, und die gewünschte Darstellung der Seitennummerierung kann mit wenigen Programmzeilen umgesetzt werden.



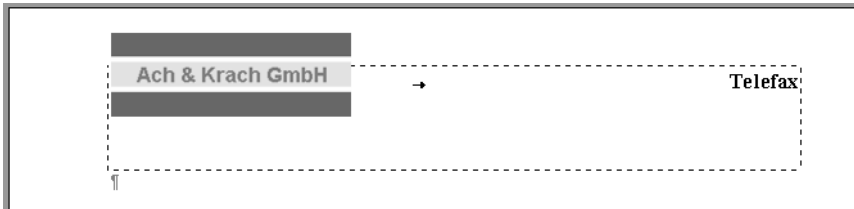
Die Prozeduren in obigem Abschnitt finden Sie in der Beispieldatei *Bsp23\_01.dot*. Diese befindet sich auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap23*.

## Firmenlogo einfügen

Wurde früher das offizielle Firmenpapier mit dem Logo und der Geschäftsadresse von der Druckerei hergestellt, so kommt heute oft nur noch weißes Papier zum Einsatz. Das Logo und die Adresse werden direkt während des Ausdrucks erzeugt. Ein weiterer Grund für diese Vorgehensweise kann das Erstellen von *.pdf*-Dateien für die Veröffentlichung im Internet oder für den Versand via E-Mail sein. In beiden Fällen möchte der Ersteller, dass die Vorgaben des Corporate Design auch beim Empfänger eingehalten werden, und dazu gehört auch das Logo.

In diesem Beispiel wird das Logo bewusst nur auf der ersten Seite des Dokuments eingetragen. Damit die Prozedur flexibel eingesetzt werden kann, sollen die Grafikdatei und deren Position frei definiert werden können. Zusätzlich kann ein optionaler Wert als Argument übergeben werden, welcher als Dokumenttyp in der rechten oberen Ecke des Dokuments ausgegeben wird.

**Abbildg. 23.3**    Eingefügtes Firmenlogo in der Kopfzeile des Dokuments



In Listing 23.4 sind die entsprechenden Anforderungen an eine solche allgemeingültige Programmsequenz umgesetzt. Die Prozedur verfügt über fünf Argumente, die folgende Bedeutung haben.

**Tabelle 23.2**    Eingabeargumente der Prozedur *procKopfzeileLogoEinfügen*

Argument	Bedeutung
doc	Objektvariable auf das zu bearbeitende Dokument
strLogoDatei	Dateiname der Logodatei. Dieser sollte jeweils eindeutig, am besten mit einer absoluten Pfadangabe, übergeben werden.
sngLogoPositionOben	Abstand zwischen dem oberen Rand des Logos und der Seite (Papierrand). Die Angabe erfolgt in Zentimeter.
sngLogoPositionLinks	Abstand zwischen dem linken Rand des Logos und der Seite (Papierrand). Die Angabe erfolgt in Zentimeter.
strDokumentTyp	Optional. Bezeichnung des Dokumenttyps, die in der Kopfzeile zusätzlich aufgeführt wird.

**Listing 23.4**    Prozedur zum Einfügen des Logos an einer definierbaren Stelle auf der ersten Seite des Dokuments

```
Public Sub procKopfzeileLogoEinfügen( _
    ByVal doc As Word.Document, _
    ByVal strLogoDatei As String, _
    ByVal sngLogoPositionOben As Single, _
    ByVal sngLogoPositionLinks As Single, _
    Optional strDokumentTyp As String)

    Dim hdr As Word.HeaderFooter
    Dim shp As Word.Shape

    'Kopfzeilen-Bereich als Range festlegen.
    Set hdr = doc.Sections(1).Headers(wdHeaderFooterFirstPage)

    'Inhalt der Kopfzeile sicherheitshalber löschen
    'und Formatvorlage "Kopfzeile" zuweisen
    With hdr
```



**Listing 23.4** Prozedur zum Einfügen des Logos an einer definierbaren Stelle auf der ersten Seite des Dokuments (*Fortsetzung*)

```

        .Range.Delete
        .Range.Style = wdStyleHeader

'Logo einfügen
If fktExistiertDatei(strLogoDatei) Then
    Set shp = .Shapes.AddPicture( _
        FileName:=strLogoDatei, Anchor:=hdr.Range, _
        LinkToFile:=False, SaveWithDocument:=True)

    With shp
        .Name = "Logo_" & Format$(Now, "yymmddhhnnss")
        .RelativeHorizontalPosition = wdRelativeHorizontalPositionPage
        .RelativeVerticalPosition = wdRelativeVerticalPositionPage
        .LockAnchor = True
        .Left = CentimetersToPoints(sngLogoPositionLinks)
        .Top = CentimetersToPoints(sngLogoPositionOben)
    End With
End If

'Dokumenttyp einfügen
If Not Len(Trim$(strDokumentTyp)) = 0 Then
    .Range.InsertBefore (strDokumentTyp)
    .Range.InsertBefore (vbTab)
End If
End With
End Sub

```

Die eigentliche Funktionsweise der Prozedur muss nicht explizit erläutert werden. Einzelne Programmzeilen wollen wir aber dennoch etwas genauer betrachten.

Das Firmenlogo wird als Shape-Objekt in das Dokument eingefügt und darin gespeichert. So steht das Logo auch dann zur Verfügung, wenn das Dokument auf einem anderen System bearbeitet wird.

Das Logo wird innerhalb der Kopfzeile hinzugefügt (`hdr.Shapes.AddPicture`) und zusätzlich mit derselben verbunden (`Anchor:=hdr.Range`).

```

Set shp = hdr.Shapes.AddPicture( _
    FileName:=strLogoDatei, Anchor:=hdr.Range, _
    LinkToFile:=False, SaveWithDocument:=True)

```

Damit das Logo zu einem späteren Zeitpunkt gezielt bearbeitet werden kann, wird es mit einem Namen versehen. Der eigentliche Namen wird aus einem Präfix (Logo\_) und einer laufenden Nummer zusammengesetzt, welche vom aktuellen Datum und der momentanen Uhrzeit abgeleitet wird. So ist sichergestellt, dass das Shape-Objekt innerhalb der Shapes-Auflistung gefunden wird und der Name garantiert eindeutig ist:

```

shp.Name = "Logo_" & Format$(Now, "yymmddhhnnss")

```

Bevor ein Shape-Objekt im Dokument positioniert werden kann, muss unbedingt festgelegt werden, von welcher Position her gemessen wird. In diesem Fall wird von der Seite, also von der Kante des Papiers, gemessen:

```
shp.RelativeHorizontalPosition = wdRelativeHorizontalPositionPage
shp.RelativeVerticalPosition = wdRelativeVerticalPositionPage
```

Im Weiteren sollte jedes Shape-Objekt, nachdem es mit einem bestimmten Absatz verbunden wurde, zusätzlich verankert werden, damit die definierte Verbindung nicht unabsichtlich verändert wird:

```
shp.LockAnchor = True
```



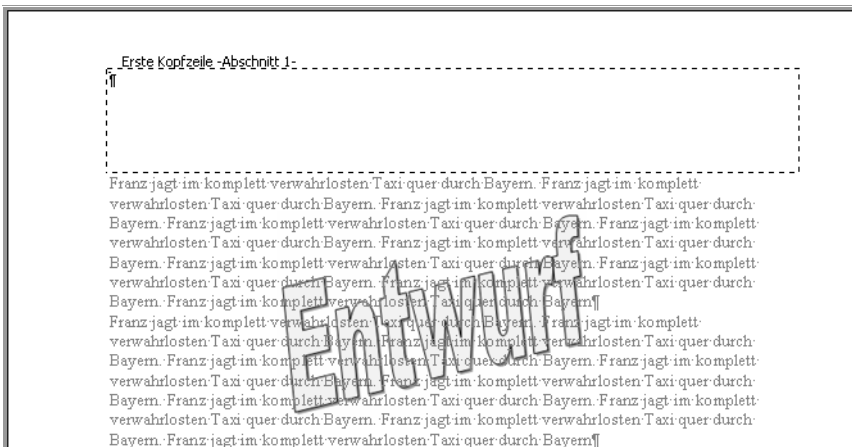
Die Prozeduren in obigem Abschnitt finden Sie in der Beispieldatei *Bsp23\_02.dot*. Diese befindet sich auf der CD-ROM zum Buch im Ordner `\Beispiele\Kap23`.

## Wasserzeichen einfügen

Soll ein Dokument klassifiziert werden, eignen sich für diesen Fall so genannte Wasserzeichen ganz besonders. Das Wasserzeichen wird normalerweise in der Kopfzeile eingefügt und in der Zeichnungsebene hinter den Text gelegt. Das Einfügen innerhalb der Kopfzeilen stellt in erster Linie sicher, dass das Wasserzeichen auf allen Seiten des Dokuments sichtbar ist. Die eigentliche Position der Grafik kann auch außerhalb des Kopfzeilenbereichs liegen – beispielsweise auf der Seite zentriert, wie in Abbildung 23.4 dargestellt.

Damit die Prozedur für unterschiedliche Wasserzeichen genutzt werden kann, soll der Dateiname der Grafik als Argument übergeben werden können. Das aktuelle Beispiel zentriert die Grafik auf dem Dokument. Es wäre jedoch ein Leichtes, die Prozedur so zu erweitern, dass die effektive Position als zusätzliches Argument übergeben werden könnte.

Abbildg. 23.4 Das eingefügte Wasserzeichen ist in der Kopfzeile hinterlegt



In Listing 23.5 sind die entsprechenden Anforderungen an eine allgemein gültige Prozedur umgesetzt. Diese verfügt über zwei Argumente, die anschließend kurz vorgestellt werden.

Das eigentliche Einfügen der Grafik im Dokument erfolgt in einer Hilfsprozedur mit der Bezeichnung *procKopfzeileWasserzeichenEinfügen\_2*. Die zugehörigen Programmzeilen sind in Listing 23.6 dargestellt. Die Prozedur ist als Private deklariert; so ist sichergestellt, dass kein direkter Aufruf derselben erfolgen kann.

**Tabelle 23.3** Eingabeargumente der Prozedur *procKopfzeileWasserzeichenEinfügen*

Argument	Bedeutung
Doc	Objektvariable auf das zu bearbeitende Dokument
strWasserzeichenDatei	Dateiname der Grafik, die als Wasserzeichen eingefügt wird. Dieser sollte jeweils eindeutig, am besten mit einer absoluten Pfadangabe, übergeben werden.

**HINWEIS** Damit die Funktionsweise der Prozedur in Listing 23.5 besser verständlich ist, wurde die Beispieldatei *Bsp23\_03.dot* bereits entsprechend formatiert.

In der Dokumentvorlage wurde ein Abschnittswechsel eingefügt. In allen Abschnitten wurden die beiden Layouteigenschaften für die Kopf- und Fußzeilen (*Gerade/ungerade anders* und *Erste Seite anders*) aktiviert. Bei der Kopfzeile der ersten Seite des zweiten Abschnitts wurde die Verknüpfung zur vorherigen gelöst. Dies bedeutet, dass das Dokument über vier unterschiedliche Kopfzeilen verfügt. Die Hilfsprozedur wird dementsprechend viermal abgearbeitet.

**Listing 23.5** Prozedur zum Einfügen eines Wasserzeichens mit frei wählbarer Grafikdatei

```
Public Sub procKopfzeileWasserzeichenEinfügen( _
    ByVal doc As Word.Document, _
    ByVal strWasserzeichenDatei As String)

    Dim rng As Word.Range
    Dim intHdr As WdHeaderFooterIndex

    If fktExistiertDatei(strWasserzeichenDatei) Then
        'Alle Kopfzeilen bearbeiten
        For intHdr = wdHeaderFooterPrimary To wdHeaderFooterEvenPages
            Set rng = doc.Sections(1).Headers(intHdr).Range
            procKopfzeileWasserzeichenEinfügen_2 _
                rng:=rng, _
                strGrafik:=strWasserzeichenDatei

            'Nächste Kopfzeile vom gleichen Typ
            While Not (rng.NextStoryRange Is Nothing)
                Set rng = rng.NextStoryRange
                procKopfzeileWasserzeichenEinfügen_2 _
                    rng:=rng, _
                    strGrafik:=strWasserzeichenDatei
            Wend
        Next intHdr
    End If
End Sub
```

Die Prozedur stellt sicher, dass alle drei möglichen Kopfzeilen bearbeitet werden. Dies erfolgt unabhängig davon, wie die Layouteigenschaften des Dokuments gesetzt wurden:

```
For intHdr = wdHeaderFooterPrimary To wdHeaderFooterEvenPages
```

Anhand der `NextStoryRange`-Eigenschaft kann festgestellt werden, ob für einen einzelnen Kopfzeilenbereich weitere Bereiche in einem weiteren Abschnitt vorhanden sind. Alle diese zusätzlichen Bereiche werden innerhalb einer Schleife bearbeitet. Das `StoryRange`-Objekt sowie die `NextStoryRange`-Eigenschaft wurden bereits in Kapitel 6 vorgestellt.

```
While Not (rng.NextStoryRange Is Nothing)
Set rng = rng.NextStoryRange
```

Die Hilfsprozedur aus Listing 23.6 fügt das eigentliche Wasserzeichen in der entsprechenden Kopfzeile ein und setzt die Grafik an die gewünschte Position.

**Listing 23.6**    Die Hilfsprozedur *procKopfzeileWasserzeichenEinfügen\_2* fügt die Grafik ein und formatiert sie.

```
Private Sub procKopfzeileWasserzeichenEinfügen_2( _
ByVal rng As Word.Range, _
ByVal strGrafik As String)

Dim shp As Word.Shape

'Wasserzeichen einfügen
Set shp = ActiveDocument.Shapes.AddPicture( _
    FileName:=strGrafik, Anchor:=rng, _
    LinkToFile:=False, SaveWithDocument:=True)

With shp
    .Name = "Wasserzeichen_" & CStr(Rnd())
    .RelativeHorizontalPosition = wdRelativeHorizontalPositionPage
    .RelativeVerticalPosition = wdRelativeVerticalPositionPage
    .LockAnchor = True
    .Left = (ActiveDocument.Sections(1).PageSetup.PageWidth - .Width) / 2
    .Top = (ActiveDocument.Sections(1).PageSetup.PageHeight - .Height) / 2
    .ZOrder msoSendToBack
End With
End Sub
```

Die eigentliche Funktionsweise der Hilfsprozedur muss nicht explizit erläutert werden. Dennoch wollen wir einzelne Programmzeilen etwas genauer betrachten.

Das Wasserzeichen wird ebenso behandelt wie das Firmenlogo aus dem Abschnitt »Firmenlogo einfügen« in diesem Kapitel. Die Datei wird in das Dokument eingefügt und im Kopfzeilenbereich verbunden.

Alle Grafiken werden mit einem Namen versehen. Dieser besteht aus einem statischen Präfix und einem dynamischen Suffix. Das Suffix wird automatisch anhand der `Rnd`-Funktion erzeugt:

```
shp.Name = "Wasserzeichen_" & CStr(Rnd())
```

Die eigentliche Position der Grafik wird dynamisch berechnet. So ist sichergestellt, dass die Grafik unabhängig von ihrer Größe stets zentriert in das Dokument eingefügt wird:

```
shp.Left = (ActiveDocument.Sections(1).PageSetup.PageWidth - .Width) / 2
shp.Top = (ActiveDocument.Sections(1).PageSetup.PageHeight - .Height) / 2
```

### Wasserzeichen als WordArt-Objekt

Im Beispiel aus Listing 23.5 wurde eine externe Grafikdatei als Wasserzeichen in das Dokument eingefügt. Neben einer solchen Grafik könnte auch ein Wordart-Objekt als so genanntes Wasserzeichen eingesetzt werden. Ein vereinfachtes Beispiel ist in Listing 23.7 dargestellt.

Für den geneigten Leser sollte es ein Leichtes sein, die beiden Prozeduren so anzupassen, dass anstelle einer externen Grafikdatei ein Wordart-Objekt als Wasserzeichen eingetragen wird.

**Listing 23.7** Ein Wasserzeichen ohne externe Grafikdatei mittels eines WordArt-Objekts erzeugen

```
Sub Demo_WordartEinfügen()
    Dim doc As Word.Document
    Dim hdr As Word.HeaderFooter
    Dim shp As Word.Shape

    Set doc = Documents.Add
    Set hdr = doc.Sections(1).Headers(wdHeaderFooterPrimary)
    Set shp = hdr.Shapes.AddTextEffect( _
        PresetTextEffect:=msoTextEffect1, _
        Text:="Entwurf", _
        FontName:="Arial", FontSize:=40, FontBold:=True, FontItalic:=True, _
        Left:=100, Top:=100, _
        Anchor:=hdr.Range)

    With shp
        .Name = "Wasserzeichen"
        .LockAnchor = True
        .Rotation = -30
        .ZOrder msoSendToBack
    End With
End Sub
```

### Wasserzeichen entfernen

Wird die Klassifikation des Dokuments geändert, muss das Wasserzeichen entsprechend angepasst werden. Dies bedeutet, dass zuerst das alte Wasserzeichen entfernt und anschließend bei Bedarf ein neues eingefügt wird.

In Listing 23.8 ist eine Prozedur aufgeführt, die alle Shape-Objekte, deren Namen mit dem Präfix *Wasserzeichen\_* beginnen, entfernt.

**Listing 23.8** Prozedur zum Entfernen der eingefügten Wasserzeichen

```
Public Sub vbmKopfzeileWasserzeichenEntfernen()
    Dim hdr As Word.HeaderFooter
    Dim shp As Word.Shape

    Set hdr = ActiveDocument.Sections(1).Headers(wdHeaderFooterPrimary)
```

**Listing 23.8** Prozedur zum Entfernen der eingefügten Wasserzeichen (*Fortsetzung*)

```

For Each shp In hdr.Shapes
    If Not InStr(1, shp.Name, "Wasserzeichen_", vbTextCompare) = 0 Then
        shp.Delete
    End If
Next shp
End Sub

```

Die einzelnen Shape-Objekte, die sich innerhalb der Kopfzeile befinden, werden mit einer Schleife bearbeitet. Enthält der Name des Objekts die Zeichenkette *Wasserzeichen\_*, wird dieses entfernt.

**HINWEIS**

Word verwaltet die Shape-Objekte, die sich innerhalb der Kopfzeilen befinden, auf eine ganz spezielle Art. So reicht eine einfache Schleife innerhalb irgendeines Kopfzeilenbereiches aus, um alle Shape-Objekte in *allen* unterschiedlichen Kopfzeilenbereichen unabhängig vom entsprechenden Abschnitt zu bearbeiten.

Aus diesem Grunde können die Wasserzeichen, wie in Listing 23.8 dargestellt, bedeutend einfacher entfernt werden, als diese vorher eingefügt wurden.



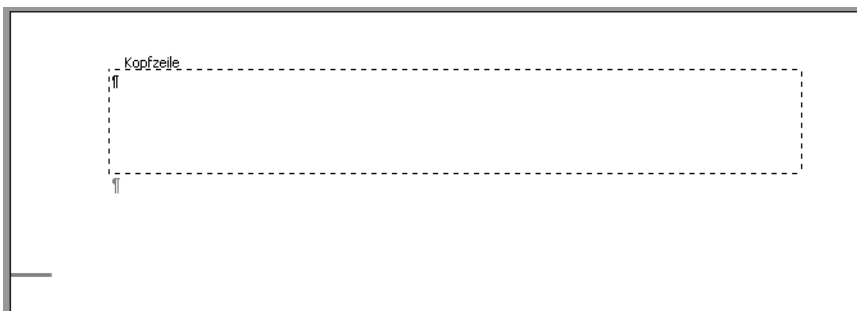
Die Prozeduren in obigem Abschnitt finden Sie in der Beispieldatei *Bsp23\_03.dot*. Diese befindet sich auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap23*.

## Falz- und Lochmarke setzen

Spezielle Hilfslinien im Dokument können den Anwender bei seiner täglichen Arbeit unterstützen. Zu solchen Hilfslinien können die Loch- bzw. Falzmarken am linken Rand des Dokuments gezählt werden.

Die Prozedur soll für beide Arten von Hilfslinien eingesetzt werden können. Die eigentliche Position der Lochmarke wird berechnet. Diese Linie sollte sich immer in der Mitte des Blattes befinden, unabhängig vom Format und der Ausrichtung.

Die Position der beiden Falzmarken wird fix auf ein Maß von 10,5 cm bzw. 21,0 cm festgelegt. Dies entspricht den Maßen aus der DIN 676. So passt das Dokument in einen Briefumschlag vom Format C5/6.

**Abbildg. 23.5** Eingefügte Falzmarke in der Kopfzeile des Dokuments


In Listing 23.9 wurde eine entsprechende Prozedur erstellt, die den gestellten Anforderungen gerecht wird. Sie verfügt über zwei Eingabeargumente, die kurz zusammengefasst werden.

**Tabelle 23.4** Eingabeargumente der Prozedur *procKopfzeileFalzLochmarkeEinfügen*

Argument	Bedeutung
doc	Objektvariable auf das zu bearbeitende Dokument
inhHilfslinie	Ein Wert aus der <b>EVarianteHilfslinien</b> -Enumeration. Legt die gewünschte Hilfslinie, die eingefügt wird, fest. <b>eHilfslinieLochmarke</b> – eine Lochmarke wird eingefügt <b>eHilfslinieFalzmarke</b> – die beiden Falzmarken werden eingefügt <b>eHilfslinieFalzUndLochmarke</b> – Loch- und Falzmarken werden eingefügt

Das eigentliche Einfügen der Hilfslinie wurde in eine separate Prozedur ausgelagert, diese ist in Listing 23.10 aufgeführt.

**Listing 23.9** Prozedur zum Einfügen der Loch- bzw. Falzmarke in das entsprechende Dokument

```

Const sngFALZMARKE_OBEN As Single = 10.5
Const sngFALZMARKE_UNTEN As Single = 21#

Public Enum EVarianteHilfslinien
    eHilfslinieLochmarke = 1
    eHilfslinieFalzmarke = 2
    eHilfslinieFalzUndLochmarke = 3
End Enum

Public Sub procKopfzeileFalzLochmarkeEinfügen( _
    ByVal doc As Word.Document, _
    ByVal inhHilfslinie As EVarianteHilfslinien)

    'Position der Marke bestimmen
    Select Case inhHilfslinie
        Case eHilfslinieFalzmarke
            subHilfslinieEinfügen docA:=doc, sngPositionTop:=sngFALZMARKE_OBEN
            subHilfslinieEinfügen docA:=doc, sngPositionTop:=sngFALZMARKE_UNTEN
        Case eHilfslinieLochmarke
            subHilfslinieEinfügen docA:=doc, sngPositionTop:= _
                PointsToCentimeters(doc.Sections(1).PageSetup.PageHeight / 2)
        Case eHilfslinieFalzUndLochmarke
            subHilfslinieEinfügen docA:=doc, sngPositionTop:=sngFALZMARKE_OBEN
            subHilfslinieEinfügen docA:=doc, sngPositionTop:=sngFALZMARKE_UNTEN
            subHilfslinieEinfügen docA:=doc, sngPositionTop:= _
                PointsToCentimeters(doc.Sections(1).PageSetup.PageHeight / 2)
    End Select
End Sub

```

Die Funktionsweise der Prozedur bedarf keiner Erklärung. In einem ersten Schritt werden die übertragenden Argumente ausgewertet und es erfolgen die Aufrufe der Hilfsprozedur, welche die eigentliche Marke in die Kopfzeile des Dokuments einfügt.

**Listing 23.10** Hilfsprozedur zum Zeichnen der Hilfslinie im Dokument

```
Private Sub subHilfslinieEinfügen( _
    ByVal docA As Word.Document, _
    ByVal sngPositionTop As Single)

    Dim hdr As Word.HeaderFooter
    Dim shp As Word.Shape
    Dim intHdr As WdHeaderFooterIndex

    'Kopfzeile bestimmen
    If docA.Sections(1).PageSetup.DifferentFirstPageHeaderFooter Then
        intHdr = wdHeaderFooterFirstPage
    Else
        intHdr = wdHeaderFooterPrimary
    End If
    Set hdr = docA.Sections(1).Headers(intHdr)

    'Loch- bzw. Falzmarke erstellen und formatieren
    Set shp = hdr.Shapes.AddLine(1, 1, 1, 1, hdr.Range)
    With shp
        With .Line
            .Weight = 0.25
            .Style = msoLineSingle
            .DashStyle = msoLineSolid
            .ForeColor.RGB = RGB(128, 128, 128)
            .BeginArrowheadStyle = msoArrowheadNone
            .EndArrowheadStyle = msoArrowheadNone
        End With
        .Height = CentimetersToPoints(0)
        .Width = CentimetersToPoints(0.8)
        .RelativeHorizontalPosition = wdRelativeHorizontalPositionPage
        .RelativeVerticalPosition = wdRelativeVerticalPositionPage
        .Left = CentimetersToPoints(0)
        .Top = CentimetersToPoints(sngPositionTop)
        .LockAnchor = True
    End With
End Sub
```

Die Funktionsweise der Hilfsprozedur bedarf ebenfalls keiner detaillierten Erklärung. Nachdem in einem ersten Schritt die entsprechende Kopfzeile festgelegt wurde, wird ein Zeichenobjekt (Linie) in das Dokument eingefügt.

Als Startposition werden irgendwelche Werte eingegeben, die zu einem späteren Zeitpunkt korrigiert werden, sobald die Eigenschaften `RelativeHorizontalPosition` und `RelativeVerticalPosition` gesetzt wurden:

```
Set shp = hdr.Shapes.AddLine(1, 1, 1, 1, hdr.Range)
```

Damit die Hilfslinie im Dokument diskret dargestellt wird, wurde sie statt schwarz in einem Grauton formatiert:

```
shp.ForeColor.RGB = RGB(128, 128, 128)
```



**HINWEIS** Werden Zeichenelemente mittels einer VBA-Prozedur auf ein Dokument übertragen, sollten unbedingt *alle* Eigenschaften des betreffenden Objekts bearbeitet werden.

Der Grund dazu ist nahe liegend. Für alle Eigenschaften, die nicht bearbeitet werden, wird der so genannte Standardwert verwendet. Welchen Wert eine einzelne Eigenschaft zum Zeitpunkt der Bearbeitung aufweist, kann auf jeder Arbeitsstation unterschiedlich sein.

Der Anwender hat sogar die Möglichkeit, ein Objekt nach seinem Gutdünken zu formatieren und diese Formate als Standardeigenschaften einzutragen. Dazu muss im Kontextmenü (rechte Maustaste betätigen) des Zeichenobjekts der Befehl *Als Standard für AutoForm festlegen* gewählt werden.



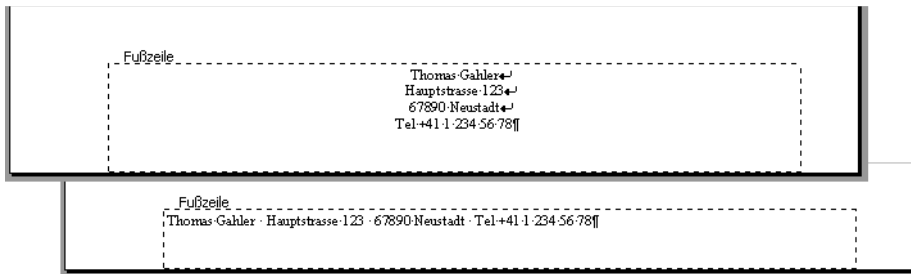
Die Prozedur in obigem Abschnitt finden Sie in der Beispieldatei *Bsp23\_04.dot*. Diese befindet sich auf der CD-ROM zum Buch im Ordner `\Beispiele\Kap23`.

## Absenderadresse eintragen

Die Fußzeile ist ein geeigneter Bereich, um allgemeine Absenderangaben im Dokument zu platzieren. In diesem Beispiel kann eine Adresse, bestehend aus einer unbegrenzten Anzahl von Einzelelementen, eingefügt werden.

Damit die Prozedur möglichst viele Anforderungen abdeckt, soll das Trennzeichen zwischen den Elementen und der Adresse sowie die Ausrichtung anhand von Argumenten definiert werden können (vgl. Abbildung 23.6).

**Abbildg. 23.6** Eingefügte Absenderadresse in der Fußzeile des Dokuments



In Listing 23.11 sind die Anforderungen an eine allgemeingültige Prozedur umgesetzt. Diese verfügt über vier Argumente, die kurz vorgestellt werden.

**Tabelle 23.5** Eingabeargumente der Prozedur *procFusszeileAdresseEinfügen*

Argument	Bedeutung
doc	Objektvariable auf das zu bearbeitende Dokument.
strAdresse()	Ein Datenfeld, das die einzelnen Datenwerte für die Absenderadresse enthält.

**Tabelle 23.5**    Eingabeargumente der Prozedur *procFusszeileAdresseEinfügen* (Fortsetzung)

Argument	Bedeutung
intTrennzeichen	Ein Wert aus der <b>EVarianteTrennzeichen</b> -Enumeration. Legt das verwendete Trennzeichen zwischen den einzelnen Datenelementen der Absenderadresse fest. <b>eTrennzeichenLF</b> – fügt eine Zeilenschaltung ein <b>eTrennzeichenCR</b> – für eine Absatzmarke ein <b>eTrennzeichenPunkt</b> – fügt einen Punkt ein
intAusrichtung	Ein Wert aus der <b>wdParagraphAlignment</b> -Enumeration. Legt die Ausrichtung der Adresse in der Fußzeile fest. <b>wdAlignParagraphLeft</b> – linksbündig <b>wdAlignParagraphCenter</b> – zentriert <b>wdAlignParagraphRight</b> – rechtsbündig

**Listing 23.11**    Prozedur zum Einfügen der Absenderadresse mit unterschiedlichen Darstellungsvarianten

```

Public Enum EVarianteTrennzeichen
    eTrennzeichenLF = 11
    eTrennzeichenCR = 13
    eTrennzeichenPunkt = 183
End Enum

Public Sub procFusszeileAdresseEinfügen ( _
    ByVal doc As Word.Document, _
    ByRef strAdresse() As String, _
    ByVal intTrennzeichen As EVarianteTrennzeichen, _
    ByVal intAusrichtung As WdParagraphAlignment)

    Dim rng As Word.Range
    Dim intFtr As WdHeaderFooterIndex
    Dim strText As String
    Dim strTrenner As String

    'Tatsächliches Trennzeichen bestimmen
    Select Case intTrennzeichen
        Case eTrennzeichenPunkt
            strTrenner = " " & Chr$(intTrennzeichen) & " "
        Case Else
            strTrenner = Chr$(intTrennzeichen)
    End Select

    'Absenderadresse aufbereiten
    For intFtr = LBound(strAdresse) To UBound(strAdresse)
        strText = strText & strTrenner & strAdresse(intFtr)
    Next intFtr
    strText = Mid$(strText, Len(strTrenner) + 1)

    'Alle Fußzeilen bearbeiten
    For intFtr = wdHeaderFooterPrimary To wdHeaderFooterEvenPages
        Set rng = doc.Sections(1).Footers(intFtr).Range
        rng.ParagraphFormat.Alignment = intAusrichtung
        rng.Text = strText
    Next intFtr
End Sub

```

Die Funktionsweise der Prozedur ist schnell erklärt. In einem ersten Schritt wird das Trennzeichen zwischen den Datenelementen der Absenderadresse aufbereitet. Ein zweiter Schritt ist für die Aufbereitung der eigentlichen Absenderadresse zuständig. Abschließend wird die Ausrichtung der Fußzeile festgelegt und dieser der gesamte Text zugewiesen. Dies geschieht unabhängig von den eingestellten Layouteigenschaften für alle drei Fußzeilenarten.

Der Punkt als Trennzeichen muss speziell aufbereitet werden, da diesem Zeichen ein zusätzliches Leerzeichen voran- und nachgestellt wird. Alle anderen Trennzeichen können direkt umgesetzt werden, weil innerhalb der `EVarianteTrennzeichen`-Enumeration die einzelnen Werte dem Zeichen-Code des einzusetzenden Zeichens entsprechen:

```
Case eTrennzeichenPunkt
    strTrenner = " " & Chr$(intTrennzeichen) & " "
Case Else
    strTrenner = Chr$(intTrennzeichen)
```

Die Absenderadresse wird innerhalb einer Schleife aufgebaut. Alle Elemente des Datenfelds werden in eine Zeichenkette übernommen. Zu jedem Eintrag wird das definierte Trennzeichen hinzugefügt. Da die Trennzeichen nur zwischen den einzelnen Elementen benötigt werden, wird jenes am Anfang der Zeichenkette anschließend wieder entfernt:

```
For intFtr = LBound(strAdresse) To UBound(strAdresse)
    strText = strText & strTrenner & strAdresse(intFtr)
Next intFtr
strText = Mid$(strText, Len(strTrenner) + 1)
```



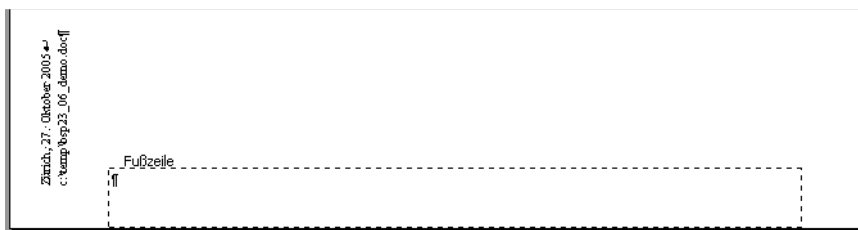
Die Prozedur in obigem Abschnitt finden Sie in der Beispieldatei *Bsp23\_05.dot*. Diese befindet sich auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap23*.

## Dateiname und Dokumentdatum setzen

Neben den eigentlichen Informationen, die ein Dokument enthält, werden oftmals noch zusätzliche Statusinformationen ausgegeben. Dazu gehören unter anderem der Dateiname oder ein Datum. In diesem Beispiel soll beides zusammen in der linken unteren Ecke des Dokuments eingetragen werden. Der Eintrag soll senkrecht am Papierrand erscheinen.

Damit die Prozedur für unterschiedliche Zwecke verwendet werden kann, sollen die beiden Informationen frei definiert und bei Bedarf sogar weggelassen werden können (vgl. Abbildung 23.7).

Abbildg. 23.7 Datum und Dateiname, eingefügt in einem unsichtbaren Textfeld in der Fußzeile des Dokuments



In Listing 23.12 sind die festgelegten Anforderungen an eine allgemeingültige Prozedur umgesetzt. Diese verfügt über vier Argumente.

**Tabelle 23.6** Eingabeargumente der Prozedur *procFusszeileDateinameDatumEinfügen*

Argument	Bedeutung
doc	Objektvariable auf das zu bearbeitende Dokument
intDateiname	Ein Wert aus der <b>EVarianteDateiname</b> -Enumeration. Legt fest, wie der Dateiname in dem Dokument ausgegeben wird. <b>eKeinDateiname</b> – kein Dateiname wird eingefügt <b>eNurDateiname</b> – das Feld <i>FileName</i> wird eingefügt, der Dateiname wird in dem Dokument ausgegeben. <b>eDateinameInklPfad</b> – das Feld <i>FileName</i> wird eingefügt, der Dateiname wird zusammen mit dem Speicherpfad ausgegeben.
intDatum	Ein Wert aus der <b>EVarianteDatum</b> -Enumeration. Legt fest, wie das Datum in dem Dokument ausgegeben wird. <b>eKeinDatum</b> – kein Datum wird eingefügt <b>eErstellDatum</b> – das Erstelldatum des Dokuments wird eingefügt <b>eSpeicherDatum</b> – das letzte Speicherdatum des Dokuments wird eingefügt <b>eDruckDatum</b> – das aktuelle Datum beim Ausdruck des Dokuments wird eingefügt <b>eAktuellesDatum</b> – das aktuelle Systemdatum wird als dynamisches Feld eingefügt <b>eHeuteAlsStatischesDatum</b> – das aktuelle Datum wird als statischer Text in das Dokument eingefügt
strOrtZumDatum	Optional. Zum Datum kann eine zusätzliche Bezeichnung des Ortes eingefügt werden.

**Listing 23.12** Prozedur zum Einfügen des Dateinamens und des Dokumentdatums mit definierbarem Format

```
Public Enum EVarianteDatum
    eErstellDatum = wdFieldCreateDate
    eSpeicherDatum = wdFieldSaveDate
    eDruckDatum = wdFieldPrintDate
    eAktuellesDatum = wdFieldDate
    eHeuteAlsStatischesDatum = wdFieldQuote
    eKeinDatum = 0
End Enum

Public Enum EVarianteDateiname
    eKeinDateiname = 0
    eNurDateiname = 1
    eDateinameInklPfad = 2
End Enum

Public Sub procFusszeileDateinameDatumEinfügen( _
    ByVal doc As Word.Document, _
    ByVal intDateiname As EVarianteDateiname, _
    ByVal intDatum As EVarianteDatum, _
    Optional strOrtZumDatum As String)

    Dim ftr As Word.HeaderFooter
    Dim rng As Word.Range
    Dim shp As Word.Shape
```

**Listing 23.12** Prozedur zum Einfügen des Dateinamens und des Dokumentdatums mit definierbarem Format (Fortsetzung)

```

Dim intFtr As WdHeaderFooterIndex
Dim strFeldText As String

'Alle Fußzeilen bearbeiten
For intFtr = wdHeaderFooterPrimary To wdHeaderFooterEvenPages
    Set ftr = doc.Sections(1).Footers(intFtr)

    'Textfeld für Dateiname, Datum usw. am linken Papierrand einfügen
    Set shp = ftr.Shapes.AddTextbox(Orientation:=msoTextOrientationUpward, _
        Left:=1, Top:=1, Width:=1, Height:=1, Anchor:=ftr.Range)
    With shp
        .Name = "Dateiname_" & CStr(Rnd())
        .Fill.Visible = msoFalse
        .Line.Visible = msoFalse
        With .TextFrame
            .MarginLeft = 0#
            .MarginRight = 0#
            .MarginTop = 0#
            .MarginBottom = 0#
        End With
        .RelativeHorizontalPosition = wdRelativeHorizontalPositionPage
        .RelativeVerticalPosition = wdRelativeVerticalPositionPage
        .Left = CentimetersToPoints(0.7)
        'Position in Abhängigkeit mit der Ausrichtung überarbeiten.
        If doc.Sections(1).PageSetup.Orientation = wdOrientPortrait Then
            .Top = CentimetersToPoints(11.7)
        Else
            .Top = CentimetersToPoints(3#)
        End If
        .Height = CentimetersToPoints(17#)
        .Width = CentimetersToPoints(0.8)
        .LockAnchor = True
        .WrapFormat.Type = wdWrapNone
        Set rng = .TextFrame.TextRange

    'Textfeld formatieren
    With .TextFrame.TextRange
        .ParagraphFormat.Style = wdStyleFooter
        With .Font
            .Size = 7
            .ColorIndex = wdGray25
        End With
    End With

    'Dateiname als Feld in Textfeld einfügen
    If Not intDateiname = eKeinDateiname Then
        If intDateiname = eDateinameInklPfad Then
            strFeldText = "\" & Lower & p"
        Else
            strFeldText = "\" & Lower"
        End If
        .Fields.Add Range:=rng, Type:=wdFieldFileName, _
            Text:=strFeldText, PreserveFormatting:=True
    End If

    'Zeilenschaltung einfügen

```

**Listing 23.12** Prozedur zum Einfügen des Dateinamens und des Dokumentdatums mit definierbarem Format (*Fortsetzung*)

```

        .InsertBefore (Chr$(11))
        rng.SetRange rng.Start, rng.Start

'Datum als Feld in Textfeld einfügen
    If Not intDatum = eKeinDatum Then
        If intDatum = eHeuteAlsStatischesDatum Then
            strFeldText = Chr$(34) & Format$(Now, "d. MMMM yyyy") & Chr$(34)
        Else
            strFeldText = "\@" & Chr$(34) & "d. MMMM yyyy" & Chr$(34) & _
                " \* MERGEFORMAT"
        End If
        .Fields.Add Range:=rng, Type:=intDatum, Text:=strFeldText, _
            PreserveFormatting:=True
        'Ort zum Datum
        .InsertBefore (strOrtZumDatum)
    End If
End With
End With
Next intFtr
End Sub

```

Die Prozedur fügt als erstes ein Textfeld in die Fußzeile des Dokuments ein. Dieses Textfeld wird formatiert und positioniert. Die Ausrichtung des Textes innerhalb des Textfeldes ist senkrecht zur Ausrichtung des Papiers. Da dem Textfeld weder eine Rahmen- noch eine Hintergrundfarbe zugewiesen wurde, bleibt es auf dem Ausdruck unsichtbar.

In einem weiteren Schritt werden die beiden Informationen, der Dateiname bzw. ein Dokumentdatum, in das Textfeld eingetragen.

Ob der Dateiname mit oder ohne Pfadangabe auf dem Dokument ausgegeben wird, wird über ein Argument des FileName-Feldes gesteuert:

```

If intDateiname = eDateinameInklPfad Then
    strFeldText = "\* Lower \p"
Else
    strFeldText = "\* Lower"
End If

```

Die Übergabe der Argumente an die Feldfunktion wird beim Hinzufügen des Field-Objektes im Text-Argument abgewickelt:

```

.Fields.Add Range:=rng, Type:=intDatum, Text:=strFeldText, PreserveFormatting:=True

```

Damit die unterschiedlichen Datumsfelder mit einigen wenigen Programmzeilen eingefügt werden konnten, wurden innerhalb der EVarianteDatum-Enumeration die einzelnen Werte mit den definierten Konstanten aus der WdFieldType-Enumeration gleichgesetzt.

Für das statische Datumsfeld wird ein Quote-Feld verwendet. Mit diesem Feld kann ein Text in ein Dokument eingefügt werden – also genau das, was mit dem statischen Datum bezweckt werden soll. Das Text-Argument dieses Feldes muss gesondert aufgebaut werden:

```

If intDatum = eHeuteAlsStatischesDatum Then
    strFeldText = Chr$(34) & Format$(Now, "d. MMMM yyyy") & Chr$(34)
Else
    strFeldText = "\" & Chr$(34) & "d. MMMM yyyy" & Chr$(34) & " \* MERGEFORMAT"
End If
.Fields.Add Range:=rng, Type:=intDatum, Text:=strFeldText, PreserveFormatting:=True

```



Die Prozedur in obigem Abschnitt finden Sie in der Beispieldatei *Bsp23\_06.dot*. Diese befindet sich auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap23*.

## Zusammenfassung

In diesem Kapitel wurden verschiedene Lösungsbeispiele aufgezeigt, die alle im Zusammenhang mit der Kopf- bzw. der Fußzeile stehen. Alle Beispiele können sehr schnell erweitert und an persönliche Bedürfnisse angepasst werden:

- Im Bereich der Kopfzeilen wurde gezeigt, wie man Seitenzahlen (Seite 801 ff.), Firmenlogo (Seite 803 ff.), Wasserzeichen (Seite 806 ff.) oder Falz- bzw. Lochmarken (Seite 810 ff.) einfügen kann.
- Im Bereich der Fußzeilen wurde aufgezeigt, wie man Absenderadresse (Seite 813 ff.) oder Dateiname bzw. Dokumentdatum (Seite 815 ff.) einfügen kann.





## Kapitel 24

# Grafiken mit Beschriftungen

### In diesem Kapitel:

Wissenschaftliche Figuren beschriften	822
Beschriftung und Grafik im Positionsrahmen	826
Zusammenfassung	829

Im Kapitel 6 wurden die Grundlagen der Arbeit mit dem Word-Objektmodell vorgestellt. Diese Lösung veranschaulicht die folgende Objekte.

---

**Im Blickpunkt:****Word-Objekte**

Dialog  
Field  
Frame  
InlineShape  
PageSetup  
Range  
Selection  
Table

---

Seit der Version 97 fügt Word Beschriftungen für Shape-Objekte (das Word-Objektmodell ist eingehend in Kapitel 6 erläutert) standardmäßig in Textfelder ein. Daraus entstehen zwei grundsätzliche Nachteile:

- Das Textfeld ist nicht mit dem grafischen Objekt verbunden. Wird Letzteres im Dokument verschoben, wandert das Textfeld nicht automatisch mit. Dieses Verhalten verwirrt den Benutzer und kostet viel Zeit.
- Beschriftungen in Textfeldern werden bei der automatischen Erstellung von Verzeichnissen und Querverweisen von Word nicht erkannt. Folglich fehlen diese Einträge in den Verzeichnissen. Es hilft also nicht, Grafik und Beschriftung in einem Textfeld oder Zeichnungsbereich zu vereinen.

Für das erste Problem gibt es mehrere Lösungswege. Für das zweite aber ist unabdingbar, dass die Beschriftung Teil des Dokumenttextes ist. Folglich muss die Grafik ein `InlineShape` sein, damit sie mit der Beschriftung zusammengehalten wird. Womit wir ein drittes Problem haben: Wie kann die Textflussformatierung trotzdem beibehalten werden?

Hier bieten sich zwei Alternativen an: die Grafik samt Beschriftung entweder in eine Tabelle oder in einen Positionsrahmen einzufügen. Beide Lösungen stellen wir in diesem Kapitel vor, wobei jene für die Tabelle eine Beschriftung auf wissenschaftliche Art, rechts neben der Grafik, erstellt.

## Wissenschaftliche Figuren beschriften

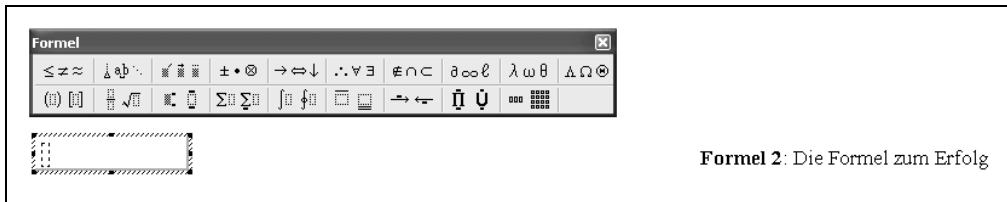
Die Benutzerschnittstelle von Word fügt Beschriftungen entweder ober- oder unterhalb des grafischen Objekts ein. Zwar können sie nachträglich verschoben werden, aber der Benutzer soll sich möglichst dem Dokumentinhalt widmen und nicht durch solche Nebensächlichkeiten abgelenkt werden. Deshalb ist es sinnvoll, ihm in der Vorlage für wissenschaftliche Arbeiten ein Werkzeug bereitzustellen, das ihm diese gestalterische Arbeit abnimmt.

Die Lösung in Listing 24.1 fordert zunächst den Benutzer auf, eine Beschriftung einzugeben, fügt diese dann in die rechte Spalte einer Tabelle ein und startet schließlich den Formel-Editor, wie in Abbildung 24.1 dargestellt.

**HINWEIS**

Es ist nicht möglich, mit Office-Bordmitteln programmtechnisch eine Formel im Formel-Editor zu erstellen, denn der Formel-Editor bietet keine Automatisierungsschnittstelle an. Sie müssten sich bei entsprechendem Bedarf nach einer Anwendung von Drittherstellern umsehen. Die Firma MathType hat den Formel-Editor an Microsoft lizenziert und wäre eine mögliche Kontaktstelle (<http://www.mathtype.com>).

Abbildg. 24.1 In wissenschaftlichen Arbeiten muss die Beschriftung oft rechts neben der Figur stehen.



Die Hauptprozedur *FormelEinfuegenUndBeschriften* weist die gegenwärtige Markierung einem Range-Objekt zu und kontrolliert, ob sie sich an einer für die Tabelle geeigneten Stelle befindet. Dies darf weder in einer bestehenden Tabelle noch mitten in einem Absatz sein. In beiden Fällen wird die Markierung zum Ende der Tabelle oder des Absatzes bewegt und ein neuer Absatz eingefügt. Erst dann wird die neue Tabelle erstellt.

Dies geschieht mit Hilfe der Funktion *TabelleErstellen*, die als Argument das Range-Objekt des Zielortes für die Tabelle erwartet. Eine Tabelle mit zwei Spalten und einer Reihe wird eingefügt. Befindet sich diese Tabelle unmittelbar unter einer bestehenden, betrachtet Word die beiden als Einheit, wodurch die Tabelle plötzlich mehr als eine Zeile hat. In diesem Fall wird sie von der anderen abgetrennt:

```
If tbl.Rows.Count > 1 Then tbl.Split BeforeRow:=rng.Rows(1)
```

Sie haben die Wahl, die Tabellenspalten dem Inhalt anzupassen oder umgekehrt den Inhalt dazu zu zwingen, eine vorgegebene Breite zu respektieren. Das vorliegende Beispiel nutzt die zweite Möglichkeit: Die Spaltenbreite wird beim Erstellen der Tabelle fixiert (*AutoFitBehavior:=wdAutoFitFixed*) und anschließend das Maß festgelegt (*tbl.Columns(1).Width*). Die linke Spalte für das grafische Objekt erstreckt sich über 60% der Textbreite; die rechte über die übrigen 40 %.

Zudem wird dafür gesorgt, dass die Beschriftung am unteren Rand der Tabellenzelle ausgerichtet wird (*tbl.Cell(1, 2).VerticalAlignment = wdCellAlignVerticalBottom*) und dass der Tabelleninhalt links und rechts mit dem Dokumenttext bündig steht. Hierfür wird der Abstand zwischen dem Tabellenrand und dem Inhalt entfernt (*tbl.LeftPadding = 0#*).

Die Ausführung kehrt an die rufende Hauptprozedur zurück, die als Zielbereich die Zelle für die Beschriftung festlegt und dann die Prozedur *BeschriftungEinfuegen* aufruft, die ihrerseits dieses Range-Objekt als Argument erwartet. Der Benutzer erhält nun in Form einer InputBox die Aufforderung zur Eingabe des Beschriftungstextes. Ist diese erfolgt, wird der Zielbereich mit der Formatvorlage *Beschriftung* formatiert (*rng.Style = wdStyleCaption*) sowie rechts ausgerichtet. (Die Ausrich-

tung muss nach der Zuweisung der Formatvorlage erfolgen, da diese eine bestehende Formatierung ersetzt.)

Die Beschriftung wird in drei Teilen eingefügt:

- die Bezeichnung (»Formel«)
- eine SEQ-Feldfunktion für die Nummerierung mit der Bezeichnung »Formel« (Diese Bezeichnung wird von Word benötigt, um Verzeichnisse und Querverweise zu erstellen.)
- ein Doppelpunkt als Trennzeichen, gefolgt von der Benutzereingabe, deren Fett-Formatierung anschließend entfernt wird

Interessant ist in diesem Teil der Lösung der Umgang mit dem Range-Objekt. Sie können deutlich sehen, wie es manipuliert wird, so dass jeder Teil dem vorherigen folgt (besonders nach Einfügen der Feldfunktion). Auch ist ersichtlich, wie Text gezielt formatiert wird.

Zum Schluss werden alle SEQ-Feldfunktionen im Dokument aktualisiert, damit die Nummerierung stimmt.

Zurück in der Hauptprozedur, wechselt der Zielbereich zur ersten Spalte, und die Einfügemarke wird hierher versetzt:

```
Set rng = tbl.Cell(1, 1).Range
rng.Collapse Direction:=wdCollapseStart
rng.Select
```

Zuletzt wird ein Formel-Objekt erstellt und die Kontrolle wieder dem Benutzer übergeben:

```
Set ils = rng.InlineShapes.AddOLEObject(ClassType:="Equation.3", FileName:="",
LinkToFile:=False, DisplayAsIcon:=False)
```

**Listing 24.1** Eine Beschriftung rechts ausgerichtet in einer Tabelle einfügen; das Objekt steht links daneben

```
Option Explicit

Sub FormelEinfuegenUndBeschriften()
    Dim rng As Word.Range
    Dim ils As Word.InlineShape
    Dim tbl As Word.Table
    Dim strBeschriftungsText As String

    strBeschriftungsText = InputBox("Bitte die Beschriftung eingeben")
    'Prüfen, ob keine Beschriftung eingegeben oder Abbrechen betätigt wurde
    If Len(strBeschriftungsText) = 0 Then Exit Sub

    Set rng = Selection.Paragraphs(1).Range
    'Falls die Markierung in einer Tabelle steht
    'sie ausserhalb der Tabelle verschieben
    If rng.Information(wdWithInTable) Then
        Set rng = rng.Tables(1).Range
        rng.Collapse Direction:=wdCollapseEnd
        rng.InsertAfter vbCrLf
        rng.Collapse Direction:=wdCollapseEnd
    End If
```

**Listing 24.1** Eine Beschriftung rechts ausgerichtet in einer Tabelle einfügen; das Objekt steht links daneben (*Fortsetzung*)

```
'Falls die Markierung in einem Absatz mit Text steht,
'die Tabelle in einen leeren einfügen, der dem
'aktuellen folgt
'Testen, ob der aktuellen Absatz "leer" ist
If Len(rng.Paragraphs(1).Range.Text) > 1 Then
    rng.Collapse Direction:=wdCollapseEnd
    rng.InsertBefore vbCr
    rng.Collapse Direction:=wdCollapseStart
End If

Set tbl = TabelleErstellen(rng)
'Zuerst die Beschriftung rechts einfügen, da der Benutzer
'im Abschluss die Formel eingeben muss
Set rng = tbl.Cell(1, 2).Range
rng.Collapse Direction:=wdCollapseStart
BeschriftungEinfuegen rng, strBeschriftungsText

'Der Formel-Editor in der ersten Spalte starten
Set rng = tbl.Cell(1, 1).Range
rng.Collapse Direction:=wdCollapseStart
rng.Select
Set ils = rng.InlineShapes.AddOLEObject(ClassType:="Equation.3", FileName="", _
    LinkToFile:=False, DisplayAsIcon:=False)
End Sub

Private Function TabelleErstellen(ByRef rng As Word.Range) As Word.Table
    Dim tbl As Word.Table
    Dim pgs As Word.PageSetup
    Dim sngTextBreite As Single
    Dim cel As Word.Cell

    'Die Tabelle einfügen. Spalte links für die Formel,
    'Spalte rechts für die Beschriftung.
    'Die Formel wird sich der Breite der linken Spalte anpassen
    Set tbl = rng.Tables.Add(Range:=rng, NumRows:=1, _
        NumColumns:=2, AutoFitBehavior:=wdAutoFitFixed)
    'Sicherstellen, dass sich die neue Tabelle nicht mit einer anderen
    'direkt daneben befindlichen verbunden hat
    If tbl.Rows.Count > 1 Then tbl.Split BeforeRow:=rng.Rows(1)
    'Die erste Spalte soll 2/3 der Seite, zwischen den Seitenränder, breit sein
    Set pgs = rng.Sections(1).PageSetup
    sngTextBreite = (pgs.PageWidth - pgs.LeftMargin - pgs.RightMargin)
    tbl.Columns(1).Width = 0.6 * sngTextBreite
    tbl.Columns(2).Width = 0.4 * sngTextBreite
    'Beschriftung unten in der Tabellenzelle ausrichten
    tbl.Cell(1, 2).VerticalAlignment = wdCellAlignVerticalBottom
    'Der Text soll bündig mit dem Dokumenttext stehen
    tbl.LeftPadding = 0#
    tbl.RightPadding = 0#
    Set TabelleErstellen = tbl
End Function

Private Sub BeschriftungEinfuegen(ByRef rng As Word.Range, strBeschriftungsText)
    Dim fld As Word.Field
```

**Listing 24.1** Eine Beschriftung rechts ausgerichtet in einer Tabelle einfügen; das Objekt steht links daneben (*Fortsetzung*)

```
'Die Zelle mit der Beschriftung-Formatvorlage formatieren
rng.Style = wdStyleCaption
'Beschriftung rechts ausrichten
rng.Paragraphs.Alignment = wdAlignParagraphRight
rng.Text = "Formel "
rng.Collapse Direction:=wdCollapseEnd
'Nach Bezeichnung eine SEQ-Feldfunktion für die Nummerierung einfügen
Set fld = rng.Fields.Add(Range:=rng, Text:="SEQ Formel", _
PreserveFormatting:=False)
'Nach der Feldfunktion kommt die Beschriftung, die nicht mehr fett ist
Set rng = fld.Result
rng.Collapse Direction:=wdCollapseEnd
rng.MoveStart Unit:=wdCharacter, Count:=1
rng.Text = ": " & strBeschriftungsText
rng.Font.Bold = False
'Alle SEQ-Feldfunktionen aktualisieren
For Each fld In rng.Parent.Fields
    If fld.Type = wdFieldSequence Then fld.Update
Next
End Sub
```

## Beschriftung und Grafik im Positionsrahmen

Für das Beispiel in Abbildung 24.2 wurde eine Grafik samt zugehöriger Beschriftung in einen Positionsrahmen eingefügt. Wird dieser verschoben, wandern Grafik und Beschriftung mit. Die Prozedur in Listing 24.2 passt dabei die Größe des Positionsrahmens der Grafik an. Es ist aber umgekehrt auch möglich, die Größe der Grafik dem Behälter anzupassen: Erstellen Sie zuerst den Positionsrahmen und sorgen Sie dafür, dass sich der Zielbereich im Positionsrahmen befindet. Fügen Sie dann die Grafik als `InlineShape` ein.

**Abbildg. 24.2** Die rechtsbündige Grafik steht zusammen mit ihrer Beschriftung in einem Positionsrahmen.

Mit der Prozedur *GrafikInPosRahmenMitBeschriftung* (die Symbolschaltfläche "mit Textfluss") wird sie zusammen mit der Grafik in einen Positionsrahmen, rechts ausgerichtet mit Textflussformatierung, eingefügt. Grafik und Beschriftung bleiben damit zusammen, wenn der Positionsrahmen verschoben wird. Zudem erscheint die Beschriftung im Inhaltsverzeichnis.



Abbildung 1 Northwinds Nancy Daviolo

Zuerst prüft die Prozedur *GrafikInPosRahmenMitBeschriftung*, ob der Zielbereich (Absatz, worin sich die Einfügemarke befindet) das Einfügen eines Positionsrahmens unterstützt. Das bedeutet, der Absatz muss sich im Dokumenttextkörper befinden und darf nicht in einer Tabelle, Endnote, Fußnote oder Ähnlichem enthalten sein. (Beachten Sie hier die Verwendung der Eigenschaft *Selection.Type*.) Sofern kein Problem gefunden wird, läuft die Ausführung weiter, ansonsten wird zum Ausstiegspunkt gesprungen.

Ist alles soweit in Ordnung, wird der Benutzer aufgefordert, eine Grafikdatei auszuwählen. Die Prozedur zeigt das Word-eigene Dialogfeld *Grafik einfügen* an mit der Methode *Display*. Wie in Kapitel 14 erläutert, wird mit dieser Methode das Dialogfeld nur angezeigt, aber nicht von Word ausgewertet. Stattdessen werden die Einstellungen für die weitere Bearbeitung in Variablen gespeichert.

Wählt der Benutzer *Abbrechen*, springt der Code zum Ausstiegspunkt. Andernfalls wird die Variable *lAuswahl* dahingehend ausgewertet, ob der Benutzer die Grafik mit oder ohne Verknüpfung einfügen möchte, und wenn mit, ob die Grafikdaten im Dokument gespeichert werden sollen. Die Werte der Variablen *bLinkToFile* und *bSaveWithDoc* werden entsprechend festgelegt und in der *AddPicture*-Methode verwendet.

Die eingefügte Grafik wird markiert, dann der Positionsrahmen eingefügt und rechts ausgerichtet (*fram.HorizontalPosition = wdFrameRight*). Es fällt auf, dass sich diese Prozedur ausschließlich des *Selection*- und nicht des *Range*-Objekts bedient. Das hat zwei Gründe:

- Wir brauchen zu Beginn der Prozedur die Eigenschaft *Selection.Type*, um festzustellen, ob der Zielbereich gültig ist.
- Der Positionsrahmen orientiert sich an der gegenwärtigen Markierung, dies obwohl die Methode *Frames.Add* ein *Range*-Argument hat. Soll – wie in diesem Fall – bestehender Text in einen Positionsrahmen aufgenommen werden, muss mit dem *Selection*-Objekt gearbeitet werden.

Nach Einfügen des Positionsrahmens ist die Grafik immer noch markiert. Eine Eigenart des Positionsrahmens ist, dass falls er nur eine Grafik enthält, die beiden »verschmelzen«; Word »sieht« nur die Grafik. Deshalb müssen weitere Zeichen in den Positionsrahmen eingefügt werden, bevor mit dem Inhalt weitergearbeitet werden kann. Da sich die Beschriftung unterhalb der Grafik befinden soll, wird ein Absatz nach der Grafik benötigt. Zu diesem Zweck wird die Markierung auf einen Punkt verkleinert (*Selection.Collapse wdCollapseEnd*) und der Absatz eingefügt.

Zum Schluss blendet die Prozedur das Dialogfeld *Beschriftung* ein.

**Listing 24.2** Eine »mit Text in Zeile« formatierte Grafik in einen Positionsrahmen einfügen

```
Sub GrafikInPosRahmenMitBeschriftung()
    Dim strBildName As String
    Dim lAuswahl As Long
    Dim bLinkToFile As Boolean
    Dim bSaveWithDoc As Boolean
    Dim ils As Word.InlineShape
    Dim fram As Word.Frame

    ' Die Markierung darf/soll nicht in einer Tabelle, Positionsrahmen o.ä. stehen.
    If (Selection.Type <> wdSelectionIP And Selection.Type <> wdSelectionNormal) _
        Or Selection.Information(wdWithInTable) Or _
        Selection.Information(wdInEndnote) Or _
        Selection.Information(wdInFootnote) Then
        MsgBox "Die Markierung muss im Dokumenttext stehen.",_
```

**Listing 24.2** Eine »mit Text in Zeile« formatierte Grafik in einen Positionsrahmen einfügen (*Fortsetzung*)

```

        vbOKOnly + vbCritical, strMSG_TITEL
        GoTo Fertigstellen
    End If
    ' Bild Dateiname, und Art der Einfügung
    With Dialogs(wdDialogInsertPicture)
        ' Wenn nicht "Einfügen" gewählt wurde, abbrechen.
        If .Display <> -1 Then GoTo Fertigstellen
        lAuswahl = .LinkToFile
        strBildName = .Name
    End With

    ' Feststellen, ob Bild verknüpft und/oder im Dokument zu speichern ist.
    Select Case lAuswahl
        Case 0
            bLinkToFile = False
            bSaveWithDoc = True
        Case 1
            bLinkToFile = True
            bSaveWithDoc = True
        Case 2
            bLinkToFile = True
            bSaveWithDoc = False
        Case Else
    End Select
    ' Bild einfügen...
    Set ils = ActiveDocument.InlineShapes.AddPicture(FileName:=strBildName, _
        LinkToFile:=bLinkToFile, SaveWithDocument:=bSaveWithDoc, Range:=Selection.Range)
    ' ...und markieren.
    ils.Select
    ' Positionsrahmen um die Markierung einfügen...
    Set fram = Selection.Frames.Add(Range:=Selection.Range)
    ' ...und rechts ausrichten.
    fram.HorizontalPosition = wdFrameRight
    ' Neuen Absatz für die Beschriftung nach dem Bild einfügen.
    Selection.Collapse wdCollapseEnd
    Selection.Text = vbCr
    ' Benutzer für die Beschriftung auffordern.
    Dialogs(wdDialogInsertCaption).Show

Fertigstellen:

End Sub

```



Die Beispieldatei *Bsp24\_01.doc* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap24*.



# Zusammenfassung

In diesem Kapitel wurden zwei Ergänzungen zu der von Word angebotener Funktionalität für Beschriftungen vorgestellt.

- Im Abschnitt »Wissenschaftliche Figuren beschriften« (Seite 822) wird erläutert, wie eine rechts ausgerichtete Beschriftung neben dem Objekt positioniert wird.
- Wie eine Beschriftung mit seinem Objekt zusammengehalten werden kann, steht im Abschnitt »Beschriftung und Grafik im Positionsrahmen« (Seite 826) beschrieben. Obwohl Beschriftung und Objekt mit Textfluss formatiert sind, wird die Beschriftung von der Verzeichnis-Funktionalität erkannt.



## Kapitel 25

# Rechtschreibfehler für den Ausdruck formatieren

### In diesem Kapitel:

Wie die Lösung funktioniert

833

Zusammenfassung

838

Das Word-Objektmodell haben Sie bereits in Kapitel 6, und UserForms in Kapitel 14 kennen gelernt. Im vorliegenden Kapitel finden Sie nun eine Lösung, in der die folgenden Themen veranschaulicht werden. Der programmtechnische Umgang mit Rechtschreibfehlern wurde bislang nicht behandelt und ist neu.

---

**Im Blickpunkt:**

Document-Objekt

    Add-Methode

    SaveAs-Methode

    Document-Objekt:SpellingErrors

Range-Objekt

    ConvertToTable-Methode

    Font

    Information

Selection-Objekt

    GoTo-Methode

    HomeKey

    Word-interne Textmarken

UserForm

    Kontrollkästchen

    Combobox-Steuerelement

    Tag

---

Seit Word 95 können Rechtschreibfehler im Text mit wellenförmigen Unterstrichen gekennzeichnet werden. Diese werden jedoch nicht ausgedruckt. Gelegentlich besteht der Bedarf – vor allem in Lernumgebungen –, diese auszudrucken oder in einer Liste zusammenzutragen.

Der vorliegende Lösungsvorschlag zeigt, wie das Objekt-Modell von Word Rechtschreibfehler handhabt. Grammatikfehler können auf ähnliche Weise bearbeitet werden.



---

Die Beispieldatei *Bsp25\_01.dot* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap25*. Speichern Sie die Datei im *StartUp*-Ordner von Word, um das Werkzeug beim Starten von Word automatisch mitzuladen.

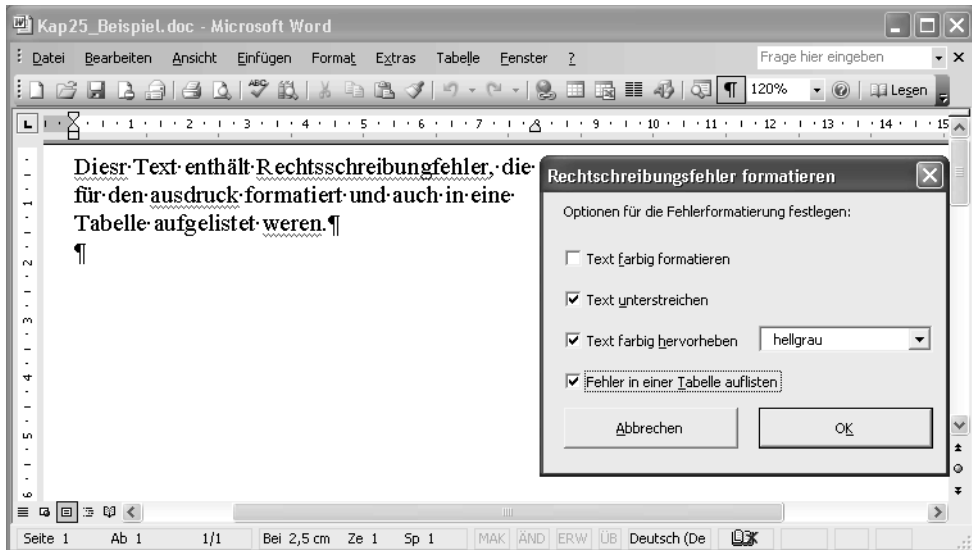
---

# Wie die Lösung funktioniert

Beim Aufruf des Werkzeugs, um Rechtschreibfehler zu formatieren, wird ein Formular wie das in Abbildung 25.1 eingeblendet. Darin stehen dem Anwender vier Optionen zur Auswahl, von denen er so viele aktivieren darf, wie er möchte.

Beim Aktivieren einer der Optionen, mit denen Text farbig formatiert oder hervorgehoben werden kann, wird eine ComboBox mit einer Liste von 16 Farben eingeblendet. Diese sind in der Reihenfolge ihres `WdColorIndex`-Werts aufgelistet; die Farbe wird anhand seiner Position in der Liste bestimmt.

Abbildg. 25.1 Optionen für die Formatierung der Rechtschreibfehler im Ausdruck



Der Code für das Formular steht in Listing 25.1. Die erste Prozedur – *RechtschreibfehlerOptionen* –, die sich in einem gewöhnlichen Modul befindet, blendet das Formular ein und speichert die Benutzereinstellungen nach deren Bestätigung für den weiteren Verlauf. Das Formular wird dann geschlossen und aus dem Speicher entfernt, bevor die Prozedur für die Formatierung (Listing 25.2) aufgerufen wird.

**HINWEIS** Mehr über die Arbeit mit VBA-Formularen (UserForms) finden Sie in Kapitel 14 beschrieben.

Die weiteren Prozeduren dieses Listings steuern das Formular. Diese sind:

- Zwei Ereignisprozeduren für die Kontrollkästchen für die Schriftfarbe und die Hervorhebung. Sie blenden die entsprechenden ComboBoxen ein und aus.
- Zwei Ereignisprozeduren für die Schaltflächen *Abbrechen* und *OK*. In beiden Fällen wird das Formular ausgeblendet. Der Wert des Tag-Steuerelements hält fest, welche Schaltfläche betätigt wurde.

- *UserForm\_Initialize* wird ausgeführt beim Laden des Formulars. Die Listen der ComboBoxen werden gefüllt, und es wird sichergestellt, dass diese nicht sichtbar sind.
- Die letzte Prozedur – *FarbenAuflisten* – füllt die ComboBoxen. Sie steht in einem gewöhnlichen Modul, das mit `Option Private Module` deklariert wurde. Diese Deklaration sorgt dafür, dass die darin enthaltenen Prozeduren und globalen Variablen außerhalb dieses Projekts nicht sichtbar sind (es handelt sich um das VBA-Gegenstück zur Deklaration `Friend` in der klassischen Visual Basic-Sprache).

**Listing 25.1** Prozeduren rund um das Formular

```
'Code in einem gewöhnlichen Modul
Public Sub RechtschreibfehlerOptionen()
    Dim frm As frmRSFehlerFormatieren

    Set frm = New frmRSFehlerFormatieren
    frm.Show
    If frm.Tag = "1" Then
        bFarbigFormatieren = frm.chkFarbig
        bMitUnterstrichFormatieren = frm.chkUnterstreichen
        bMitHervorhebungFormatieren = frm.chkHervorheben
        bInTabelleSchreiben = frm.chkTabelle
        lFarbeText = frm.cboFarbeText.ListIndex
        lFarbeHervorhebung = frm.cboFarbeHervorheben.ListIndex
    End If
    Unload frm
    Set frm = Nothing
    RechtschreibfehlerFuerDenAusdruckFormatieren
End Sub

'Code im Klassenmodul des UserForms
Private Sub chkFarbig_Click()
    SichtbarkeitFestlegen chkFarbig, cboFarbeText
End Sub

Private Sub chkHervorheben_Click()
    SichtbarkeitFestlegen chkHervorheben, cboFarbeHervorheben
End Sub

Private Sub cmdAbbrechen_Click()
    Me.Tag = "0"
    Me.Hide
End Sub

Private Sub cmdOK_Click()
    Me.Tag = "1"
    Me.Hide
End Sub

Private Sub UserForm_Initialize()
    FarbenAuflisten Me.cboFarbeText
    FarbenAuflisten Me.cboFarbeHervorheben
    Me.cboFarbeText.Visible = False
    Me.cboFarbeHervorheben.Visible = False
End Sub
```

Listing 25.1 Prozeduren rund um das Formular (Fortsetzung)

```

Option Private Module
Public Sub FarbenAuflisten(cbo As MSForms.ComboBox)
    cbo.AddItem ""
    cbo.AddItem "schwarz"
    cbo.AddItem "blau"
    cbo.AddItem "türkis"
    cbo.AddItem "hellgrün"
    cbo.AddItem "rosa"
    cbo.AddItem "rot"
    cbo.AddItem "gelb"
    cbo.AddItem "weiß"
    cbo.AddItem "dunkelblau"
    cbo.AddItem "dunkelgrünblau"
    cbo.AddItem "grün"
    cbo.AddItem "violett"
    cbo.AddItem "dunkelrot"
    cbo.AddItem "dunkelgelb"
    cbo.AddItem "dunkelgrau"
    cbo.AddItem "hellgrau"
End Sub

Public Sub SichtbarkeitFestlegen(chk As MSForms.CheckBox, ctl As MSForms.Control)
    If chk.Value = True Then
        ctl.Visible = True
    Else
        ctl.Visible = False
    End If
End Sub

```

Das Betätigen der Schaltfläche *OK* löst die Prozedur *RechtschreibfehlerFuerDenAusdruckFormatieren* in Listing 25.2 aus. Zuerst wird kontrolliert, ob das aktuelle Dokument Rechtschreibfehler enthält. Wenn nicht, wird die Ausführung abgebrochen.

Sonst wird das Dokument gespeichert, dann nochmals (im gleichen Pfad) unter einen neuen Namen, der um den Text im Konstantwert *strNeueDateiEndung* erweitert wurde.

Da der Umfang der durch die Rechtschreibprüfung erkannten Fehler durch den Arbeitsspeicher beschränkt ist, empfiehlt es sich, einzelne Bereiche statt des ganzen Dokuments durcharbeiten. Deshalb wird von der ersten bis zur letzten Seite »geblättert« und der Inhalt einer Range-Variablen zugewiesen.

Jedem Rechtschreibfehler im jeweiligen Bereich wird die vom Anwender festgelegte Formatierung zugewiesen. Falls er die Fehler in einer Tabelle auflisten möchte, werden Seitenzahl, fehlerhafter Text sowie der Satz, in dem sich der Fehler befindet, einer zeichengetrennten Zeichenkette hinzugefügt. Diese wird schließlich in ein neues Dokument eingefügt und in eine Tabelle (Abbildung 25.2) konvertiert.

Abbildg. 25.2 Liste der im Dokument gefundenen Rechtschreibfehler

Seite	fehlerhafter Text	Kontext
1	Dieser	Dieser Text enthält Rechtschreibfehler, die für den Ausdruck formatiert und auch in eine Tabelle aufgelistet werden.
1	Rechtsschreibungsfehler	Dieser Text enthält Rechtschreibfehler, die für den Ausdruck formatiert und auch in eine Tabelle aufgelistet werden.
1	ausdruck	Dieser Text enthält Rechtschreibfehler, die für den Ausdruck formatiert und auch in eine Tabelle aufgelistet werden.
1	werden	Dieser Text enthält Rechtschreibfehler, die für den Ausdruck formatiert und auch in eine Tabelle aufgelistet werden.

Die Abbildung 25.3 veranschaulicht die Wirkung der Optionen, fehlerhaften Text zu unterstreichen sowie hervorzuheben.

Abbildg. 25.3 Die Rechtschreibfehler wurden unterstrichen und hervorgehoben

Dieser Text enthält Rechtsschreibungsfehler, die für den Ausdruck formatiert und auch in eine Tabelle aufgelistet werden.

Listing 25.2 Die Prozeduren, um Rechtschreibfehler zu formatieren und in einer Tabelle auflisten

```
Option Private Module

Public Const strNeueDateiEndung As String = "_RSFehler"
Public Const strTrennzeichen As String = "|"

Public bFarbigFormatieren As Boolean '= False
Public bMitUnterstrichFormatieren As Boolean '= True
Public bMitHervorhebungFormatieren As Boolean '= False
Public bInTabelleSchreiben As Boolean '= False
Public lFarbeText As Long '= 0
Public lFarbeHervorhebung As Long '= 0

Sub RechtschreibfehlerFuerDenAusdruckFormatieren()
    Dim doc As Word.Document
    Dim strDokNeuerName As String
    Dim rng As Word.Range
    Dim lSeitenZaehler As Long
    Dim rngRSFehler As Range
    Dim strTabellendaten As String

    Set doc = ActiveDocument
    'Die Arbeit speichern, damit keine Daten verloren gehen
    doc.Save
```



Listing 25.2 Die Prozeduren, um Rechtschreibfehler zu formatieren und in einer Tabelle auflisten (Fortsetzung)

```

'Falls keine Rechtschreibfehler vorhanden sind, abbrechen.
If doc.SpellingErrors.Count > 0 Then
    MsgBox "Keine Rechtschreibfehler gefunden."
    Exit Sub
End If

'Dokument für den formatierten Fehler unter einem anderen Namen
'im gleichen Pfad speichern wie das ursprüngliche.
strDokNeuerName = NeuerDateiName(doc, strNeueDateiEndung)
doc.SaveAs FileName:=doc.Path & "\" & strDokNeuerName, AddToRecentFiles:=False

'Auf der ersten Seite anfangen
Selection.HomeKey wdStory
Set rng = ActiveDocument.Range

'Die Rechtschreibprüfung hat Mühe in einem großen Dokument mit vielen Fehlern.
'Deshalb werden einzelne Seiten abgearbeitet.
For lSeitenZaehler = 1 To doc.Range.Information(wdNumberOfPagesInDocument)
    'Den Text einer Seite einem Bereich mit Hilfe der \Page Textmarke zuweisen
    Set rng = Selection.Bookmarks("\Page").Range

    'Die Rechtschreibfehler den gewünschten Einstellungen entsprechend formatieren.
    For Each rngRSFehler In rng.SpellingErrors
        With rngRSFehler
            'Die Eigenschaft ColorIndex ist gültig in allen Word-Versionen
            If bFarbigFormatieren = True Then .Font.ColorIndex = lFarbeText
            If bMitUnterstrichFormatieren = True Then .Font.Underline = wdUnderlineDotDash
            If bMitHervorhebungFormatieren = True Then _
                rngRSFehler.HighlightColorIndex = lFarbeHervorhebung

            'Falls die Fehler in einer Tabelle aufgelistet werden sollen, werden sie in einer
            'Zeichenkette gesammelt, die am Schluss in eine Tabelle konvertiert wird.
            If bInTabelleSchreiben = True Then
                strTabellenDaten = strTabellenDaten & CStr(lSeitenZaehler) & _
                    & strTrennzeichen & rngRSFehler.Text & strTrennzeichen & _
                    TrimEndParas(rngRSFehler.Sentences(1).Text) & vbCrLf
            End If
        End With
    Next rngRSFehler
    Selection.GoTo What:=wdGoToPage, Which:=wdGoToNext
Next lSeitenZaehler

'Die Tabelle in einem neuen Dokument erstellen
If bInTabelleSchreiben = True Then
    Dim rngTabelle As Word.Range
    Dim tbl As Word.Table
    Dim docFehlerLog As Word.Document

    Set docFehlerLog = Documents.Add
    Set rngTabelle = docFehlerLog.Range

    'Die letzte Absatzmarke abschneiden
    strTabellenDaten = Left(strTabellenDaten, Len(strTabellenDaten) - 1)

    'Spaltenüberschriften hinzufügen und die zeichengetrennte Liste einfügen

```

**Listing 25.2** Die Prozeduren, um Rechtschreibfehler zu formatieren und in einer Tabelle auflisten (*Fortsetzung*)

```

    rngTabelle.Text = "Seite" & strTrennzeichen & "Fehlbarer Text" & _
        strTrennzeichen & "Kontext" & vbCrLf & strTabellenDaten

    '... und den Bereich in eine Tabelle umwandeln
    Set tbl = rngTabelle.ConvertToTable(Separator:=strTrennzeichen, Numcolumns:=3)

    'Die erste Zeile fett formatieren.
    tbl.Rows(1).Range.Font.Bold = True
End If
End Sub

'Die Datei-Endung ersetzen und den neuen Dateinamen zurückgeben
Function NeuerDateiName(doc As Word.Document, ext As String) As String
    Dim s As String
    Dim loc As Long

    s = doc.Name
    loc = InStr(s, ".doc")
    s = Left(s, loc - 1)
    NeuerDateiName = s & ext
End Function

'Absatzmarken am Ende einer Zeichenkette entfernen
Function TrimEndParas(s As String) As String
    Do While Asc(Right(s, 1)) = 13
        s = Left(s, Len(s) - 1)
    Loop
    TrimEndParas = s
End Function

```

## Zusammenfassung

Dieser Lösungsvorschlag stellt den Umgang mit Rechtschreibfehlern (SpellingErrors) vor. Er veranschaulicht zudem, wie Farben in einer ComboBox-Liste aufgenommen und ausgewertet werden.

## Kapitel 26

# Rechnung erstellen mit Formularfeldern

### In diesem Kapitel:

Formular zur Bestätigung von Bestellungen

840

Zusammenfassung

848

Das Word-Objektmodell haben Sie bereits in Kapitel 6, ADO in Kapitel 10 und UserForms in Kapitel 14 kennen gelernt. Im vorliegenden Kapitel finden Sie nun eine Lösung, in der die folgenden Themen veranschaulicht werden, wobei das Hauptgewicht auf dem Einsatz von Formularfeldern liegt.

---

**Im Blickpunkt:**

ADO-Verbindung zur Access-Datenbank

Formular

Formularfeld Optionen

FormField.Name-Eigenschaft

schützen

Table

Rows.Insert

Rows.Delete

Zelleninhalt mit Range.FormattedText kopieren

UserForm

Instanz wiederverwenden

ListBox-Steuerelement

Tag

---

Eine logische Verwendung für Formularfelder ist die Erstellung von Offerten, Lieferscheinen, Bestellungsbestätigungen, oder Rechnungen. Produkte können aus Dropdown-Feldern gewählt, Preise automatisch eingetragen und Berechnungen automatisch durchgeführt werden. Eine tolle Sache ... wäre nicht eine variable Anzahl von Posten zu berücksichtigen. Da ist eine automatisierte Lösung unabdingbar, um für jeden neuen Posten einen Satz von Formularfeldern einzufügen oder auch überzählige zu entfernen.

## Formular zur Bestätigung von Bestellungen

In Abbildung 26.1 ist eine solche Bestellungsbestätigung mit Formularfeldern (hier grau hinterlegt) dargestellt. Bei Eintritt in ein Formularfeld der Spalte »Beschreibung« wird die UserForm mit einer Produktliste eingeblendet. Nach dem Betätigen der Schaltfläche *Übernehmen* wird der ausgewählte Listeneintrag in das aktuelle Feld eingefügt. Zudem werden Produkt-ID sowie Einzelpreis in die entsprechenden Formularfelder der gleichen Tabellenzeile übertragen. Anschließend wird das Formularfeld für die Anzahl der Artikel aktiviert, so dass der Anwender sofort mit der Eingabe fortfahren kann. Den Code für diese Handlungen finden Sie in Listing 26.1.

**HINWEIS** Die graue Schattierung der Formularfelder kann mit der Schaltfläche *Formularfeld-Schattierung* der *Formular-Symbolleiste* ausgeblendet werden. Die entsprechende Eigenschaft im Objektmodell ist `FieldShading`:

```
doc.ActiveWindow.View.FieldShading = False
```

Die Schaltflächen der Symbolleiste *Rechnung schreiben* dienen dazu, neue Zeilen in die Tabelle einzufügen bzw. die aktuelle Zeile zu löschen. Der zugehörige Code steht in Listing 26.2 bereit.

Abbildg. 26.1 Eine auf Formularfeldern basierende Bestellungsbestätigung mit Hilfe von Makros schreiben

The screenshot shows a spreadsheet application with a form for order confirmation. The form includes text fields for recipient address, subject, and signature, and a table for order items. A 'Produkt auswählen' dialog box is open, showing a list of products. A 'Rechnung schreiben' button is visible at the bottom right.

Prod.-ID	Beschreibung	Anzahl	Stückpreis	Preis
60	Camembert-Pierrot	1	17,00	17,00 €
20	Sir-Rodney's-Marmalade	10	40,50	405,00 €
18	Camaron-Tigers	2	31,25	62,50 €
			<b>Total</b>	<b>484,50 €</b>

Etwas konkreter ist der Aufbau der Rechnungsdokumentvorlage in Abbildung 26.2 veranschaulicht. Für die Anschrift des Empfängers sind Textformularfelder mit einem beschreibenden Standardtext wie »Anrede« oder »Ort« vorgesehen, der später durch die Benutzereingabe ersetzt wird.

Die Grußformel enthält *If*- sowie *Ref*-Feldfunktionen, die sich auf die Adressfelder »Anrede« und »Nachname« beziehen. Entspricht der Inhalt des Formularfeldes `txtAnrede` der Zeichenfolge »Herr«, wird das Wort »geehrte« um den Buchstaben »r« ergänzt. Die *Ref*-Feldfunktionen geben den Inhalt der Formularfelder `txtAnrede` und `txtNachname` unverändert wieder. Um diese Angaben dynamisch anzuzeigen, wurde für die beiden Formularfelder das Kontrollkästchen *Beim Verlassen berechnen* im Dialogfeld *Formularfeld-Optionen* aktiviert.

**HINWEIS** Mehr zum Thema Feldfunktionen steht in Kapitel 6 beschrieben.

Abbildg. 26.2 Die Dokumentvorlage für das Rechnungsformular

«Anrede»«Vorname»«Nachname»  
«Straße»  
«PLZ»«Ort»  
Betreff: «Betreff»  
Sehr geehrte(r) If: Ref.txtAnrede: Herr. "r". "" Ref.txtAnrede: Ref.txtNachname:  
besten Dank für Ihre Bestellung vom:   

Prod.-ID	Beschreibung	Anzahl	Stückpreis	Preis
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	0,00-€
			<b>Total</b>	<b>0,00-€</b>

«Notizen»  
Unterschrift

Rechnung schreiben  
Zelle hinzufügen | Aktuelle Zelle löschen

Dem Textformularfeld für das Bestelldatum, unterhalb der Grußformel, wurde der Typ »Datum« zugewiesen. Somit kann der Anwender nur ein gültiges Datum und keinen anderen Text eingeben.

In den *Formularfeld-Optionen* für die Spalten »Prod.-ID« und »Stückpreis« wurde das Kontrollkästchen *Eingabe zulassen* ausgeschaltet. Für die Inhalte dieser Spalten ist der Automatisierungscode verantwortlich, der Anwender darf sie nicht anwählen.

Die Textformularfelder der Spalte »Preis« sind vom Typ *Berechnung* mit dem Zahlenformat #.##0,00 €; (#.##0,00 €). Der Preis für jede Zeile wird mit der Formel =Product(Left), die Spaltensumme mit der Formel =Sum(Above) berechnet.

#### HINWEIS

Aggregatfunktionen wie Sum und Left sind die einzige Methode, Berechnungen in Word-Tabellen relativ durchzuführen. Die Inhalte aller Zellen, bis zur ersten ohne numerischen Wert, werden in die Funktion miteinbezogen.

Die Prozedur *ProduktListeEinblenden* (Listing 26.1) wurde aus der Dropdownliste *Ereignis* in den *Formularfeld-Optionen* für die Formularfelder der Spalte »Beschreibung« gewählt. Die UserForm in Abbildung 26.1 wird eingeblendet. Während deren Initialisierung wird eine ADO-Verbindung (Connection) zur *Nordwind*-Datenbank hergestellt (die Prozedur *AccessDatenHolen*), um die Artikel-liste in einen ADO-Datensatz (Recordset) zu lesen. Dieser wird der Prozedur *ListeFuellen* übergeben, wo in einer Schleife der Feldinhalt in die Listenspalten geschrieben wird (wovon nur die erste Spalte angezeigt wird).

#### TIPP

Um eine Spalte in einem Listensfeld oder in einer Dropdownliste zu unterdrücken, setzen Sie die Spaltenbreite auf 0. Die *ColumnWidths*-Eigenschaft akzeptiert eine mit dem System-Argument-Trennzeichen getrennte Liste von Werten in Points.

Das Betätigen einer der Schaltflächen schreibt einen Wert in die Tag-Eigenschaft der UserForm und blendet diese dann aus; sie wird *nicht* aus dem Speicher entfernt. Der Vorteil dieser Vorgangsweise

ist, dass eine Verbindung zur Datenbank nicht bei jedem Aufruf der UserForm erfolgt. Zudem bleibt die Auswahl in der Liste bestehen, so dass der Anwender nicht immer wieder von vorne in der Liste beginnen muss.

In der Prozedur *ProduktListeEinblenden* ist ersichtlich, wie getestet wird, ob die UserForm schon geladen ist und wie diese Instanz weiter verwendet wird. Sonst wird eine neue Instanz aufgerufen.

Am Schluss dieser Prozedur wird der Wert der Tag-Eigenschaft geprüft und – falls »1« – die Prozedur *DatenInFormfelderSchreiben* aufgerufen. Sie überträgt die Artikelangaben aus der Liste in die entsprechenden Formularfelder und springt das Formularfeld in der Spalte »Anzahl« an, so dass der Anwender direkt weiterarbeiten kann.

**Listing 26.1** Produktliste einblenden und die Angaben des gewählten Eintrags in die Formularfelder einfügen

```
'Als globale Variable bleibt die UserForm im Speicher.
'Dies vermindert den Zugriff auf die Datenbank, da die Liste
'weniger oft geladen werden muss.

Dim frm As frmProduktListe

Sub ProduktListeEinblenden()
    Dim lFormZaehler As Long
    Dim rw As Word.row

    'Tabellenzeile, wovon die Prozedur aufgerufen wurde, festhalten.
    Set rw = Selection.Rows(1)
    'Schleife nur ausführen, wenn mindestens eine Form vorhanden ist.
    For lFormZaehler = 1 To UserForms.Count
        'Aber der erste Indexwert ist immer 0, also Zähler minus 1.
        If UserForms(lFormZaehler - 1).Name = "frmProduktListe" Then
            'Die Form weiterhin benutzen, wenn sie nur verborgen ist,
            Set frm = UserForms(lFormZaehler - 1)
            Exit For
        End If
    Next
    'sonst, eine neue Instanz aufrufen.
    If frm Is Nothing Then
        Set frm = New frmProduktListe
    End If
    frm.Show
    Select Case frm.Tag
        Case Is = "0"
        Case Is = "1"
            DatenInFormfelderSchreiben frm, rw
        Case Else
    End Select
End Sub

Sub DatenInFormfelderSchreiben(frm As frmProduktListe, rw As Word.row)
    Dim strProduktName As String
    Dim strProduktNr As String
    Dim strProduktPreis As String
    Dim lAuswahl As Long

    lAuswahl = frm.lstProdukte.ListIndex
```

**Listing 26.1** Produktliste einblenden und die Angaben des gewählten Eintrags in die Formularfelder einfügen (Fortsetzung)

```
'-1: kein Eintrag wurde ausgewählt; 0 = erster Eintrag der Liste
If lAuswahl >= 0 Then
    strProduktName = frm.lstProdukte.List(lAuswahl, 0)
    strProduktNr = frm.lstProdukte.List(lAuswahl, 1)
    strProduktPreis = frm.lstProdukte.List(lAuswahl, 2)

    rw.Cells(1).Range.FormFields(1).Result = strProduktNr
    rw.Cells(2).Range.FormFields(1).Result = strProduktName
    rw.Cells(4).Range.FormFields(1).Result = Format(strProduktPreis, "0.00")
    rw.Cells(3).Range.FormFields(1).Select
End If
End Sub

Sub AccessDatenHolen(ByRef RS As ADODB.Recordset, strSQL As String)
    Dim conn As ADODB.Connection

    Set conn = New ADODB.Connection
    conn.Open "Provider=Microsoft.Jet.OLEDB.4.0;" & _
        "Data Source=C:\WordBuch\CD-ROM\Datenbank\nordwind.mdb;"
    Set RS.ActiveConnection = conn
    RS.CursorLocation = adUseClient
    RS.CursorType = adOpenStatic
    RS.LockType = adLockOptimistic
    RS.Open Source:=strSQL
    'Da Daten nur gelesen und nicht geschrieben werden, können wir
    'die Verbindung kappen und Ressourcen sparen.
    Set RS.ActiveConnection = Nothing

    conn.Close
    Set conn = Nothing
    'Debug.Print RS.Fields.Count, RS.RecordCount
End Sub

Sub ListeFuellen(RS As ADODB.Recordset, lst As msforms.ListBox)
    Dim lDS_Counter As Long

    lDS_Counter = 0
    'Artikel aus der Datenbank in das Listfeld füllen.
    Do While Not RS.EOF
        lst.AddItem RS.Fields("Artikelname").Value
        lst.List(lDS_Counter, 1) = RS.Fields("Artikel-Nr").Value
        lst.List(lDS_Counter, 2) = RS.Fields("Einzelpreis").Value
        lDS_Counter = lDS_Counter + 1
        RS.MoveNext
    Loop
End Sub

'Code hinter der UserForm
Private Sub btnAbbrechen_Click()
    Me.Hide
    Me.Tag = "0"
End Sub

Private Sub btnUebernehmen_Click()
```



**Listing 26.1** Produktliste einblenden und die Angaben des gewählten Eintrags in die Formularfelder einfügen (Fortsetzung)

```

Me.Hide
Me.Tag = "1"
End Sub

Private Sub UserForm_Initialize()
    Dim RS As ADODB.Recordset
    Set RS = New ADODB.Recordset
    AccessDatenHolen RS, "SELECT Artikelname, [Artikel-Nr], Einzelpreis FROM Artikel" & _
        " ORDER BY Artikelname"
    ListeFuellen RS, lstProdukte
    RS.Close
    Set RS = Nothing
End Sub

```

Die Rechnungsposten werden mit der Symbolleiste *Rechnung schreiben* verwaltet. Hinter den Symbolschaltflächen stehen die Prozeduren *TabellenZeileLöschen* sowie *TabellenZeileEinfuegen*. Die erstere ist relativ einfach. Zunächst wird sichergestellt, dass sich die Markierung in der Tabelle befindet. Dann wird die aktuelle Tabellenzeile sowie der Indexwert der letzten Tabellenzeile festgehalten. Handelt es sich bei der aktuellen Zeile weder um die erste noch die letzte, wird der Dokumentschutz aufgehoben, die aktuelle Zeile gelöscht, der Dokumentschutz wieder hergestellt und das Formularfeld mit dem Gesamtbetrag aktualisiert.

Das Hinzufügen einer neuen Zeile ist ein etwas komplexer Vorgang. Nachdem sichergestellt ist, dass sich die Einfügemarke in der Tabelle befindet, wird ermittelt, in welcher Zeile sie steht. Befindet sie sich in der ersten Zeile, wird die neue direkt darunter, an zweiter Stelle, eingefügt. Befindet sie sich in der letzten Zeile, wird die neue unmittelbar davor eingefügt. Sonst wird die neue Zeile immer unter der aktuellen eingefügt.

#### HINWEIS

Diese Nachprüfung wurde der Vollständigkeit halber zwar im Beispiel eingebaut, aber in dieser Vorlage wird die Einfügemarke nie in der ersten oder letzten Zeile stehen, weil sich darin keine ungesperrten Formularfelder befinden.

Der Dokumentschutz wird aufgehoben, dann wird das Einfügen der neuen Zeile von der Prozedur *NeueZeileEinfuegen* vorgenommen. Sie sorgt auch dafür, dass der Inhalt der noch aktuellen Zeile in die neue übernommen wird. Dazu wird die *Range.FormattedText*-Eigenschaft auf jede Zelle der Zeile angewendet. Bitte beachten Sie, wie der Bereich zuerst um ein Zeichen verkleinert wird, so dass die Ende-der-Zelle-Marke nicht mit einbezogen wird.

Um sicherzustellen, dass jedes Formularfeld einen eindeutigen Namen hat, werden mit der Prozedur *NameHolen* alle am Ende der aktuellen Namen stehenden Ziffern entfernt. Diesem »Ur-Namen« wird der Inhalt einer eigens für diesen Zweck erstellten Dokumentvariablen hinzugefügt – eine laufende Zahl, die zu Beginn der Prozedur um eins erhöht wurde.

Um dem Formularfeld den Namen zuweisen zu können, muss es markiert werden. Es ist nicht möglich, einem Formularfeld über die *Name*-Eigenschaft einen Namen zu geben. Dies muss über das Dialogfeld *Formularfeld-Optionen* geschehen, das sich nur auf die gegenwärtige Markierung auswirkt.

Zum Schluss wird der Dokumentschutz wieder aktiviert, das neue Formularfeld in der zweiten Spalte (»Beschreibung«) angesprungen und die UserForm eingeblendet, so dass der Anwender sofort weiterarbeiten kann.

**Listing 26.2    Tabellenzeilen für die Rechnungsposten löschen und hinzufügen**

```

Sub TabellenZeileLöschen()
    Dim doc As Word.Document
    Dim rng As Word.Range
    Dim tbl As Word.Table
    Dim rw As Word.row
    Dim rwLetzte As Word.row

    Set doc = ActiveDocument
    Set rng = Selection.Range
    Set tbl = AktuelleTabelle(rng)
    If Not tbl Is Nothing Then
        Set rw = rng.Rows(1)
        Set rwLetzte = tbl.Rows.Last
        'Die zu löschende Zeile darf weder die erste noch die letzte sein
        If (rw.Index <> 1) And (rw.Index <> rwLetzte.Index) Then
            DokschutzAufheben doc
            rw.Delete
            Dokschuetzen doc
            doc.Bookmarks("txtTotal").Range.Fields.Update
        End If
    End If
End Sub

Sub TabellenZeileEinfuegen()
    Dim doc As Word.Document
    Dim rng As Word.Range
    Dim tbl As Word.Table
    Dim rw As Word.row
    Dim rwNeu As Word.row
    Dim lLetzteZeileIndex As Long

    Set doc = ActiveDocument
    Set rng = Selection.Range
    Set tbl = AktuelleTabelle(rng)
    If Not tbl Is Nothing Then
        DokschutzAufheben doc
        lLetzteZeileIndex = tbl.Rows.Last.Index
        Set rw = rng.Rows(1)
        Select Case rw.Index
            Case 1
                'Falls die aktuelle Zeile die erste ist, wird
                'die neue Zeile nach dieser eingefügt.
                'Die Formularfelder kommen auch aus dieser Zeile.
                Set rwNeu = NeueZeileEinfuegen(tbl.Rows(2), tbl.Rows(2))
            Case lLetzteZeileIndex
                'Falls die aktuelle Zeile die letzte ist, wird
                'die neue Zeile vor dieser eingefügt.
                'Die Formularfelder kommen aus der vorherigen Zeile.
                Set rwNeu = NeueZeileEinfuegen(tbl.Rows(lLetzteZeileIndex - 1), tbl.Rows.Last)
            Case Else
                'Sonst wird die neue Zeile nach der aktuellen eingefügt
                Set rwNeu = NeueZeileEinfuegen(rw, tbl.Rows(rw.Index + 1))
        End Select
        Dokschuetzen doc
        rwNeu.Range.FormFields(2).Select
    End If
End Sub

```

Listing 26.2 Tabellenzeilen für die Rechnungsposten löschen und hinzufügen (Fortsetzung)

```

    ProduktListeEinblenden
End If
End Sub

Function NeueZeileEinfuegen(rwOrig As Word.row, rwPos As Word.row) As Word.row
    Dim tbl As Word.Table
    Dim rwNeu As Word.row
    Dim rngOrig As Word.Range
    Dim rngNeu As Word.Range
    Dim fflDOrig As Word.FormField
    Dim fflDNeu As Word.FormField
    Dim lZellenZaehler As Long
    Dim docVar As Word.Variable

    Set tbl = rwOrig.Range.Tables(1)
    Set docVar = tbl.Parent.Variables("FormularfeldSatz")
    'Den Zeilenzaehler um eins erhöhen.
    docVar.Value = CStr(Trim(Val(tbl.Parent.Variables("FormularfeldSatz")) + 1))
    'Die neue Zeile hinzufügen.
    Set rwNeu = tbl.Rows.Add(rwPos)
    'Durch die Zellen beider Zeilen schleifen.
    For lZellenZaehler = 1 To rwOrig.Cells.Count
        Set rngNeu = rwNeu.Cells(lZellenZaehler).Range
        Set rngOrig = rwOrig.Cells(lZellenZaehler).Range
        'Die Bereiche dürfen die Ende-der-Zelle-Marke nicht enthalten
        rngNeu.MoveEnd Unit:=wdCharacter, Count:=-1
        rngOrig.MoveEnd Unit:=wdCharacter, Count:=-1

        'Den Inhalt in die neue Zeile "kopieren"
        rngNeu.FormattedText = rngOrig.FormattedText

        Set fflDOrig = rngOrig.FormFields(1)
        Set fflDNeu = rngNeu.FormFields(1)
        'Bei der Benennung muss das Formularfeld markiert sein ...
        fflDNeu.Select
        '... weil der Name nur über das Dialogfeld zugewiesen werden kann.
        With Dialogs(wdDialogFormFieldOptions)
            .Name = NameHolen(fflDOrig.Name) & docVar.Value
            .Execute
        End With
    Next
    Set NeueZeileEinfuegen = rwNeu
End Function

'Alle Ziffern vom Ende der Zeichenkette entfernen und das Resultat zurückgeben.
Function NameHolen(ByVal strName As String) As String
    Dim lZaehler As Long

    For lZaehler = Len(strName) To 1 Step -1
        If Not IsNumeric(Mid(strName, lZaehler, 1)) Then
            strName = Mid(strName, 1, lZaehler)
            Exit For
        End If
    Next
    NameHolen = strName

```

**Listing 26.2** Tabellenzeilen für die Rechnungsposten löschen und hinzufügen (*Fortsetzung*)

```
End Function

'Gibt ein Table-Objekt zurück, wenn die Markierung in einer Tabelle ist.
Function AktuelleTabelle(rng As Word.Range) As Word.Table
    Dim tbl As Word.Table

    If rng.Information(wdWithInTable) Then
        Set tbl = rng.Tables(1)
    End If
    Set AktuelleTabelle = tbl
End Function

Sub DokschutzAufheben(doc As Word.Document)
    If doc.ProtectionType <> wdNoProtection Then
        doc.Unprotect
    End If
End Sub

Sub Dokschuetzen(doc As Word.Document)
    doc.Protect Type:=wdAllowOnlyFormFields, NoReset:=True, Password="", _
        UseIrm:=False, EnforceStyleLock:=False
End Sub
```



Die Beispieldatei *Bsp26\_01.dot* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap26*.

## Zusammenfassung

In diesem Lösungsvorschlag für eine Bestellungsbestätigungsvorlage wurden aus dem Word-Objektmodell folgende Elemente besonders hervorgehoben:

- Formularfelder und ihre programmtechnische Anpassung
- Tabellen

Zudem wurde Folgendes vorgestellt:

- Eine ADO-Verbindung zu einer Access-Datenbank, um Informationen zu holen
- Eine UserForm mit Listen-Steuerelement

## Kapitel 27

# Attraktive Tischkarten im Nu

### In diesem Kapitel:

Das Seriendruck-Hauptdokument	850
Die WordArt-Objekt-Vorlage	851
Die Tischkarten erstellen	853
Zusammenfassung	857

In Kapitel 6 wurden die Grundlagen der Arbeit mit dem Word-Objektmodell vorgestellt. Diese Lösung veranschaulicht die folgenden Objekte, Eigenschaften und Methoden, mit Schwergewicht auf der Arbeit mit grafischen Objekten und WordArt.

---

**Im Blickpunkt:**

Seriendruck

WordArt

Word-Objekte

    Shapes

    Shapes.TextEffect

    MailMerge

    Table

---

Tischkarten werden oft für große Anlässe gebraucht. Heutzutage werden Adresslisten mühelos in einer Word-Tabelle, in einem Excel-Arbeitsblatt oder in einer Access-Datenbank angelegt. Mit der Seriendruck-Funktionalität von Word werden persönliche Einladungen und adressierte Umschläge schnell erstellt. Warum nicht auf ähnliche Art und Weise auch Tischkarten erstellen?

Für den Word-Anwender gibt es zwei Hindernisse:

- Gewöhnlicher Text kann nur um 90 oder 270 Grad gedreht werden, nicht jedoch um 180 Grad. WordArt-Objekte lassen sich zwar beliebig drehen, aber ...
- ... es ist nicht möglich, Seriendruckfelder in WordArt-Objekte zu integrieren.

Mit einer Automatisierungslösung können Seriendruck und WordArt vereint werden, um ansprechende Tischkarten schnell und problemlos zu reproduzieren.

Aus Sicht des Anwenders erfolgt die Herstellung in drei Schritten:

- Das Seriendruck-Hauptdokument mit einer Tabelle für die Tischkarten erstellen.
- Die gewünschte WordArt-Form festlegen.
- Ein Makro ausführen, das den Seriendruck mit den Daten zusammenführt und anschließend die Daten in WordArt-Objekte übernimmt.

## Das Seriendruck-Hauptdokument

Tischkarten werden ähnlich wie Etiketten gehandhabt und in einer Word-Tabelle angeordnet. Liegt keine vorgestanzte Papierform vor, kann die Größe der Tabellenzellen selbst definiert werden – die Gitternetzlinien zeigen die Begrenzungen an. Im Falle von Tischkarten werden zwei Tabellenzellen pro Karte gebraucht; der dazwischen liegende Zellenrand dient als Falzlinie. Falls Rahmenlinien auf den Karten unerwünscht sind, können, wie die Abbildung 27.1 zeigt, feine Striche an den linken und rechten Tabellenrändern als Falzlinien dienen.

Seriendruckfelder für den Karteninhalt werden wie üblich eingefügt. Falls ein Blatt Papier mehrere Karten enthält, wird eine *Next*-Feldfunktion am Anfang der ersten Zelle jeder neuen Karte gebraucht. Damit wird der Seriendruck gezwungen, den nächsten Datensatz auszugeben.

Abbildg. 27.1 Ein Seriendruck-Hauptdokument für Tischkarten

«Vorname»«Nachname»¶ ...¶ «Position»»
«Vorname»«Nachname»¶ ...¶ «Position»»
«Nächster Datensatz»«Vorname»«Nachname»¶ ...¶ «Position»»
«Vorname»«Nachname»¶ ...¶ «Position»»

## Die WordArt-Objekt-Vorlage

Der Anwender soll selbst entscheiden, wie die Information auf der Tischkarte auszusehen hat, indem er ein WordArt-Objekt entwirft und als Vorlage bestimmt.

Weil das Objekt in die Tabellenzellen passen muss, soll er die Tabelle in ein neues Dokument kopieren, alle Zellen außer einer löschen und darin das WordArt-Objekt erstellen, wie in Abbildung 27.2 dargestellt.

Dieses Dokument muss in dem gleichen Ordner wie das Seriendruck-Hauptdokument gespeichert werden, unter dem Namen »WordArt.doc«.

**HINWEIS** Die Anweisungen entsprechen dem hier verwendeten Code. Sie dürfen die Lösung selbstverständlich den eigenen Bedürfnissen anpassen.

Abbildg. 27.2 Ein WordArt-Objekt in einer einzelnen Tabellenzelle dient als Vorlage für die Tischkarten.



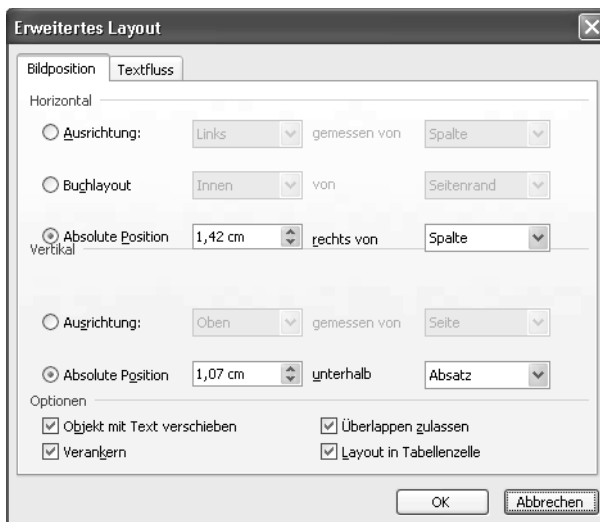
Bei der Positionierung des Objekts muss darauf geachtet werden, dass es

- mit Textfluss formatiert wird (also *nicht* »mit Text in Zeile«),
- senkrecht relativ zur Tabellenzelle und nicht zur Seite positioniert wird
- und verankert wird.

Die Abbildung 27.3 veranschaulicht die Positionierungseinstellungen.

**HINWEIS** Mehr über die Arbeit mit Grafiken (Shapes) steht im entsprechenden Abschnitt von Kapitel 6 beschrieben.

Abbildg. 27.3 Die Positionierungseinstellungen für das WordArt-Objekt





# Die Tischkarten erstellen

Jetzt ist nur noch das Makro aus Listing 27.1 auszuführen, um die Tischkarten zu erstellen (Abbildung 27.4). Wir schlagen vor, es einer Symbolschaltfläche in einer Vorlage zuzuweisen, die als Add-In geladen wird.

Die Prozedur *TischkartenErstellen* kontrolliert zuerst, ob das aktuelle Dokument ein Seriendruck-Hauptdokument ist; falls nicht, wird abgebrochen.

Andernfalls wird das Dokument gespeichert, der Dokumentpfad festgehalten (um später *WordArt.doc* öffnen zu können) und eine Dokumentvariable mit dem aktuellen Zeitpunkt angelegt. Erst dann wird der Seriendruck in ein neues Dokument ausgeführt.

Da MailMerge.Execute das Ergebnisdokument des Seriendrucks nicht zurückgibt, erfolgt eine Prüfung der Dokumentvariablen, die durch das Ergebnisdokument geerbt wird. Ist sie nicht vorhanden oder hat sie einen abweichenden Wert, wird der Vorgang abgebrochen. Ansonsten wird das Seriendruck-Hauptdokument geschlossen und das Erstellen der WordArt-Objekte im Ergebnisdokument fortgesetzt.

*WordArt.doc* wird unsichtbar geöffnet und das einzige grafische Objekt erfasst. Danach wird durch jede Tabellenzelle des Ergebnisdokuments geschleift. (Das Resultatdokument kann mehrere Tabellen enthalten, wenn der Seriendrucktyp nicht »Verzeichnis« ist.) Der Zelleninhalt wird in einer Variablen festgehalten, dann aus der Zelle gelöscht. Das WordArt-Objekt wird durch die Prozedur *WordArtErstellen* eingefügt.

Ihr werden die WordArt-Vorlage, der Textinhalt der Zelle, der Zellenbereich sowie die Angabe, ob das WordArt-Objekt gedreht werden soll, übergeben. Die Prozedur prüft jede Shape- und TextEffect-Eigenschaft der WordArt-Vorlage und weist sie dem neu erstellten Objekt zu.

Zum Schluss wird das WordArt-Vorlage-Dokument geschlossen und das Ergebnisdokument aktiviert.

## ACHTUNG

Die Ausführung kann wegen der vielen grafischen Formatierungen recht lange dauern.

Listing 27.1

Den Seriendruck zusammenführen und im Ergebnisdokument WordArt-Objekte einfügen

```
Sub TischkartenErstellen()
    Dim strPfad As String
    Dim docSeriendruck As Word.Document
    Dim docResultat As Word.Document
    Dim docWordArt As Word.Document
    Dim strDokVar As String
    Dim strDokVarInhalt As String

    If ActiveDocument.MailMerge.MainDocumentType = _
        wdNotAMergeDocument Then
        MsgBox "Das aktuelle Dokument muss ein Seriendruckdokument " & _
            "mit verbundener Datenquelle sein.", vbCritical + vbOKOnly
        Exit Sub
    End If

    Set docSeriendruck = ActiveDocument
    'Das Seriendruck-Hauptdokument vor der Zusammenführung speichern.
```

**Listing 27.1** Den Seriendruck zusammenführen und im Ergebnisdokument WordArt-Objekte einfügen *(Fortsetzung)*

```

docSeriendruck.Save
strPfad = docSeriendruck.Path
strDokVar = "WordArt"
'Die genaue Zeit als Dokumentvariable speichern, so dass nachgeprüft
'werden kann, ob am Schluss das Ergebnisdokument das aktuelle ist.
strDokVarInhalt = Format(Date, "yyyymmdd") & Format(Time, "hhmmss")
docSeriendruck.Variables(strDokVar) = strDokVarInhalt

docSeriendruck.MailMerge.Destination = wdSendToNewDocument
docSeriendruck.MailMerge.Execute
Set docResultat = ActiveDocument
'Sicherstellen, dass das aktuelle Dokument das Seriendruckergebnis ist.
'Es wird die sich im Hauptdokument befindenden Variablen 'erben'.
If docResultat.Variables(strDokVar) <> strDokVarInhalt Then
    MsgBox "Das aktuelle Dokument ist nicht das Seriendruckergebnis." &
        "Der Vorgang wird abgebrochen. Die Tischkarten konnten nicht erstellt werden.", _
        vbCritical + vbOKOnly
    Exit Sub
End If
docSeriendruck.Close SaveChanges:=wdDoNotSaveChanges

Dim shp As Word.Shape
Set docWordArt = Application.Documents.Open(Filename:=strPfad & "\WordArt.doc", _
    Visible:=False)
Set shp = docWordArt.Shapes(1)

Dim tbl As Word.Table
Dim cel As Word.Cell
Dim rng As Word.Range
Dim bDrehen As Boolean
Dim strText As String

For Each tbl In docResultat.Tables
    For Each cel In tbl.Range.Cells
        Set rng = cel.Range
        If cel.RowIndex Mod 2 = 0 Then
            bDrehen = False
        Else
            bDrehen = True
        End If
        strText = TrimZelle(rng.Text)
        rng.Delete
        WordArtErstellen shp, strText, rng, bDrehen
    Next
Next
docWordArt.Close SaveChange:=wdDoNotSaveChanges
docResultat.Activate
End Sub

```

Listing 27.1 Den Seriendruck zusammenführen und im Ergebnisdokument WordArt-Objekte einfügen (Fortsetzung)

```

Sub WordArtErstellen(shp As Word.Shape, strText As String, _
    rng As Word.Range, bDrehen As Boolean)

    Dim shpNeu As Word.Shape
    Dim te As Word.TextEffectFormat
    Dim lPresetEffect As Long

    Set te = shp.TextEffect
    '-2 bei Shapes bedeutet allgemein, dass keine Auswahl getroffen wurde.
    'Dieser Wert kann von einem anderen Objekt nicht übernommen werden.
    'Deshalb treffen wir eine Wahl.
    If te.PresetTextEffect = -2 Then
        lPresetEffect = 1
    Else
        lPresetEffect = te.PresetTextEffect
    End If
    'Das neue WordArt-Objekt mit den Eigenschaften der Vorlage formatieren.
    Set shpNeu = rng.Document.Shapes.AddTextEffect( _
        PresetTextEffect:=lPresetEffect, Text:=strText, _
        FontName:=te.FontName, FontSize:=te.FontSize, FontBold:=te.FontBold,
        FontItalic:=te.FontItalic, Left:=shp.Left, Top:=shp.Top, Anchor:=rng) _
    shpNeu.Height = shp.Height
    shpNeu.Width = shp.Width
    With shpNeu.TextEffect
        .PresetShape = te.PresetShape
        .KernedPairs = te.KernedPairs
        .NormalizedHeight = te.NormalizedHeight
        .RotatedChars = te.RotatedChars
        .Tracking = te.Tracking
    End With
    With shpNeu.Fill
        .BackColor = shp.Fill.BackColor
        .ForeColor = shp.Fill.ForeColor
        Select Case shp.Fill.GradientColorType
            Case msoGradientOneColor
                .OneColorGradient shp.Fill.GradientStyle, _
                    Variant:=shp.Fill.GradientVariant, _
                    Degree:=shp.Fill.GradientDegree
            Case msoGradientTwoColors
                .TwoColorGradient Style:=shp.Fill.GradientStyle, _
                    Variant:=shp.Fill.GradientVariant
            Case msoGradientPresetColors
                .PresetGradient Style:=shp.Fill.GradientStyle, _
                    Variant:=shp.Fill.GradientVariant, _
                    PresetGradientType:=shp.Fill.PresetGradientType
            Case msoGradientColorMixed
        End Select
        If shp.Fill.Pattern <> -2 Then
            .Patterned shp.Fill.Pattern
        End If
        If shp.Fill.TextureType = msoTexturePreset And _
            shp.Fill.PresetTexture <> -2 Then
            .PresetTextured shp.Fill.PresetTexture
        End If
        .Transparency = shp.Fill.Transparency
    End With

```

**Listing 27.1** Den Seriendruck zusammenführen und im Ergebnisdokument WordArt-Objekte einfügen (Fortsetzung)

```

With shpNeu.Line
    .BackColor = shp.Line.BackColor
    .DashStyle = shp.Line.DashStyle
    .ForeColor = shp.Line.ForeColor
    .Style = shp.Line.Style
    .Transparency = shp.Line.Transparency
    .Weight = shp.Line.Weight
End With
With shpNeu.Shadow
    .ForeColor = shp.Shadow.ForeColor
    .OffsetX = shp.Shadow.OffsetX
    .OffsetY = shp.Shadow.OffsetY
    .Transparency = shp.Shadow.Transparency
    If shp.Shadow.Type <> -2 Then
        .Type = shp.Shadow.Type
    End If
End With
With shpNeu.WrapFormat
    .AllowOverlap = shp.WrapFormat.AllowOverlap
    .DistanceBottom = shp.WrapFormat.DistanceBottom
    .DistanceLeft = shp.WrapFormat.DistanceLeft
    .DistanceRight = shp.WrapFormat.DistanceRight
    .DistanceTop = shp.WrapFormat.DistanceTop
    .Side = shp.WrapFormat.Side
    .Type = shp.WrapFormat.Type
End With
If bDrehen Then shpNeu.Rotation = 180
End Sub

Function TrimZelle(s As String) As String
    s = Left(s, Len(s) - 2)
    TrimZelle = s
End Function

```



Die Beispieldatei *Bsp27\_02.dot* mit dem VBA-Code sowie das Seriendruck-Hauptdokument *Bsp27\_01.doc* und die WordArt-Vorlage *WordArt.doc* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap27*.

Abbildg. 27.4 Die fertigen Tischkarten, bereit für den Ausdruck



## Zusammenfassung

Die Lösung in diesem Kapitel zeigt auf, wie mit dem Seriendruck und WordArt ein Satz dekorative Tischkarten erstellt werden können. Dabei werden aus dem Word-Objektmodell folgende Elemente veranschaulicht:

- Ansprechen des Seriendruckergebnisdokuments
- Das Festhalten und Festlegen von TextEffect- (WordArt-)Eigenschaften

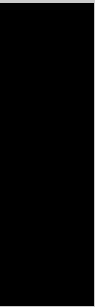


# Teil F

## XML und Smart-Technologien

### In diesem Teil:

Kapitel 28	Word 2003 und XML: Eine Einführung	861
Kapitel 29	Smarttags	911
Kapitel 30	Smart Documents	943





## Kapitel 28

# Word 2003 und XML: Eine Einführung

### In diesem Kapitel:

Was ist XML und wozu dient es?	862
Welche Aufgabe erfüllt XML in Word?	868
XML-Bestandteile	869
XML-Schema-Definitionen	875
XML-Daten manipulieren	885
XML in Word 2003	890
WordProcessingML außerhalb von Word generieren	908
Zusammenfassung	909

Word 2003 stellt einiges an neuer Funktionalität zur Verfügung, unter anderem Werkzeuge für die Arbeit mit XML. Diese sind hauptsächlich an Entwickler gerichtet, befinden sich aber teilweise auch in der Benutzerschnittstelle und können von jedem Anwender bedient werden.

XML dient in Word 2003 mehreren Zwecken, die in diesem und den folgenden Kapiteln dieses Teils beschrieben werden:

- als Dateiformat
- als Datensammelstelle innerhalb eines Word-Dokuments, die unabhängig von der Word-Dokumentstruktur verwaltet wird
- als Grundlage für die in Office XP eingeführte Smarttag-Funktionalität (siehe Kapitel 29)
- als Grundlage für die Smart Document-Funktionalität (siehe Kapitel 30)

Um diese Funktionalität einzusetzen, benötigen Sie Grundkenntnisse der XML-Technologie. Im vorliegenden Kapitel bieten wir eine kurze Übersicht dieses breit gefächerten Gebiets, mit Hinweisen auf weitere Informationsquellen, und stellen schließlich kurz die XML-Funktionalität in der Benutzerschnittstelle vor.

**HINWEIS**

Nicht alle Versionen von Word und Office unterstützen die Gesamtheit der XML-Funktionalität. Lediglich die eigenständige Version von Word 2003 (ohne Office) sowie Office Professional Edition 2003 und Office Professional Enterprise Edition 2003 umfassen alle Funktionen. Andere Ausgaben, wie Office Standard Edition 2003, verfügen lediglich über die Konvertierfilter, um in das XML-Dateiformat speichern oder XML-Dateien öffnen zu können, und unterstützen Smarttags.

## Was ist XML und wozu dient es?



»XML« steht für »Extensible Markup Language«. Dabei handelt es sich um eine Art Sprache, mit der Daten beschrieben und codiert werden. Eine standardisierte Methode der Datencodierung vereinfacht die Wiederverwendung und den Austausch von Informationen. XML hat in dieser Hinsicht einige Vorteile:

- XML ist ein offener Standard. Jeder darf ihn einsetzen und an seine eigenen Bedürfnisse anpassen.
- Der Einsatz von XML ist bereits weit verbreitet.
- XML ist ausführlich und öffentlich dokumentiert, mit klar ausgelegten Richtlinien. Außerdem existiert eine Vielzahl von Werkzeugen für die Arbeit mit XML, so dass damit codierte Daten problemlos wiederverwendet werden können.

Auch wenn Sie bislang mit XML keinen direkten Kontakt hatten, ist Ihnen vielleicht die Binsenwahrheit schon zu Ohren gekommen, XML trenne Inhalt (Daten) von dessen Präsentation. Sie stimmt, ist jedoch unvollständig. Wenn schon Daten mit XML festgehalten werden können, müsste es doch möglich sein, damit auch Informationen über deren Darstellung zu speichern. Und das ist tatsächlich der Fall, wie später aufgezeigt wird.

Beispiele von in XML codierten Daten sind in Listing 28.1 sowie in Listing 28.2 ersichtlich.

Listing 28.1 Inhalt von *BSP28\_01.htm*: Einige Kontinente mit ihren längsten Flüssen, in XML codiert

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Kontinente</title>
  </head>

  <body>
    <table border="2">
      <tr>
        <th>Kontinent</th>
        <th>Längster Strom</th>
      </tr>
      <tr>
        <td>Afrika</td>
        <td>Nil</td>
      </tr>
      <tr>
        <td>Asien</td>
        <td>Jangtse</td>
      </tr>
    </table>
  </body>
</html>
```

Listing 28.2 Inhalt von *BSP28\_02.xml*: Einige Kontinente mit ihren längsten Flüssen, auch in XML codiert

```
<?xml version="1.0" encoding="UTF-8"?>
<Kontinente>
  <Kontinent>
    <Name>Afrika</Name>
    <LängsterStrom>Nil</LängsterStrom>
  </Kontinent>
  <Kontinent>
    <Name>Asien</Name>
    <LängsterStrom>Jangtse</LängsterStrom>
  </Kontinent>
</Kontinente>
```



Die Beispieldateien *BSP28\_01.htm* sowie *BSP28\_02.xml* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap28*.

### XML-Datei manuell erstellen

Falls aus irgendeinem Grund die Beispieldateien auf der Buch-CD nicht zur Verfügung stehen, können diese manuell erstellt werden:

1. Erstellen Sie in Word oder im Windows-Editor ein leeres Dokument.
2. Geben Sie den Inhalt eines Listings genau so ein wie hier abgebildet (inklusive Großschreibung).

3. Führen Sie *Datei/Speichern* aus und wählen Sie einen Speicherort.
4. Legen Sie den Dateityp fest.
  - In Word wählen Sie in der Liste *Dateityp* den Eintrag *Nur Text (\*.txt)* aus.
  - Im Editor wählen Sie in der Liste *Dateityp* den Eintrag *Textdatei (\*.txt)* und in der Liste *Codierung* den Eintrag *UTF-8* aus.
5. Geben Sie den Dateinamen ein – beispielsweise "*Bsp28\_01.htm*" – inklusive der Anführungszeichen. Betätigen Sie die Schaltfläche *Speichern*.
6. Word müsste das Dialogfeld *Dateikonvertierung* einblenden. Stellen Sie sicher, dass die Option *Andere Codierung* aktiviert und in der Liste daneben der Eintrag *Unicode (UTF-8)* ausgewählt ist.

(Wenn Sie eine Datei in gewöhnlicher ANSI-Codierung speichern, werden Sonderzeichen wie beispielsweise Umlaute als ungültig bezeichnet.)

Auf die Zeichencodierung einer XML-Datei muss geachtet werden; sie muss mit der Deklaration am Dateianfang übereinstimmen. In den Beispielen dieses Buches wurde UTF-8 gebraucht, was bei XML den Standard darstellt. XML-Parser müssen nur UTF-8 sowie UTF-16 unterstützen (weitere Zeichencodierungen werden auf freiwilliger Basis erkannt).

Beim Speichern einer Datei als UTF-8 in Word oder im Editor wird zusätzlich ein BOM (»Byte Ordering Mark«) am Dateianfang eingefügt. Daran kann eine Anwendung erkennen, mit welcher Zeichencodierung die Datei erstellt wurde.

Leider fügt Word beim Speichern von XML-Dateien im Modus *Nur Daten speichern* kein BOM ein. Folglich muss unter Umständen beim Öffnen einer Word 2003-XML-Datei mit Unicode-Zeichen über das Dialogfeld *Dateikonvertierung* die Zeichencodierung explizit angegeben werden.

## XML-Vokabulare

Beim ersten Blick auf Listing 28.1 könnte der mit HTML Vertraute meinen, es handle sich, mit Ausnahme der ersten Zeile, um eine HTML-Datei. Und er hätte damit nicht Unrecht. Diese wurde mit einer auf XML basierenden Version von HTML geschrieben – nämlich XHTML.

Obwohl es angeblich die gleichen Informationen enthält, sieht Listing 28.2, mit Ausnahme der ersten Zeile sowie dem Gebrauch der Zeichen `<>`, entscheidend anders aus. Wenn beide angeblich XML sind, warum gleichen sie einander so wenig?

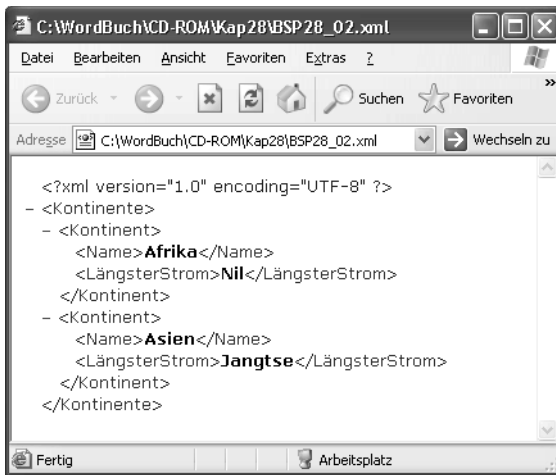
Eine der Stärken von XML ist, dass es den Bedürfnissen der Aufgabe angepasst werden kann, solange den Grundrichtlinien gefolgt wird. Dies ermöglicht die Erstellung eigener »Vokabulare«. Das Listing 28.1 wurde mit dem Vokabular XHTML, das Listing 28.2 dagegen mit einem anderen – nennen wir es »KML« (K für Kontinente) – geschrieben.

Beide sind also XML-Dateien, und XML-Werkzeuge können damit arbeiten. Unterschiedlich sind sie dennoch, wie die Interpretation der Vokabulare durch den Internet Explorer (siehe Abbildung 28.1 sowie Abbildung 28.2) veranschaulicht.

Abbildg. 28.1 Die XML-Datei *BSP28\_01.htm*, interpretiert vom Internet Explorer


Kontinent	Längster Strom
Afrika	Nil
Asien	Jangtse

Die XHTML-Datei wird genauso dargestellt, als hätte der Internet Explorer eine reine HTML-Datei vorgefunden; die Daten befinden sich in einer kleinen Tabelle mit Überschriftenzeile. Und damit kommen wir zurück auf die Binsenwahrheit über die Trennung von Daten und Präsentation. Offensichtlich umschreiben die Tags `<table>`, `<tr>`, `<td>` keine Informationen über Kontinente oder Flüsse, sondern enthalten Anweisungen, wie diese Daten darzustellen sind. Diese sind jedoch nicht Teil der Definition der XML-Sprache. Der Internet Explorer (wie andere Browser und Werkzeuge) ist programmiert, das XHTML-Vokabular zu erkennen und die Anweisungen zu interpretieren.

Abbildg. 28.2 Die XML-Datei *BSP28\_02.xml*, interpretiert vom Internet Explorer

Die »KML«-Datei in Abbildung 28.2 wird vom Internet Explorer anders behandelt. Er erkennt sofort, dass es sich um eine »echte« XML-Datei handelt und präsentiert deren Struktur. Schlüsselwörter, Tags und Daten werden in verschiedenen Farben dargestellt und Minus-Zeichen (–) ermöglichen das Zusammen- und Aufklappen der »Gliederungsebenen«. Er interpretiert, im Gegensatz zu XHTML, den Inhalt der Tags jedoch nicht.

Da liegen also zwei Dateien vor, beide in XML geschrieben, beide mit Daten über Kontinente und ihre Flüsse. Die XHTML-Datei, mit einem standardisierten Vokabular geschrieben, wird vom Browser als eine dreizeilige Tabelle dargestellt. Die »KLM«-Datei basiert auf keinem standardisierten

Vokabular und sieht im Internet Explorer nicht wesentlich anders aus als im Editor. Wozu dann die »KML«-Datei?

Genau diese strukturierte Darstellung der Daten, und nicht ihre Darstellung, ist Sinn der »KML«-Datei. Die XHTML-Datei sagt eigentlich nichts über die Struktur der Daten aus. Der Menschenverstand erkennt, dass Kontinente in einer Spalte aufgelistet sind, während die Namen von Flüssen sich in der anderen befinden. Diese Einteilung und Zugehörigkeit geht jedoch nicht klar aus den *Tags* der XHTML-Datei vor, wie im Fall der »KML«-Datei.

**HINWEIS** Mehr Informationen über die XML-Sprache finden Sie auf der deutschen Seite <http://edition-w3c.de/TR/2000/REC-xml-20001006/>.

## XML-Daten transformieren

Diese strukturierte Darstellung der Daten macht es einfach, sie zu transformieren, so dass der Internet Explorer den Dateiinhalt anders präsentiert. Zu diesem Zweck muss die »KML«-Datei leicht angepasst werden. Nach der ersten Zeile wird eine neue eingefügt, die ein »Stylesheet« spezifiziert (Listing 28.3).

**Listing 28.3** *BSP28\_03.xml*: XML-Kontinente-Datei, die auf eine XSLT-Transform verweist

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="BSP28_03.xsl"?>
<Kontinente>
  <Kontinent>
    <Name>Afrika</Name>
    <LängsterStrom>Nil</LängsterStrom>
  </Kontinent>
  <Kontinent>
    <Name>Asien</Name>
    <LängsterStrom>Jangtse</LängsterStrom>
  </Kontinent>
</Kontinente>
```

Da die Pfadangabe für href= relativ ist (nur der Dateiname), muss sich die XSLT-Datei im gleichen Ordner mit der XML-Datei befinden. Sie enthält die Anweisungen in Listing 28.4. Wird *BSP28\_03.xml* im Internet Explorer geöffnet, muss es ähnlich aussehen wie die Abbildung 28.1.

**Listing 28.4** *BSP28\_03.xsl*: XSL-Transform für die Kontinente-Datei

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <html>
      <head>
        <title>Kontinente</title>
      </head>
      <body>
        <table border="2">
          <tr>
            <th>Kontinent</th>
```

Listing 28.4 BSP28\_03.xsl: XSL-Transform für die Kontinente-Datei (Fortsetzung)

```

        <th>Längster Strom</th>
      </tr>
      <xsl:for-each select="Kontinente/Kontinent" >
        <tr>
          <td><xsl:value-of select="Name" /></td>
          <td><xsl:value-of select="LängsterStrom" /></td>
        </tr>
      </xsl:for-each>
    </table>
  </body>
</html>
</xsl:template>
</xsl:stylesheet>

```



Die Beispieldateien *BSP28\_03.xml* sowie die Transformationsdatei *BSP28\_03.xsl* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap28*.

Die neu hinzugefügte Zeile der »KML«-Datei veranlasst den Internet Explorer, eine XSLT (»Extensible Stylesheet Language Transformation«) heranzuziehen. Diese Transformationsdatei enthält Anweisungen, die den Inhalt der XML-Tags in HTML-Tags einfügen, so dass der Internet Explorer die Daten für jeden Kontinent der »KML«-Datei in einer eigenen Tabellenzeile anzeigt.

XSLT ermöglicht es, die strukturierten Daten der »KML«-Datei zur Verfügung zu stellen, so dass sie beliebig benutzt werden können. In diesem Fall werden sie als HTML-Tabelle dargestellt. XSLT ist ein äußerst mächtiges Mitglied der XML-Familie. Es kann Daten in etliche Formate umgestalten (auch XML) und sie während des Prozesses filtern, sortieren und sogar Berechnungen damit ausführen.

#### HINWEIS

Mehr Informationen über XSLT finden Sie auf der deutschen Seite <http://xml.klute-thiemann.de/w3c-de/REC-xslt-20020318/>.

Dies war ein vereinfachtes Beispiel für die vielfältige Wiederverwendung von Daten. Den Bedarf, Informationen auf mehrere Weisen wieder zu verwenden, kennen die meisten Organisationen, sei es, um sie in gedruckten Berichten sowie auf einer Webseite zu publizieren oder um sie für verschiedene Lesergruppen anders zu gestalten.

Neu sind diese Aufgaben nicht. Sie mussten jedoch meistens mit der Konvertierung der Daten aus einem geschlossenen Datenformat in ein anderes verwirklicht werden, was sich nicht immer einfach gestaltet. Der Bedarf eines offenen, standardisierten, strukturierten Datenformats, das einfach, schnell und zuverlässig anwendbar ist, ist Iso vorhanden. XML wurde hierfür entwickelt.

#### PROFITIPP

Ihnen ist vielleicht aufgefallen, dass die erste Beispieldatei die Endung *\*.htm* statt *\*.xml* hat. In Wirklichkeit versteht der Internet Explorer XHTML noch nicht, versucht aber, jede *\*.htm*-Datei zu interpretieren, auch wenn die Tags den Richtlinien nicht ganz entsprechen.

Dieser »Missstand« kann mit einer kleinen Änderung behoben werden:

- Fügen Sie die folgende Codezeile in die Datei *BSP28\_01.htm* ein:

```
<?xml-stylesheet type="text/xsl" href="BSP28_04.xsl"?>
```

## PROFITIPP

- Speichern Sie die Datei unter den Namen *BSP28\_04.xml*.
- Geben Sie den Code aus Listing 28.5 in eine neue Datei ein und speichern Sie diese unter dem Namen *BSP28\_04.xsl*.
- Öffnen Sie *BSP28\_04.xml* im Internet Explorer. Das Resultat sollte wie in Abbildung 28.1 aussehen.

Der XSLT-Code transformiert XHTML in gewöhnliches, korrektes HTML.

**Listing 28.5** *BSP28\_04.xsl*: XSLT, um XHTML in korrektes HTML zu transformieren

```
<stylesheet version="1.0"
  xmlns="http://www.w3.org/1999/XSL/Transform">
  <template match="/">
    <copy-of select="."/>
  </template>
</stylesheet>
```

## Welche Aufgabe erfüllt XML in Word?

Die vorangegangene kurze Erläuterung zum Nutzen und Zweck von XML sagt noch nichts über die Möglichkeiten von XML im Kontext von Word aus. Dieser Abschnitt bietet Ihnen einen kurzen Überblick der XML-Funktionalität in Word, basierend auf den gerade vorgestellten Konzepten.

Das einfache Beispiel im Abschnitt »XML-Vokabulare« weiter vorne in diesem Kapitel veranschaulicht den Effekt eines Vokabulars auf die Strukturen einer XML-Datei und die Interpretation dieser durch den Internet Explorer. Das eine Vokabular umschreibt die Präsentation der Daten, während das andere primär ihre Strukturierung festlegt.

Word 2003 unterstützt ebenfalls mehr als nur ein Vokabular:

- Word besitzt sein eigenes Vokabular – WordProcessingML (auch WordML genannt) –, das die Präsentation des Dokumentinhalts umschreibt. Die gesamte Word-Funktionalität, inklusive des Layouts und der Formatierung, ist darin definiert. Damit kann jedes Word-Dokument ohne Informationsverlust als XML gespeichert und wieder geöffnet werden. (Jede Ausgabe von Word 2003 unterstützt diese Konvertierung.)
- Es ist zudem in einigen Word-Versionen möglich (siehe die Einleitung zu diesem Kapitel), eigene Vokabulare in ein Dokument einzubinden. Dies ermöglicht dem Entwickler einen Zugang zu den darin enthaltenen Daten, ohne sich mit dem übrigen Inhalt des Dokuments befassen zu müssen.

Um die Verwendung eigener Vokabulare zu unterstützen, wurden weitere Funktionalitäten eingeführt:

- Eine XML-Schemabibliothek. Das Einbinden eines eigenen Vokabulars ist nur möglich, wenn der Schemabibliothek des einzelnen Benutzers ein passendes Schema hinzugefügt wurde. Liegt ein entsprechendes Schema vor, kann es Dokumenten sowie Vorlagen beliebig angefügt werden (diese wird im Abschnitt »Die Word-Schemabibliothek« in diesem Kapitel vorgestellt).
- Unterstützung von XML-Tags in den verschiedenen Dokumentformaten. Die XML-Tags eines eigenen Vokabulars und die darin enthaltenen Daten werden mitgespeichert, egal ob als \*.doc, \*.xml oder im Webseiten-Format.



- Beim Speichern im XML-Format kann wahlweise das ganze Dokument (als WordProcessingML) gespeichert werden oder nur die Elemente des eigenen Vokabulars (*Nur Daten speichern*).
- Neue Aufgabenbereiche. Die Aufgabenbereiche *XML-Struktur* sowie *XML-Dokument* unterstützen die Arbeit mit dem von einem Schema zur Verfügung gestellten Vokabular. Zusätzlich gibt es den Aufgabenbereich *Dokumentaktionen* als Benutzerschnittstelle einer Smart Document-Lösung (Smart Documents werden in Kapitel 30 vorgestellt).
- Die *IncludeText*-Funktion wurde mit zusätzlichen Schaltern ergänzt, die das Einfügen von XML-Daten in WordProcessingML-Vokabular unterstützen.
- Das »XML Expansion Pack« wurde als eine neue Art Add-In zur Unterstützung der Smart Document-Funktionalität eingeführt.
- Das Word-Objektmodell erhielt neue, XML-bezogene Objekte, Methoden und Eigenschaften.

**HINWEIS** Microsoft stellt umfangreiche Dokumentationen (alles in englischer Sprache) auf der MSDN-Website bereit. Unter anderem finden Sie dort:

- Microsoft Office 2003 Word XML SDK ([http://msdn.microsoft.com/library/default.asp?url=/library/en-us/wordxmldk/html/cdkIntroDefaultPage\\_HV01113687.asp](http://msdn.microsoft.com/library/default.asp?url=/library/en-us/wordxmldk/html/cdkIntroDefaultPage_HV01113687.asp))
- Microsoft Office 2003 XML Reference Schemas (<http://www.microsoft.com/downloads/details.aspx?familyid=fe118952-3547-420a-a412-00a2662442d9&displaylang=en>)
- Microsoft Office 2003 XML Toolbox (ein .NET Framework 1.1-Add-In, das über eine Symbolleiste nützliche Werkzeuge für die Arbeit mit XML zur Verfügung stellt) ([http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnofftalk/html/ODC\\_office01012004.asp](http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnofftalk/html/ODC_office01012004.asp))

## XML-Bestandteile

Die XML-Sprache allein genügt nicht, um nützliche Lösungen bereit zu stellen. Im Abschnitt »XML-Daten transformieren« in diesem Kapitel sahen wir beispielsweise, dass es einer Transformation bedarf, um die Daten auf benutzerfreundliche Art darzustellen. Tabelle 28.1 listet die Hauptmitglieder der XML-Familie auf, die zum erfolgreichen Einsatz von XML als Datenverwaltungsstruktur beitragen.

In diesem Abschnitt werden wir die Grundlagen der verschiedenen Bestandteile der XML-Familie und ihre Anwendung kurz vorstellen.

**Tabelle 28.1** Die Mitglieder der XML-Familie

Standard	Beschreibung
XML	Die »Extensible Markup Language«
XML-Schema	Eine Sprache, die die Strukturen und Datentypen eines XML-Vokabulars und den Aufbau eines XML-Dokuments definiert
»Namespaces« (Namensräume)	Ein Konstrukt, um Namenskonflikte in XML-Dateien zu vermeiden, die auf mehreren Vokabularen (Schemas) basieren. Ermöglicht die Bezeichnung des Herkunftsschemas für jedes Element.

**Tabelle 28.1** Die Mitglieder der XML-Familie (Fortsetzung)

Standard	Beschreibung
DOM	Das »Document Object Model«. Beschreibt XML-Dokumentstrukturen aus der objektorientierten Sicht. Wird von einer Programmiersprache wie Visual Basic oder VB.NET verwendet, um ein XML-Dokument zu lesen oder schreiben.
XSLT	»Extensible Stylesheet Language: Transformations«. Die Formatierungsfähigkeiten erinnern an CSS, diese Sprache kann aber viel mehr. Damit können Daten gefiltert und anders zusammengestellt werden, um viele Dokumenttypen zu generieren.
XPath	Eine Sprache, die das Ansprechen von Elementen und Attributen einer Datei ermöglicht. Diese können entweder anhand des Namens oder des Inhalts identifiziert werden.

**HINWEIS** XML-Schema und XSLT sind ihrerseits auch XML-Vokabulare, wie aus Listing 28.4 hervorgeht.

## XML-Parser

Um etwas mit dem Inhalt eines XML-Dokuments anfangen zu können, wird Software benötigt, die alle XML-Bestandteile und die XML-Richtlinien kennt. Diese Art Software wird als »XML-Parser« bezeichnet. Der Internet Explorer ist kein XML-Parser, zieht jedoch einen hinzu, wenn er einem XML-Dokument begegnet: den Microsoft MSXML-Parser.

## XML-Elemente, -Tags, -Inhalt und -Attribute

**XML-Elemente** sind die Hauptbestandteile jedes XML-Dokuments; sie stellen Informationseinheiten dar. Elemente können »einfach« sein und nur Zeichen (Daten) enthalten. Oder sie können »komplex« aufgebaut werden und weitere Elemente umfassen. Beispiel eines komplexen Elements wäre ein »Adresse«-Element, das die weiteren Elemente »Strasse«, »PLZ« und »Ort« einschließt. Gemischte Elemente, die sowohl Daten als auch Elemente beinhalten, sind ebenfalls möglich.

**Tags** bezeichnen den Anfangs- sowie den Endpunkt eines Elements und stehen in spitzen Klammern < >. Zwischen den spitzen Klammern befindet sich der Elementname. Das End-Tag eines Elements erkennt man an dem Schrägstrich vor dem Elementnamen: <Adresse>Elementinhalt</Adresse>. Leere Elemente (ohne Dateninhalt) sind auch möglich; sie haben nur ein Tag, mit Schrägstrich am Ende: <Adresse/>.

Ein komplexer Inhalt könnte demzufolge so aussehen:

```
<Adresse><Strasse>Hauptstrasse 1</Strasse><PLZ>10000</PLZ>
<Ort>Irgendeine Stadt</Ort></Adresse>
```

Die Information eines Elements kann durch *Attribute* qualifiziert werden. Beispielsweise könnte festgehalten werden, ob es sich um eine Geschäfts- oder Privatadresse handelt:

```
<Adresse Art="Geschäft" >Elementinhalt</Adresse>
```

Attribute befinden sich im Anfangs-Tag (nach dem Elementnamen) und werden durch Kommas getrennt. Ein Attribut besteht aus drei Teilen: der Bezeichnung, gefolgt von einem Gleichheitszeichen (=) sowie dem Inhalt in 'einfachen' oder "doppelten" Anführungszeichen.

Eine gültige Element- oder Attribut-Bezeichnung muss

- mit einem Buchstaben, Unterstrich oder Doppelpunkt anfangen,
- gefolgt von weiteren Buchstaben, Ziffern, Strichen, Unterstrichen, Doppelpunkten oder Punkten.

#### HINWEIS

Es wird davon abgeraten, Doppelpunkte in Bezeichnungen zu gebrauchen oder solche mit der Zeichenfolge »XML« (egal ob groß oder klein geschrieben) zu beginnen.

## Fehlende sowie mehrfach vorkommende Werte

Sind Sie gewohnt, mit Tabellen für die Datenverwaltung oder als Programmierer mit Strukturen zu arbeiten, erwarten Sie, dass jede Dateneinheit (Datensatz) die gleichen Elemente (Felder) umfasst. Wird ihnen kein Inhalt zugewiesen, haben sie einen standardmäßigen Wert wie Null, 0 oder eine »leere« Zeichenkette.

Ein XML-Vokabular kann, durch eine entsprechende Schema-Definition, diese Datenstruktur erzwingen. Die Sprache erlaubt jedoch fehlende oder gar mehrfach vorkommende Elemente. Die Auflistung von Kontinenten und Flüssen dürfte beispielsweise Kontinente ohne Flüsse oder mehrere Flüsse für einen Kontinent enthalten, wie das Listing 28.6 veranschaulicht.

Listing 28.6

XML- Datei mit fehlenden sowie mehrfachen Datenelementen

```
<?xml version="1.0" encoding="UTF-8"?>
<Kontinente>
  <Kontinent>
    <Name>Afrika</Name>
  </Kontinent>
  <Kontinent>
    <Name>Asien</Name>
    <LängsterStrom>Jangtse</LängsterStrom>
  </Kontinent>
  <Kontinent>
    <Name>Nordamerika</Name>
    <LängsterStrom>Mississippi</LängsterStrom>
    <LängsterStrom>St. Lawrence</LängsterStrom>
  </Kontinent>
</Kontinente>
```

## Elemente oder Attribute?

Eine oft gestellte Frage ist, wann Attribute und wann Elemente in einer XML-Struktur gebraucht werden; warum die Art der Adresse nicht wie folgt festgehalten wird:

```
<Adresse>
  <Art>Geschäft</Art>
</Adresse>
```

Darüber wird viel diskutiert; feste Regeln gibt es nicht. Wir stützen unsere Entscheidungen auf die folgenden Überlegungen:

- Informationen, die dem Benutzer normalerweise nicht vorgestellt werden, werden in Attributen hinterlegt. In einem Word-Dokument beispielsweise gibt es keine Schnittstelle, um den Inhalt von Attributen anzuzeigen oder zu bearbeiten. Nur das Kontextmenü bietet Zugang zu diesen Informationen.
- Im Gegensatz zu Elementen kann die Reihenfolge von Attributen innerhalb eines Elements weder durch eine Dokumenttyp-Deklaration (DTD, siehe den folgenden Abschnitt) noch ein Schema festgelegt werden.
- Attribute können, anders als Elemente, keine untergeordneten Attribute haben.

### HINWEIS

Erstellen Sie eine XML-Datei, die auf einem existierenden Schema basiert, wird diese Entscheidung schon gefallen sein, und Sie haben sich daran zu halten.

## XML-Dokumente und Deklarationen

XML-Dokumente sind in drei Teilen aufgebaut:

- die XML-Deklaration (nicht immer erforderlich)
- eine Dokumenttyp-Deklaration (nicht immer erforderlich)
- der Dokumentinhalt (auch Dokumentinstanz genannt)

Die minimal mögliche Deklaration lautet:

```
<?xml version="1.0" encoding="UTF-8" ?>
```

Die folgenden Anforderungen werden gestellt:

- Wenn sie vorhanden ist, muss sie die erste Zeile des Dokuments sein, ohne vorhergehende Zeichen jeglicher Art (also auch keine Leerzeichen).
- Die Einführungs- sowie Schlusszeichen <? und ?> sind erforderlich.
- Auf Groß- und Kleinschreibung wird geachtet; demzufolge ist <?XML Version="1.0" ?> oder jede andere Variation *ungültig*.
- Die Versionsnummer muss von Anführungszeichen umgeben sein.

Die Deklaration kann auch, wie Sie bereits im Abschnitt »XML-Vokabulare« in diesem Kapitel gesehen haben, die Zeichencodierung festlegen.

Ein weiteres erlaubtes Attribut der Deklaration ist standalone. Wird es auf no gesetzt, ist das XML-Dokument mit einer Dokumenttyp-Definition (DTD, eine Alternative zu einem Schema) verbunden. Da Word 2003 nur Schemas und keine DTD unterstützt, werden wir darauf nicht näher eingehen.

Dieses Kapitel enthält ein Beispiel einer DTD-Deklaration:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

Die Verknüpfung eines XML-Dokuments mit einem Schema wird anders vorgenommen. Darauf kommen wir im Abschnitt »XML-Schema-Definitionen« zurück.

## XML-Dokumentinstanz, Markup und Inhalt

Die Dokumentinstanz besteht aus einem einzigen Element – dem Wurzelement – und dessen Inhalt. Der Inhalt darf Markups sowie Zeichendaten umfassen. Markup ist ein breiter Begriff; darunter fallen beispielsweise

- Start- und End- sowie Leere-Elemente-Tags
- Entity- und Zeichen-Referenzen wie &amp; (für &) und &quot; (für "). Fünf sind vordefiniert: &amp; (&), &apos; ('), &gt; (>) &lt; (<) sowie &quot; (").
- Kommentare wie <!-- Kommentar -->
- CDATA-Abschnitt-Tags wie <![CDATA[die Daten]]>
- Dokumenttyp-Deklarationen (wie oben beschrieben)
- Verarbeitungsanweisungen
- XML-Deklarationen
- Text-Deklarationen
- Leerzeichen außerhalb des Wurzelements



Verarbeitungsanweisungen werden vom XML-Parser an die Verbraucheranwendung weitergegeben. Beispielsweise fügt Word 2003 eine Verarbeitungsanweisung in eine WordProcessingML-Datei ein, die den Internet Explorer veranlasst, die Datei möglichst in Word zu öffnen und den Windows-Explorer veranlasst, die Datei mit einem anderen Symbol zu versehen.

Normalerweise werden »überzählige« Leerzeichen vom Parser entfernt und XML-Steuerzeichen wie < müssen codiert werden. Dies kann es erschweren, gewisse Daten (z.B. Programmzeilen) in XML zu erfassen. Text innerhalb eines CDATA-Abschnitts wird vom XML-Parser unverändert an die Anwendung weitergeleitet:

```
<![CDATA[
If x < 0 Then
  MsgBox "x < 0"
Else
  If y < 0 Then
    MsgBox "y < 0"
  End If
End If
]]>
```

## Wohlgeformte und gültige XML-Dokumente

Wer jemals HTML-Dokumente geschrieben hat, hat bestimmt festgestellt, dass Browser die HTML-Syntax großzügig interpretieren. Der Internet Explorer beispielsweise besteht nicht darauf, dass ein `<html>`-Start-Tag am Dokumentanfang steht. Und oft ist es nicht notwendig, eine Anweisung mit einem End-Tag abzuschließen.

Die XML-Philosophie ist in dieser Hinsicht weniger nachsichtig, was die Eindeutigkeit erhöht und die Produktionskosten der Software, die XML bearbeitet, entsprechend verringert. Im Gegensatz zur HTML-Philosophie, die offensichtlich besagt: »Wenn ein Syntaxfehler vorkommt, mache weiter und zeige möglichst ein brauchbares Resultat an.«, arbeiten XML-Parsers nach dem Grundsatz: »Kommt ein Syntaxfehler vor, abbrechen und eine Fehlermeldung zurückgeben.«

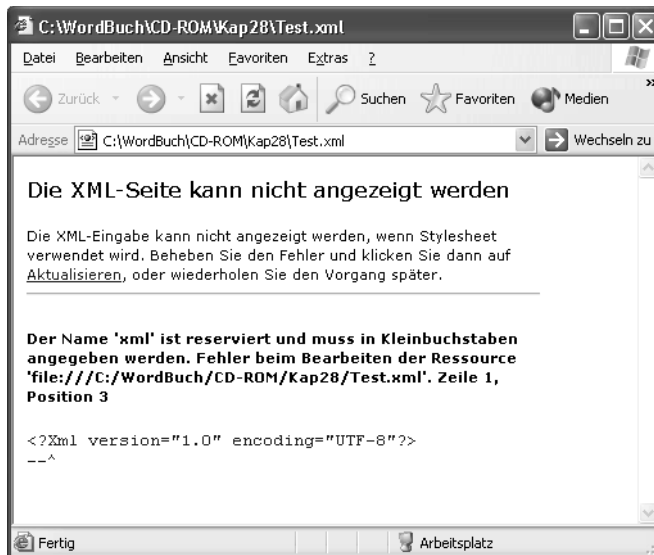
Ein Dokument, das allen Regeln der XML-Syntax nachkommt, wird als *wohlgeformt* betrachtet. Ist ein Dokument nicht wohlgeformt, ist es auch kein XML-Dokument und wird von einem XML-Parser abgelehnt, wie die Abbildung 28.3 veranschaulicht.

Einige Dinge, die die Wohlgeformtheit eines XML-Dokuments verlangt, aber in HTML nicht immer eingehalten werden müssen:

- Die Werte für Attribute müssen von Anführungszeichen umschlossen sein.
- Start- und End-Tags müssen paarweise vorhanden sein (außer es handelt sich um ein Leeres-Element-Tag).
- Nur ein Wurzelement ist erlaubt.
- Elemente müssen korrekt verschachtelt werden; überschneidende Elemente sind nicht erlaubt.
- Bei Element- und Attributnamen wird auf Groß-/Kleinschreibung geachtet.
- Gewisse Schlüsselwörter, wie DOCTYPE, müssen groß geschrieben werden.

Abbildg. 28.3

Das Dokument ist nicht wohlgeformt, weil sich ein Syntaxfehler in der XML-Deklaration befindet.



Ein wohlgeformtes Dokument ist nicht unbedingt ein *gültiges* XML-Dokument. Ein gültiges XML-Dokument muss sowohl wohlgeformt sein als auch die in einem DTD oder Schema ausgelegten Regeln befolgen. Die XML-Datei in Listing 28.1 ist wohlgeformt und gültig, weil sie die Regeln des XHTML DTD befolgt. Die XML-Datei in Listing 28.2 hingegen ist lediglich wohlgeformt, aber nicht gültig, weil sie nicht einmal eine DTD oder Schema-Deklaration enthält.

## XML-Schema-Definitionen



XML Schema Definitionen (XSD) sind eine Möglichkeit, Elemente und Attribute einer XML-Datei vorzuschreiben. Obwohl es andere Methoden gibt, wie DTD, sind Schemas die einzigen, mit denen Word 2003 arbeitet. Ein Schema listet nicht nur die erlaubten Elemente und Attribute auf, es bestimmt ihren Datentyp, ihren Platz in der Dokument-Hierarchie, ihre Reihenfolge und, ob sie erforderlich oder optional sind.

Das XML-Schema Standard ist dermaßen komplex, dass es von drei separaten 3WC-Recommendationen beschrieben ist. Es liegt auf der Hand, dass wir hier nicht alle Einzelheiten vorstellen können. Mehr Informationen finden Sie in folgender Dokumentation:

- XML Schema Teil 0: Einführung. Eine gut lesbare Beschreibung der Möglichkeiten von XML-Schema (<http://www.edition-w3c.de/TR/2001/REC-xmlschema-0-20010502/>).
- XML Schema Teil 1: Strukturen. Spezifiziert die XML-Schema-Definitionssprache, mit der die Struktur von XML 1.0-Dokumenten beschrieben und Bedingungen an deren Inhalt formuliert werden können (<http://www.edition-w3c.de/TR/2001/REC-xmlschema-1-20010502/>).
- XML Schema Teil 2: Datentypen. Hier werden Möglichkeiten beschrieben, um Datentypen zu definieren, die sowohl in XML-Schemata als auch in anderen XML-Spezifikationen eingesetzt werden können (<http://www.edition-w3c.de/TR/2001/REC-xmlschema-2-20010502/>).

Ein Beispiel sehen Sie in Listing 28.7, das das Vokabular der »KML«-Datei aus Abschnitt »XML-Vokabulare« umschreibt. Es veranschaulicht viele Grundlagen der XSD.

**Listing 28.7** *BSP28\_05.xsd*: XML-Schema für die »KML«-Datei (Listing 28.2). Bitte beachten Sie, dass auch dieses ein XML-Dokument ist.

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="Kontinente">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="Kontinent" maxOccurs="unbounded" />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="Kontinent">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="Name" type="OrtsNameTyp" minOccurs="1" />
        <xsd:element name="LängsterStrom" type="OrtsNameTyp" minOccurs="0" />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:simpleType name="OrtsNameTyp">
    <xsd:restriction base="xsd:string">
```

**Listing 28.7** *BSP28\_05.xsd*: XML-Schema für die »KML«-Datei (Listing 28.2). Bitte beachten Sie, dass auch dieses ein XML-Dokument ist. (Fortsetzung)

```
<xsd:maxLength value="20" />
</xsd:restriction>
</xsd:simpleType>
</xsd:schema>
```



Die Beispieldatei *BSP28\_05.xsd* finden Sie auf der CD-ROM zum Buch im Ordner `\Beispiele\Kap28`.

Ein Element wird durch das Tag `<xsd:element>` definiert. Das Attribut `name` ist erforderlich. Die Elemente `Kontinente` und `Kontinent` sind beide als komplexe Typen deklariert: `<xsd:complexType>`; demzufolge umfassen sie weitere Elemente. In beiden Fällen ist die nächste Anweisung `<xsd:sequence>`; was bedeutet, dass jene Elemente in der angegebenen Reihenfolge erscheinen müssen.

#### **HINWEIS**

Dieses Schema definiert zwei Elemente auf der obersten Ebene: `Kontinente` sowie `Kontinent`. Bekanntlich darf ein XML-Dokument nur ein Wurzelement haben, dieses Schema bewahrt also nicht vor diesem möglichen Fehler.

Als Nächstes werden die verschachtelten Elemente deklariert. `Kontinente` darf nur Elemente des Typs `Kontinent` enthalten, und zwar eine unbegrenzte Menge (bestimmt durch das Attribut `maxOccurs="unbounded"`) davon. Das Element `Kontinent` seinerseits darf zwei Elemente enthalten: `Name` sowie `LängsterStrom`. Mindestens einmal muss das Element `Name` vorkommen (bestimmt durch das Attribut `minOccurs="1"`); während `LängsterStrom` fehlen darf (bestimmt durch das Attribut `minOccurs="0"`).

Beide dieser Elemente sind vom Typ `OrtsNameTyp`. Es handelt sich hier um einen benutzerdefinierten Typ, der seinerseits ein einfacher Typ ist (darf nur Zeichendaten enthalten). `<xsd:restriction base="xsd:string">` bedeutet, dass der Inhalt eine Zeichenkette mit einer maximalen Länge von 20 Zeichen (`<xsd:maxLength value="20" />`) sein muss.

#### **HINWEIS**

Die Attribute `minOccurs` und `maxOccurs` sind nicht erforderlich; der standardmäßige Wert beträgt in beiden Fällen »1« (das Element muss einmal vorkommen, und nicht mehr als einmal).

XSD enthält viele einfache Datentypen für Zeichenketten, numerische Werte sowie Datenangaben. Eine Auflistung finden Sie in »XML Schema Teil 0: Einführung«. Diese können auch, wie hier veranschaulicht, in bestimmten Kombinationen mit Begrenzungen zusammengestellt werden, um benutzerdefinierte Typen zu definieren. Die Parameter, die begrenzt werden können, werden »Facetten« genannt. Beispiele dafür sind:

- die Anzahl der Zeichen (`minLength` oder `maxLength`)
- ein mit einem »Regular Expression« definierter Mustervergleich
- eine Enumeration (Auflistung gültiger Werte)
- ein numerischer Bereich

Sollen verschachtelte Elemente in einer beliebigen Reihenfolge in der XML-Datei erscheinen dürfen, ersetzen wir das Element `<xsd:sequence>` durch `<xsd:all>`.



Dieses Beispiel veranschaulicht weiter das Attribut `ref`:

```
<xsd:element ref="Kontinent" maxOccurs="unbounded" />
```

XML-Schemas können beliebig komplex sein. Elemente können verschachtelt definiert werden, wie Listing 28.8 veranschaulicht, oder, wie im obigen Beispiel, getrennt mit einem Verweis (`ref`) auf die Definition. Der Vorteil der zweiten Methode ist, dass die Element-Definition mehrmals gebraucht werden kann, ähnlich wie der Typ `OrtsNameTyp` in diesem Beispiel.

**Listing 28.8** *BSP28\_06.xsd*: Ein Schema, das alle Elemente verschachtelt deklariert

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="Kontinente">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="Kontinent" maxOccurs="unbounded" />
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="Name" type="xsd:string" minOccurs="1" />
            <xsd:element name="LängsterStrom" type="xsd:string" minOccurs="0" />
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
</xsd:schema>
```



Die Beispieldatei *BSP28\_06.xsd* finden Sie auf der CD-ROM zum Buch im Ordner `\Beispiele\Kap28`.

Die bislang vorgestellten Schemas sind eher datenzentrisch ausgerichtet. Erlaubt ist eigentlich nur eine Auflistung von Kontinenten und ihren Flüssen, ähnlich wie eine Datenbanktabelle. Was ist, wenn ein weniger strukturiertes Format erwünscht ist, und die Informationen in einem Textfluss markiert sein sollen:

```
<Kontinente>Jeder Kontinent hat seine Flüsse. Der bekannteste ist meistens der größte oder
längste. <Kontinent>Der längste <Name>Afrika</Name>s ist sehr bekannt, nicht zuletzt, weil
Agatha Christie ein Stück schrieb, das sich darauf abspielt: "Tod auf dem
<LängsterStrom>Nil</LängsterStrom>"</Kontinent></Kontinente>
```

Diese Art von Markup ist HTML-ähnlich. Um es in einem Schema zu definieren, muss das Attribut `mixed` in der Element-Definition vorhanden sein:

```
<xsd:element name="Kontinente">
  <xsd:complexType mixed="true">
```

Auch mit `mixed="true"` bleibt die erlaubte Zusammensetzung von Elementen im XML-Dokument strukturiert: sie müssen immer noch in der vordefinierten Hierarchie und Reihenfolge erscheinen.

Um eine zwanglose Zusammenstellung von Elementen zu erreichen, wie im folgenden Text, wird das XSD-Element choice benötigt.

```
<Text>...Nach einer langen Reise durch <Kontinent>Afrika</Kontinent>, fließt der
<Strom>Nil</Strom> an <Stadt>Kairo</Stadt>, der größten Stadt Aegyptens, vorbei</Text>
```

Das Element `<xsd:choice minOccurs="0" maxOccurs="unbounded" >` wird an die Stelle von sequence gesetzt. Die Attribute `minOccurs` und `maxOccurs` in dieser Kombination bedeuten, dass die Anzahl der Elemente unbegrenzt ist (es dürfen auch keine vorkommen):

```
<xsd:element name="Text">
  <xsd:complexType mixed="true">
    <xsd:choice minOccurs="0" maxOccurs="unbounded" >
      <xsd:element name="Kontinent" type="xsd:string" />
      <xsd:element name="Strom" type="xsd:string" />
      <xsd:element name="Stadt" type="xsd:string" />
    </xsd:choice>
  </xsd:complexType>
</xsd:element>
```

Letztlich stellt sich die Frage, wie alles erlaubt werden kann; jedes Element, jedes Attribut. Auch das ist möglich, mit dem Element `<any>` sowie dem Attribut `<anyAttribute>`. Damit wird jede Strukturierung dem Schema entzogen, was seinem Zweck zuwider spricht. Viele XML-Parser haben Probleme, Dokumente mit solchen Schemas zu validieren.

## XML-Namensräume und Schemas

Das Konzept des Namensraums spielt eine wichtige Rolle bei der Implementierung von XML in Word 2003. Es wird in Büchern und Dokumentationen erwähnt, aber die Erklärungen sind oft verwirrend oder ein bisschen dürftig.

Im letzten Abschnitt wurde vorgestellt, wie ein Schema die Datenstruktur für eine XML-Datei festlegt. Die Diskussion vermittelt den Eindruck, dass eine XML-Datei mit nur einem Schema verbunden sein kann; dies ist jedoch nicht der Fall. Eine XML-Datei darf mit mehreren Schemas verbunden werden und Daten für jedes enthalten. Da die gleichen Bezeichnungen für Elemente in mehreren Schemas vorkommen können, wird eine Methode benötigt, um ein Element eindeutig mit einem bestimmten Schema zu verknüpfen. Zu diesem Zweck wurde für den XML-Standard das Konzept des Namensraums entwickelt.

Stellen wir uns nun eine XML-Datei vor, die verschiedene Datenarten – beispielsweise eine Sammlung von Büchern und CDs – vereint:

```

<Sammlung>
  <Buch>
    <ID>1000</ID>
    <!-- (weitere Buch-Elemente folgen) -->
  </Buch>
  <CD>
    <ID>2000</ID>
    <!-- (weitere CD-Elemente folgen) -->
  </CD>
</Sammlung>

```

Die Gültigkeit eines jeden Eintrags soll je nach Datenart mit einem Schema geprüft werden; Bücher mit einem Schema für Bücher, CDs mit einem Schema für CDs. Die Fragen lauten:

- Wie werden die Elemente gekennzeichnet, so dass der Prozessor weiß, welches Element zu welchem Schema gehört?
- Wie werden mehrere Schemas mit der XML-Datei verbunden?

Die kurze Antwort auf die erste Frage lautet: Die Elemente werden Namensräumen zugewiesen. Durch den Namensraum weiß der Parser, in welchem Schema nachzuschauen ist.

Ein Namensraum wird im `<xsd:schema>`-Element eines Schemas deklariert, wie die dritten bis sechsten Zeilen in Listing 28.9 veranschaulichen.

**Listing 28.9** *BSP28\_07.xsd*: XML-Schema für ein einziges Buch

```

<?xml version="1.0" encoding="UTF-8" ?>
<xsd:schema
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.KAP28Beispiel.com/BSP07"
  xmlns:bu="http://www.KAP28Beispiel.com/BSP07"
  elementFormDefault="qualified">

  <xsd:element name="Buch">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="ID" type="bu:BuchIDTyp" />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:simpleType name="BuchIDTyp" >
    <xsd:restriction base="xsd:integer" >
      <xsd:minInclusive value="1000" />
      <xsd:maxInclusive value="1999" />
    </xsd:restriction>
  </xsd:simpleType>

```



Die Beispieldatei *BSP28\_07.xsd* finden Sie auf der CD-ROM zum Buch im Ordner `\Beispiele\Kap28`.

Dieses Schema legt fest, dass sich im XML-Dokument ein Element namens *Buch* befinden muss. Verschachtelt in diesem Element ist ein weiteres Element namens *ID*, das eine Ganzzahl im Bereich zwischen 1000 und 1999 sein muss.

Das erste Attribut des `<xsd:schema>`-Elements teilt dem Prozessor (einem XML-Parser beispielsweise) mit, dass jeder Elementname, dem das Namensraumpräfix `xsd` voransteht, dem Namensraum mit dem URI `http://www.w3.org/2001/XMLSchema` gehört:

```
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
```

**HINWEIS** Ein URI ist eine eindeutige Bezeichnung (»Uniform Resource Identifier«). URIs sind oft Web-Adressen, müssen es aber nicht sein. Es ist nicht gesagt, dass das Schema sich dort befindet oder dass die Web-Adresse überhaupt existiert. Wichtig ist nur, dass die Bezeichnung eindeutig ist, was bei Web-Adressen bestimmt der Fall ist.

Das zweite Attribut übermittelt dem Parser die Bezeichnung (URI) eines Namensraums, der im Schema verwendet wird. Beide Elemente, die in diesem Schema definiert werden, werden diesem Namensraum zugewiesen.

```
targetNamespace="http://www.KAP28Beispiel.com/BSP07"
```

Da die Möglichkeit besteht, dass die Elementnamen `Buch` sowie `BuchTypID` in anderen Schemas vorkommen könnten, wird weiter angewiesen, dass allen in diesem Schema definierten Elementnamen das Namensraumpräfix `bu` voranzustellen ist.

```
xmlns:bu="http://www.KAP28Beispiel.com/BSP07"
```

Schließlich legt das letzte Attribut `elementFormDefault` fest, ob verschachtelte Elemente mit dem Namensraumpräfix zu bezeichnen sind. Der standardmäßige Wert ist `unqualified`, was einem »Nein« gleichkommt. Es ist jedoch weniger verwirrend, wenn alle Elemente damit bezeichnet sind, weshalb in diesem Beispiel dem Attribut der Wert `qualified` zugewiesen wird.

```
elementFormDefault="qualified"
```

In Listing 28.10 befindet sich ein Schema für CD-Daten. Bitte beachten Sie, dass es einen anderen URI einsetzt als das Buch-Schema und einen anderen Bereich für den `ID`-Wert festlegt.

**Listing 28.10** *BSP28\_08.xsd*: XML-Schema für eine einzige CD

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsd:schema
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.KAP28Beispiel.com/BSP08"
  xmlns:cd="http://www.KAP28Beispiel.com/BSP08"
  elementFormDefault="qualified">

  <xsd:element name="CD">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="ID" type="cd:CDIDTyp" />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
```

Listing 28.10 *BSP28\_08.xsd*: XML-Schema für eine einzige CD (Fortsetzung)

```
<xsd:simpleType name="CDIDTyp" >
  <xsd:restriction base="xsd:integer" >
    <xsd:minInclusive value="2000" />
    <xsd:maxInclusive value="2999" />
  </xsd:restriction>
</xsd:simpleType>
```



Die Beispieldatei *BSP28\_08.xsd* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap28*.

Jetzt werden die zwei Schemas in einem »Überschema« vereint, wie Listing 28.11 veranschaulicht.

Listing 28.11 *BSP28\_09.xsd*: XML-Schema für Bücher und CDs gemischt, das die zwei einzelnen Schemas importiert

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsd:schema
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.KAP28Beispiel.com/BSP09"
  xmlns:bu="http://www.KAP28Beispiel.com/BSP07"
  xmlns:cd="http://www.KAP28Beispiel.com/BSP08"
  elementFormDefault="qualified">

  <xsd:import
    namespace="http://www.KAP28Beispiel.com/BSP07"
    schemaLocation="BSP28_07.xsd"/>
  <xsd:import
    namespace="http://www.KAP28Beispiel.com/BSP08"
    schemaLocation="BSP28_08.xsd" />

  <xsd:element name="Sammlung">
    <xsd:complexType>
      <xsd:choice minOccurs="0" maxOccurs="unbounded" >
        <xsd:element ref="bu:Buch" />
        <xsd:element ref="cd:CD" />
      </xsd:choice>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```



Die Beispieldatei *BSP28\_09.xsd* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap28*.

Wie in den beiden anderen Schemas wird hier ebenfalls ein eindeutiger Namensraum (*targetNamespace*) angelegt. (Ein neues Namensraumpräfix wird nicht benötigt, weil die Elemente alle in anderen Schemas definiert wurden.) Zudem wird jeweils ein *xmlns*-Attribut mit Namensraumpräfix gebraucht, für jedes der zwei zu importierenden Schemas.

Dem *<xsd:schema>*-Element folgen zwei *<xsd:import>*-Elemente. Diese geben den URI der Schemas im *namespace*-Attribut an (so dass der Prozessor sie mit den *xmlns*-Attributen verbinden kann) sowie die Pfadangabe zu den *\*.xsd*-Dateien im *schemaLocation*-Attribut.

**HINWEIS** Das Auffinden des Speicherorts eines Schemas wird von Prozessoren und Parsern unterschiedlich gehandhabt. Manche benutzen Attribute wie `schemaLocation`, um es einzubinden. Andere erwarten eine Angabe über eine interne Schnittstelle. In Word 2003 beispielsweise muss das Schema der Schemabibliothek hinzugefügt werden, bevor Word es für die Validation heranziehen kann.

Standardisierte Schema-Informationen, wie für XHTML, XSLT (Transformationen) oder XSD (Schemas) sind in vielen XML-Anwendungen schon eingebaut. Das Vorhandensein des korrekten URI genügt in diesen Fällen.

Das vom Schema definierte Wurzelement heißt *Sammlung*. Darunter dürfen beliebig viele *bu: Buch*- und *cd: CD*-Elemente verschachtelt werden, in beliebiger Reihenfolge (`<xsd:choice minOccurs="0" maxOccurs="unbounded" >`).

Erforschen wir nun die Wirkung dieser Schemas und Namensräume und betrachten zunächst die XML-Datei in Listing 28.12, die lediglich das Wurzelement *Sammlung* enthält. Diese Datei ist sowohl wohlgeformt als auch gültig.

Listing 28.12 *BSP28\_10.xml*: XML-Dokument mit einer leeren Sammlung

```
<?xml version="1.0" encoding="UTF-8" ?>
<Sammlung
  xmlns="http://www.KAP28Beispiel.com/BSP09" >
</Sammlung>
```

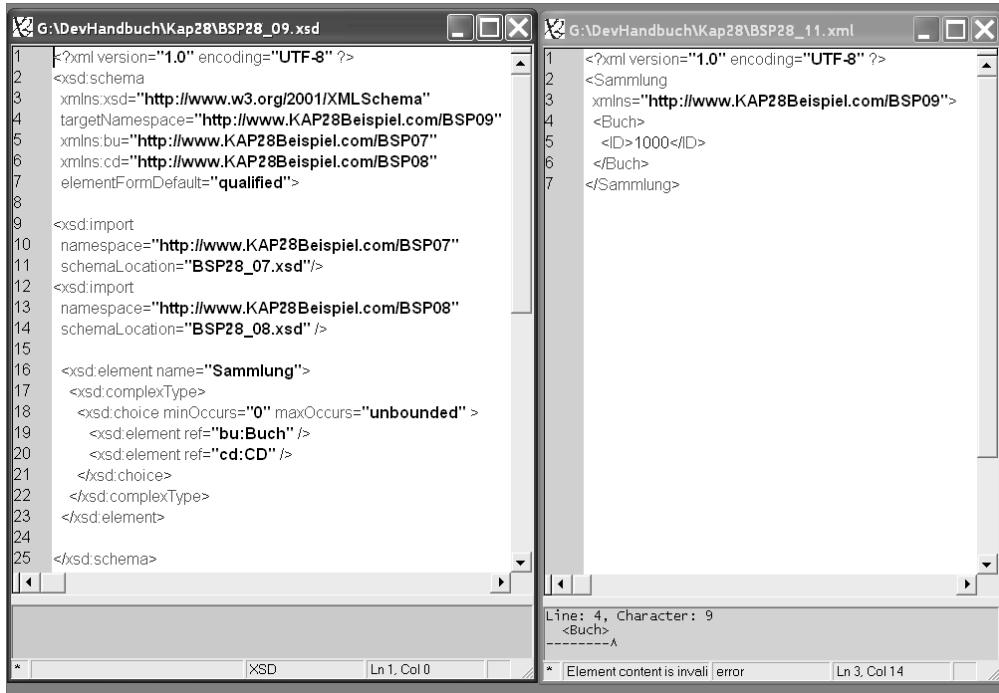
**HINWEIS** Da kein Namensraumpräfix Teil der `xmlns`-Deklaration ist, wird dieser Namensraum als standardmäßiger der XML-Datei in Listing 28.12 betrachtet. Der XML-Parser wird annehmen, dass alle Elemente ohne Namensraumpräfix (unqualified) diesem Namensraum angehören.

Wird diese Datei um ein Buch-Element und seine ID erweitert, wie in Listing 28.13, wird es ungültig, weil das Element *Buch* nicht dem standardmäßigen Namensraum (dem *Sammlung*-Schema) angehört (Abbildung 28.4).

Listing 28.13 *BSP28\_11.xml*: XML-Dokument mit einem *Buch*-Element; erster Versuch

```
<?xml version="1.0" encoding="UTF-8" ?>
<Sammlung
  xmlns="http://www.KAP28Beispiel.com/BSP09" >
  <Buch>
    <ID>1000</ID>
  </Buch>
</Sammlung>
```

Abbildg. 28.4 Die XML-Datei (rechts) ist nicht gültig (Fehlermeldung unten rechts), wenn sie mit dem Schema (links) geprüft wird.



#### HINWEIS

Im Internet sind verschiedene XML-Werkzeuge erhältlich. Die leistungsfähigsten (beispielsweise XMLSpy von <http://www.altova.com>) sind aber relativ teuer.

Es gibt jedoch einige kostenlose, die sich für einfachere Aufgaben eignen, wie Architag's »XRay XML Editor« (<http://architag.com/xray/>). Dieser prüft dynamisch die Wohlgeformtheit eines Fensterinhalts und zeigt an, wo der Fehler liegt. Zudem prüft er die Gültigkeit einer XML-Datei mit einem geöffneten Schema, wie in Abbildung 28.4 ersichtlich. (Dieses Programm erkennt nicht UTF-8, sondern nur ANSI. Es beklagt sich über die drei ersten Zeichen (das BOM) am Dateianfang und stellt Umlaute inkorrekt dar. Diese können für die Bearbeitung gelöscht und später durch nochmaliges Speichern als UTF-8-Datei im Windows-Editor oder in Word wieder hinzugefügt werden.)

Der nächste Versuch, in Listing 28.14, enthält zusätzlich den Namensraum für Buch-Elemente und den Elementnamen wurden Namensraumpräfixe vorangestellt. Diese Version ist gültig. Bitte beachten Sie, dass es erlaubt ist, in der XML-Datei ein anderes Namensraumpräfix als das im Schema definierte zu verwenden.

Listing 28.14 *BSP28\_12.xml*: XML-Dokument mit Namensraum und -präfix für das *Buch*-Element. Diese Datei ist gültig.

```
<?xml version="1.0" encoding="UTF-8" ?>
<Sammlung
  xmlns="http://www.KAP28Beispiel.com/BSP09"
```

**Listing 28.14** *BSP28\_12.xml*: XML-Dokument mit Namensraum und -präfix für das *Buch*-Element. Diese Datei ist gültig. (Fortsetzung)

```

xmlns:b="http://www.KAP28Beispiel.com/BSP07">
  <b:Buch>
    <b:ID>1000</b:ID>
  </b:Buch>
</Sammlung>

```

Schließlich veranschaulichen Listing 28.15 und Abbildung 28.5 ein gültiges XML-Dokument mit *Buch*- sowie *CD*-Element.

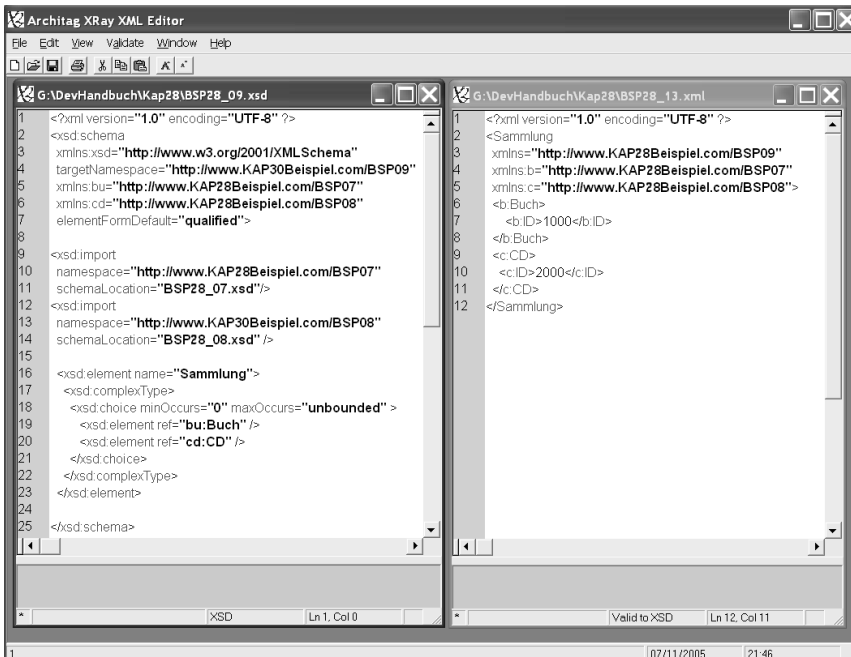
**Listing 28.15** *BSP28\_13.xml*: Gültiges XML-Dokument mit je einem *Buch*- sowie *CD*-Element

```

<?xml version="1.0" encoding="UTF-8" ?>
<Sammlung
  xmlns="http://www.KAP28Beispiel.com/BSP09"
  xmlns:b="http://www.KAP28Beispiel.com/BSP07"
  xmlns:c="http://www.KAP28Beispiel.com/BSP08">
  <b:Buch>
    <b:ID>1000</b:ID>
  </b:Buch>
  <c:CD>
    <c:ID>2000</c:ID>
  </c:CD>
</Sammlung>

```

**Abbildg. 28.5** »Alles im grünen Bereich«: Das XML-Dokument (rechts) erfüllt die Bedingungen des Schemas (links) und ist gültig





# XML-Daten manipulieren

Wie im Abschnitt »XML-Daten transformieren« in diesem Kapitel bereits kurz dargestellt, ist eine der Stärken von XML die Wiederverwendbarkeit. Die Daten können beispielsweise programmtechnisch angesprochen oder mittels einer Transformation als HTML-Seiten dargestellt werden. Dieser Abschnitt gibt einen Überblick über diese Aspekte.

## Das XML-Dokument-Objektmodell (DOM)

Bislang wurden Aufbau und Struktur von XML-Dokumenten aus einer manuellen Perspektive vorgestellt. XML-Dokumente können aber auch über eine programmtechnische Schnittstelle erstellt, geprüft und transformiert werden. Hierfür stellt der XML-Standard zwei Vorgehensweisen zur Verfügung:

- die SAX (»Simple API for XML«)
- das XML-Dokument-Objektmodell (»XML Document Object Model« DOM)

Die SAX-Methode basiert auf der Ereignis-Schnittstelle eines XML-Dokuments. Das Dokument wird Element für Element, Attribut für Attribut gelesen. Dadurch werden Ereignisse ausgelöst (wenn beispielsweise das nächste Element vorliegt). Um auf diese Ereignisse zu reagieren, benutzt die Anwendung »call-back«-Funktionen (»listener«). Dieses Modell ist schnell und braucht wenig Ressourcen, ist jedoch nicht sehr flexibel. Das Dokument wird vom Anfang bis zum Ende durcharbeitet und gibt keine Informationen über den Kontext der aktuell bearbeiteten Stelle zurück.

### HINWEIS

Mehr über die Verwendung von XML-Dokument-Ereignissen finden Sie unter der Webadresse <http://www.schumacher-netz.de/TR/2003/REC-xml-events-20031014-de.html>.

Das XML-DOM hingegen liest das gesamte XML-Dokument und baut eine hierarchische Struktur – Quellbaum – der Elemente und Attribute (»Nodes« – Knotenpunkte) auf. Diese können dann programmtechnisch in beliebiger Reihenfolge durch objektorientierte Schnittstellen angesprochen werden.

### HINWEIS

Das XML-DOM darf nicht mit dem Dokument-Objektmodell des Internet Explorers verwechselt werden, das geladene Webseiten anspricht und manipuliert. Obwohl gewisse Konzepte ähnlich sind, ist das XML-DOM spezifisch für die Bearbeitung von XML-Dokumenten.

Mehr Informationen zum DOM finden Sie auf der W3C-Webseite, beginnend bei <http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001>.

Der in der COM-Umgebung tätige Programmierer findet Unterstützung für sowohl die SAX als auch die DOM im Microsoft MSXML (MS XML Core Services) SDK. Es besitzt auch Schnittstellen für XML Schema Processing sowie XSLT. Die mit Word 2003 installierte Version und Hilfe-Datei ist MSXML 5.0 für Microsoft Office Applications.

**HINWEIS**

Mehr über das MSXML SDK und seine Interaktion mit DOM befindet sich bei <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/xmlsdk/html/b24aafc2-bf1b-4702-bf1c-b7ae3597eb0c.asp>. Die allgemeine Version des MSXML SDKs steht zum Herunterladen bereit bei <http://www.microsoft.com/downloads/details.aspx?FamilyID=2cf40ae6-368c-4b6b-a185-2dfa92fb7993&DisplayLang=en>.

Das .NET Framework stellt System.XML-Klassen für die Arbeit mit XML-Dokumenten, XML-Schemas, XSLT, XPath und das XML-DOM zur Verfügung. Diese sind in der MSDN-Bibliothek dokumentiert.

Das Word 2003-Objektmodell bietet keine eigene DOM-Schnittstelle für in Word geöffnete XML-Dokumente. Der Programmierer kann sich in einem Word-VBA-Projekt des MSXML-Parsers bedienen.

## XSLT

Viele Aufgaben können mit XSLT (Extensible Stylesheet Language Transformation) statt auf Basis des DOM gelöst werden. Im Zusammenspiel mit XPath wird es zum schlagkräftigen Werkzeug, das die Daten eines XML-Dokuments in beliebiger Reihenfolge herauslesen und neu zusammenstellen kann. Die Grundregeln für Transformationen bilden ein XML-Vokabular. Eine \*.xsl-Datei ist ein wohlgeformtes und gültiges XML-Dokument, das beschreibt, wie die Daten einer \*.xml-Datei dynamisch darzustellen sind. In diesem Abschnitt werden wir lediglich einen Überblick zu dieser großen und komplexen Sprache darstellen.

Mittlerweile gibt es die Versionen 1.0 und 2.0. Der Microsoft XML-Parser, MSXML, .NET Framework 1.0 und 1.1 sowie Word 2003 unterstützen Version 1.0.

XSLT ist eine Komponente der XSL-Sprache, die ursprünglich entwickelt wurde, um XML-Daten in »print-ready«-Dokumente zu transformieren. Die andere Hauptkomponente ist ein Vokabular für die Formatierung und heißt XSL-FSO (»XSL Formatting Objects«). Zudem gibt es die Sprache XPath, die definiert, wie Teile eines XML-Dokuments ausgewählt werden.

**HINWEIS**

Mehr über XPath erfahren Sie bei <http://www.obqo.de/w3c-trans/xpath-de>.

Im Gegensatz zur programmtechnischen Bearbeitung eines XML-Dokuments über das DOM, wo meistens jeder Knotenpunkt individuell angesprochen wird, legt eine Transformation einen Satz allgemeiner Kriterien fest, womit die XML-Daten gefiltert und neu zusammengestellt werden. Konzeptuell kann eine Transformation mit einer SQL-Anweisung verglichen werden, die beschreibt, welche Daten auszuwählen sind. Es können beispielsweise alle Adresse-Elemente, deren Art »Geschäft« ist, ausgewählt werden, oder nur die Flüsse, die sich in Afrika befinden.

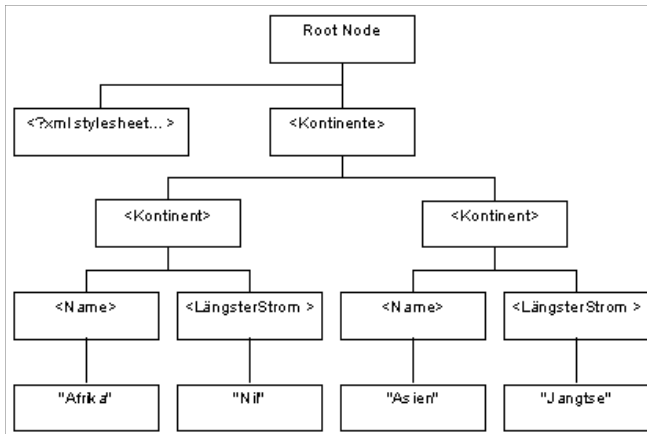
Die Anweisungen einer Transformation definieren einen Mustervergleich, um Knotenpunkte zu identifizieren und zu beschreiben, was damit zu tun ist. Ihre Reihenfolge ist nicht vorgeschrieben wie bei SQL-Anweisungen, und es werden keine Fehler verursacht, wenn Elemente oder Attribute angesprochen werden, die sich nicht im XML-Dokument befinden. Ein Beispiel zur Datenauswahl und Bearbeitung:

- für jedes Kontinent-Element eine neue Tabellenzeile erstellen, mit einer Zelle für jedes unmittelbar darauf folgende Unterelement;

- falls es sich um ein Name-Element handelt, den Zellenhintergrund blau färben.

Ähnlich wie das DOM betrachtet XSLT den Inhalt eines XML-Dokuments als eine Hierarchie von Knotenpunkten und unterscheidet dabei zwischen sieben verschiedenen Arten: Wurzelknotenpunkt (dem alle anderen untergeordnet sind), Element-Knotenpunkte, Attribut-Knotenpunkte, Kommentar-Knotenpunkte, Verarbeitungsanweisungs-Knotenpunkte, Namensraum-Knotenpunkte sowie Text-Knotenpunkte. Obwohl das Kontinente-Beispiel nicht komplex ist, lässt es sich grafisch als Hierarchie darstellen (Abbildung 28.6).

Abbildg. 28.6 Hierarchie der Knotenpunkte des XML-Dokuments *Kontinente*



Die XSLT in Listing 28.3 veranschaulicht diese Konzepte. Jedes Kriterium identifiziert einen Satz von Knotenpunkten, die es zu bearbeiten gilt. Das einzige im Listing vorhandene Kriterium lautet:

```
<xsl:template match="/">
```

Dieses Kriterium wird oft verwendet und weist XSLT an, den Wurzelknotenpunkt des XML-Dokuments zu finden. Da dieser Knotenpunkt nur einmal vorkommt, werden er und die darunter verschachtelten Anweisungen nur ein einziges Mal ausgeführt. Bei der Durcharbeitung der unterstellten Anweisungen trifft der Prozessor auf eine Schleife, wie sie in den meisten Programmiersprachen vorkommt:

```

<xsl:for-each select="Kontinente/Kontinent" >
  <tr>
    <td><xsl:value-of select="Name" /></td>
    <td><xsl:value-of select="LängsterStrom" /></td>
  </tr>
</xsl:for-each>

```

Es gibt jedoch eine Alternative, die dem zugrunde liegenden Konzept des Mustervergleichs näher kommt, wie Listing 28.16 veranschaulicht. Darin basiert die Transformation des Kontinente-XML-Dokuments auf Mustervergleichungen statt auf programmtechnischen Konstrukten.

**Listing 28.16** *BSP28\_14.xsl*: Eine Transformation des Kontinent-XML-Dokuments, basierend auf Mustervergleichen

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:template match="/">
    <html>
      <head>
        <title>Kontinente</title>
      </head>
      <body>
        <table border="2">
          <tr>
            <th>Kontinent</th>
            <th>Längster Strom</th>
          </tr>
          <xsl:apply-templates select="Kontinente/Kontinent" />
        </table>
      </body>
    </html>
  </xsl:template>

  <xsl:template match=" Kontinent" >
    <tr>
      <xsl:apply-templates />
    </tr>
  </xsl:template>

  <xsl:template match="Name|LängsterStrom" >
    <td><xsl:apply-templates /></td>
  </xsl:template>

</xsl:stylesheet>
```



Die Beispieldateien *BSP28\_14.xsl* mit der Transformation sowie *BSP28\_15.xml*, die auf diese hinweist, finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap28*.

Die Arbeitsweise dieser Transformation kann wie folgt interpretiert werden. Wie erwähnt, identifiziert `<xsl:template match="/">` den Wurzelknotenpunkt des XML-Dokuments und veranlasst den Prozessor, alle untergeordneten Anweisungen einmal auszuführen.

Der sich hierunter befindende Text (HTML-Tags) wird unverändert direkt in das Ergebnis übernommen. Wenn der Prozessor einem `<xsl:->`-Tag begegnet, wird die darin stehende Anweisung ausgewertet: `<xsl:apply-templates select="Kontinente/Kontinent" />`.

`apply-templates` weist den Prozessor an, ein `<xsl:template>`-Tag zu finden, dessen Match-Attribut dem XPath-Ergebnis des Select-Attributs entspricht. "Kontinente/Kontinent" gibt jeden Kontinent-Knotenpunkt zurück, der sich direkt unter dem Wurzelknotenpunkt Kontinente befindet.

Jedes Mal, wenn der Prozessor einen entsprechenden Knotenpunkt findet, werden die Anweisungen unter `<xsl:template match=" Kontinent" >` ausgeführt:

```
<xsl:template match=" Kontinent" >
  <tr>
    <xsl:apply-templates />
  </tr>
</xsl:template>
```

Zuerst wird das Anfangs-Tag für eine neue Tabellenzeile in die HTML-Zeichenkette geschrieben. Dann kommt noch eine `apply-templates`-Anweisung, dieses Mal ohne `select`-Attribut (ohne Mustervergleich). Das heißt für XSLT, dass es `<xsl:template match=>` für alle sich unter jedem Knotenpunkt *Kontinent* befindenden Knotenpunkte (Elemente) suchen soll.

In diesem Fall können zwei Element-Namen vorkommen: *Name* sowie *LängsterStrom*. Deshalb enthält die Transformation folgende Zeilen, wo *Match* entweder gleich *Name* oder *LängsterStrom* ist.

```
<xsl:template match="Name|LängsterStrom" >
  <td> <xsl:apply-templates /> </td>
</xsl:template>
```

Für jeden Knotenpunkt wird der aktuellen Tabellenzeile eine Zelle hinzugefügt. Dann kommt noch einmal `<xsl:apply-templates />`. In diesem Fall sind die Knotenpunkte der sich darunter befindenden Ebene die Text-Knotenpunkte (siehe Abbildung 28.6), weshalb der Dateninhalt in die Tabellenzellen geschrieben wird.

Dieses Beispiel hebt hervor, dass XSLT keine Programmiersprache ist, sondern auf der Basis von Kriterien arbeitet. Der beschriebene Fall setzt jedoch eine eingehende Kenntnis von Datenstruktur und -inhalt des KML-Dokuments voraus, um das erwartete Ergebnis (Abbildung 28.1) zu erreichen. Nicht berücksichtigt in dieser Lösung werden beispielsweise:

- Elemente zusätzlich zu *Name* und *LängsterStrom*. Solche erscheinen als Text, außerhalb der Tabelle.
- Fehlt eines dieser Elemente, werden nicht alle Tabellenzeilen gleich viele Zellen haben.
- Sind mehrere Elemente mit dem gleichen Namen vorhanden, befinden sich jedoch in verschiedenen Namensräumen oder Knotenpunkt-Hierarchien und müssen deshalb anders behandelt werden, müssen die Mustervergleiche angepasst und dafür getrennte `<xsl:template>`-Anweisungen hinzugefügt werden. (Denken Sie an das Buch/CD-Beispiel zurück, wo unter Umständen eine Anweisung für Buch/ID und eine andere für CD/ID nötig sein könnte.)

#### HINWEIS

Eine komplexe XSLT können Sie von der MSDN-Seite herunterladen und prüfen: »Transforming Word Documents into the XSL-FO Format« unter [http://msdn.microsoft.com/library/default.asp?url=/library/en-us/odc\\_wd2003\\_ta/html/OfficeWordWordMLtoXSL-FO.asp](http://msdn.microsoft.com/library/default.asp?url=/library/en-us/odc_wd2003_ta/html/OfficeWordWordMLtoXSL-FO.asp). Wenn Sie die XSL-FO Spezifikation bei W3C betrachten, fragen Sie sich vielleicht, warum Microsoft nicht diese für Word benutzte, sondern ein eigenes WordProcessingML-Vokabular entwickelte. Wahrscheinlich hätte XML-FSO Mühe, gewisse Funktionen eines modernen Textverarbeitungsprogramms so zu codieren, dass XML wieder in ein vollwertiges Word-Dokument konvertiert werden könnte.

**HINWEIS** Für eingehendere Informationen zu allen vorgestellten XML-Themen müssen Sie in XML-spezifischer Dokumentation nachschlagen. Neben den aufgeführten URLs gibt es einige Bücher auf dem Markt, unter anderen:

- »XML Pocket Consultant«, Autor William R. Stanek, Microsoft Press, ISBN 0-7356-1183-1 (englisch)
- »XML Schritt für Schritt«, Autor Michael J. Young, MS Press, ISBN 3-86063-765-7 (deutsche Übersetzung)

## XML in Word 2003

Word 2003 ist die erste Version, die XML-Unterstützung anbietet. Es umfasst zwei Schwerpunkte: das XML-Dateiformat (WordProcessingML) sowie die Möglichkeit, Tags eines anderen XML-Vokabulars in der Word-Dokumentstruktur einzubetten. Dieser Abschnitt bietet einen Überblick des WordProcessingML-Vokabulars und stellt die Werkzeuge für die Arbeit mit Word-fremden XML-Vokabularen vor.

### WordProcessingML

Bei Microsofts WordProcessingML handelt es sich um ein XML-Vokabular, das ein Word-Dokument vollständig beschreibt. Eine in diesem Dateiformat gespeicherte Datei enthält alle Informationen, um sich in der Word-Umgebung wie ein Word-Dokument zu verhalten, und lässt sich ohne Daten- oder Formatierungsverlust wieder problemlos als Word-Dokument speichern. Von XML bemerkt der Anwender nichts; genau wie bei der Arbeit mit Dateien im RTF- oder »Webseite«-Format bleiben die Tags verborgen. Das Vokabular ist vollständig beschrieben im Microsoft Office 2003 Word XML SDK.

Ein einfaches Beispiel des Vokabulars enthält das Listing 28.17; wie Word dieses auf dem Bildschirm darstellt, sehen Sie in Abbildung 28.7.

**Listing 28.17** *BSP28\_16.xml: Die Kontinente-Tabelle in WordProcessingML*

```
<?xml version="1.0" encoding="UTF-8"?>
<w:wordDocument
  xmlns:w="http://schemas.microsoft.com/office/word/2003/wordml" >
  <w:body>
    <w:tbl>
      <w:tr>
        <w:tc><w:p><w:r><w:t>Kontinent</w:t></w:r></w:p></w:tc>
        <w:tc><w:p><w:r><w:t>Längster Strom</w:t></w:r></w:p></w:tc>
      </w:tr>
      <w:tr>
        <w:tc><w:p><w:r><w:t>Afrika</w:t></w:r></w:p></w:tc>
        <w:tc><w:p><w:r><w:t>Nil</w:t></w:r></w:p></w:tc>
      </w:tr>
      <w:tr>
        <w:tc><w:p><w:r><w:t>Asien</w:t></w:r></w:p></w:tc>
        <w:tc><w:p><w:r><w:t>Jangtse</w:t></w:r></w:p></w:tc>
      </w:tr>
    </w:tbl>
  </w:body>
</w:wordDocument>
```

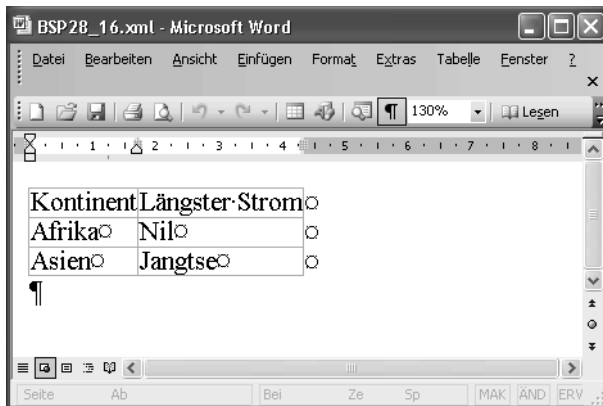
Listing 28.17 BSP28\_16.xml: Die Kontinente-Tabelle in WordProcessingML (Fortsetzung)

```
</w:tbl>
<w:p/>
</w:body>
</w:wordDocument>
```



Die Beispieldatei *BSP28\_16.xml* finden Sie auf der CD-ROM zum Buch im Ordner `\Beispiele\Kap28`.

Abbildg. 28.7 Ein WordProcessingML-Dokument mit Tabelle



WordProcessingML weist für eine Tabelle Ähnlichkeiten mit HTML oder XHTML auf: Der Anfang der Tabelle wird durch ein `<w:tbl>`-Tag, die Zeile durch `<w:tr>` und eine Zelle durch `<w:tc>` gekennzeichnet. Es ist jedoch nicht möglich, den Zelleninhalt zwischen den Tags `<w:tc>` und `</w:tc>` einzugeben. In WordProcessingML müssen sich zusätzlich ein Absatz-Element (`<w:p>`), ein Textlauf-Element (`<w:r>` – es umfasst Formatierungsbefehle für den darauf folgenden Text) sowie ein Element für den Text selbst (`<w:t>`) zwischen den `<w:tc>`-Tags befinden.

Obwohl dieses Beispiel im Vergleich zu seinem Gegenstück in HTML komplex erscheint, ist es gegenüber dem Inhalt eines von Word gespeicherten XML-Dokuments geradezu einfach. Das wird deutlich, wenn Sie die Beispieldatei in Word über *Datei/Speichern unter* als XML-Datei speichern und danach im Windows-Editor öffnen. Einen Teil des Ergebnisses sehen Sie in Listing 28.18; alle diese Informationen stammen aus den einleitenden Elementen (»Header«).



Eine bearbeitete Version des Ergebnisses (mit Zeilenschaltungen und Einzügen, um die Strukturen hervorzuheben) befindet sich in der Datei *BSP28\_17.xml* auf der CD-ROM zum Buch im Ordner `\Beispiele\Kap28`.

**Listing 28.18** Header-Abschnitt einer WordProcessingML-Datei

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<?mso-application progid="Word.Document"?>
<w:wordDocument
  xmlns:w="http://schemas.microsoft.com/office/word/2003/wordml"
  xmlns:v="urn:schemas-microsoft-com:vml"
  xmlns:w10="urn:schemas-microsoft-com:office:word"
  xmlns:sl="http://schemas.microsoft.com/schemaLibrary/2003/core"
  xmlns:aml="http://schemas.microsoft.com/aml/2001/core"
  xmlns:wx="http://schemas.microsoft.com/office/word/2003/auxHint"
  xmlns:o="urn:schemas-microsoft-com:office:office"
  xmlns:dt="uuid:C2F41010-65B3-11d1-A29F-00AA00C14882"
  w:macrosPresent="no" w:embeddedObjPresent="no"
  w:ocxPresent="no" xml:space="preserve">
```

Die ersten zwei Tags enthalten die übliche XML-Deklaration sowie eine Verarbeitungsanweisung, die signalisiert, dass es sich um eine WordProcessingML-Datei handelt. Das dritte Element, das Wurzelement, enthält einige Namensräume mehr als das Wurzelement aus Listing 28.17. Sie alle (siehe Tabelle 28.2) werden von Word automatisch hinzugefügt, egal ob das Dokument passende Objekte beinhaltet oder nicht.

**Tabelle 28.2** Von Word standardmäßig deklarierte Namenräume für WordProcessingML-Dokumente

Namensraum-präfix	Namensraum	Beschreibung
w	<i>http://schemas.microsoft.com/office/word/2003/wordml</i>	Grundlegende WordProcessingML-Elemente und -Attribute
vml	<i>urn:schemas-microsoft-com:vml</i>	Für Vector Markup Language (VML)-Grafiken
w10	<i>urn:schemas-microsoft-com:office:word</i>	Word XP-Elemente, die in Word 2003 nicht mehr vorhanden sind oder anders gehandhabt werden
sl	<i>http://schemas.microsoft.com/schemaLibrary/2003/core</i>	Benutzerdefinierte Schema-Unterstützung
aml	<i>http://schemas.microsoft.com/aml/2001/core</i>	Annotation Markup Language (AML) Elemente für Überarbeitungen und Kommentare
wx	<i>http://schemas.microsoft.com/office/word/2003/auxHint</i>	Für »Auxiliary Hints«, die Word hinzufügt, um dem Anwender zu helfen, das WordProcessingML zu entziffern. Beispiel: Word fügt ein <b>&lt;wx:sect&gt;</b> -Tag am Abschnittsanfang ein, beachtet es selbst aber nicht, sondern benutzt seine eigene, kryptische Codierung.
o	<i>urn:schemas-microsoft-com:office:office</i>	Office-Dokumenteigenschaften
dt	<i>uuid:C2F41010-65B3-11d1-A29F-00AA00C14882</i>	Wird in Zusammenhang mit Office-Dokumenteigenschaft-Attributen verwendet.



## PROFITIPP

Wie im Abschnitt »XML-Dokumentinstanz, Markup und Inhalt« in diesem Kapitel erwähnt, veranlasst die Verarbeitungsanweisung den Internet Explorer, eine WordProcessingML-Datei in Word zu öffnen. Das macht es schwierig, sich einen Überblick über den Dateiinhalt zu verschaffen. Entweder muss die Verarbeitungsanweisung entfernt oder der für die Verknüpfung verantwortliche Eintrag in der Registry geändert werden:

1. Führen Sie *Start/Ausführen* aus.
2. Geben Sie *regedit* ein und klicken Sie auf *OK*.
3. Navigieren Sie zum Eintrag `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Office\11.0\Common\Filter\text\xml`.
4. Ändern Sie die Zeichenkette (*Name*) des Eintrags »Word.Document« beispielsweise in »XWord.Document«.

Nach dem Header listet Word die Dokumenteigenschaften auf (Titel, Autor usw.). Es folgt eine Auflistung der im Dokument verwendeten Schriftarten und Formatvorlagen. Der folgende Auszug ist unvollständig, um besser die Struktur des Aufbaus zu vermitteln:

```
. <o:DocumentProperties>
  <o:Title>Kontinent</o:Title>
  <o:Author>Wolfgang Schmidt</o:Author>
..
</o:DocumentProperties>
<w:fonts>
  <w:defaultFonts
    w:ascii="Times New Roman" w:fareast="Times New Roman"
    w:h-ansi="Times New Roman" w:cs="Times New Roman" />
</w:fonts>
<w:styles>
  <w:versionOfBuiltInStylenames w:val="4" />
  <w:latentStyles w:defLockedState="off" w:latentStyleCount="156" />
  <w:style w:type="paragraph" w:default="on" w:styleId="Standard">
    <w:name w:val="Normal" />
..
</w:styles>
```

Es folgen weitere »Dokumenteigenschaften« (Einstellungen) wie die aktuelle Ansicht beim Speichern des Dokuments und die angehängte Vorlage. Erst dann folgt der Dokumentkörper:

```
<w:docPr>
  <w:view w:val="web" />
  <w:zoom w:percent="150" />
  <w:attachedTemplate w:val="" />
..
</w:docPr>
<w:body>

</w:wordDocument>
```

Eine detaillierte Behandlung des WordProcessingML-Vokabulars würde den Rahmen dieses Buchs sprengen, wir möchten aber auf einige Besonderheiten aufmerksam machen.

**HINWEIS** Die Einzelheiten von und der Umgang mit WordProcessingML werden in dem Buch »Office 2003 XML« von Evan Lenz, Mary McRae und Simon St. Laurent (O'Reilly, ISBN 0-596-00538-5) in englischer Sprache eingehend vorgestellt.

- Es ist möglich, Word-Dokumente ohne Mithilfe der Word-Anwendung zu erstellen, indem eine WordProcessingML-Datei generiert wird. Somit entfällt die Automatisierung Words auf einem Server mit allen dazugehörenden, lästigen Nachteilen.
- Dabei ist es nicht unbedingt notwendig, alle Tags und Attribute einzufügen, die Word beim Speichern eines Dokuments einsetzt. Wie das Listing 28.17 veranschaulicht, erkennt Word eine minimale Version.
- Um zu ermitteln, welche Elemente für ein Objekt oder eine Formatierung zuständig sind, speichern Sie ein möglichst einfaches Word-Dokument als XML und betrachten Sie das Ergebnis in einem Text- oder XML-Editor. Vergleichen Sie es mit den Angaben im Word 2003 XML SDK.
- Word unterstützt keinen gemischten Inhalt, wie in HTML üblich ist. Folgender Codeschnipsel, der den Text »Dieses Wort ist fett.« wiedergibt

```
<p>Dieses Wort ist <b>fett</b>.</p>
```

kann in WordProcessingML nicht vorkommen. Um den Beispielttext herum wird die folgende Element-Zusammensetzung benötigt. Bitte beachten Sie die Verwendung der Textläufe (<w:r>) sowie Textlaufeigenschaften (<w:rPr>), um die direkte Formatierung festzulegen. Der Formatierungsbehl wird in einem leeren Tag (<w:b/>) angegeben.

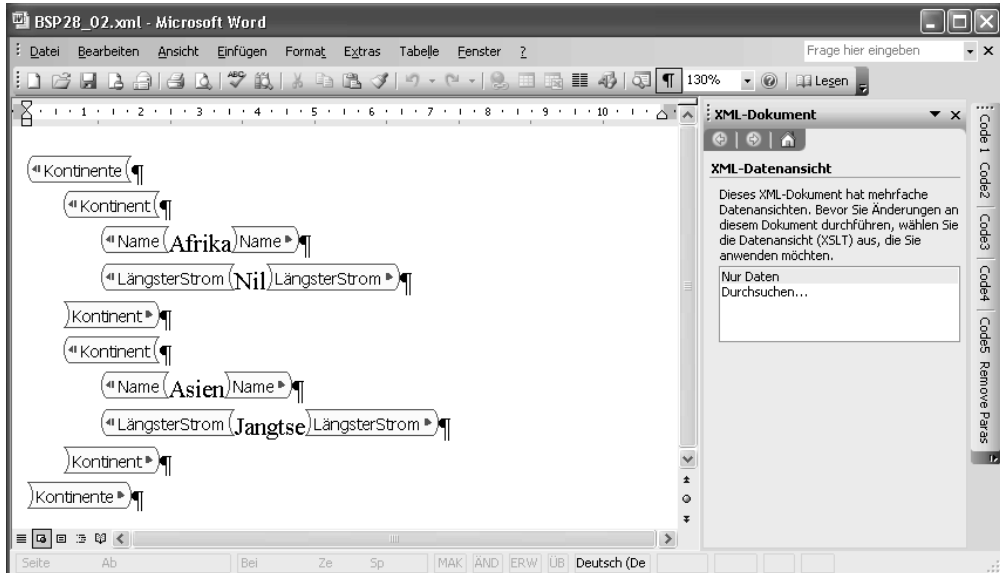
```
<w:p>
  <w:r>
    <w:t>Dieses Wort ist</w:t>
  </w:r>
  <w:r><rPr><w:b/></rPr>
    <w:t>fett</w:t>
  </w:r>
  <w:r>
    <w:t>.</w:t>
  </w:r>
</w:p>
```

## Benutzerdefiniertes XML

Zusätzlich zum WordProcessingML-Dateiformat enthält Word 2003 Schnittstellen, um andere XML-Vokabulare mit ihren Schemas, Namensräumen und Transformationen in Word-Dokumente zu integrieren. Diese Funktionalität ist zum größten Teil an den Entwickler gerichtet, um Smart Document-Lösungen bereitzustellen. Sie macht keinen benutzerfreundlichen XML-Editor aus Word.

Wenn Sie die Beispieldatei für Listing 28.2 (*BSP28\_02.xml* auf der CD) in Word öffnen, wird das Ergebnis ähnlich wie in Abbildung 28.8 aussehen. Word blendet den Aufgabenbereich *XML-Dokument* ein. Die XML-Tags werden in rosaroten, nicht bearbeitbaren Kartuschen angezeigt, deren Einzüge die Hierarchie der Elemente widerspiegeln.

Abbildg. 28.8 Die in Word geöffnete »KML«-Datei

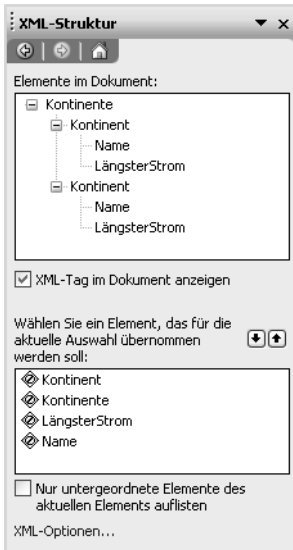


Klicken Sie auf den Pfeil neben dem Titel des Aufgabenbereichs und wählen Sie in der Liste weiterer verfügbarer Aufgabenbereiche den Eintrag *XML-Struktur* aus.

Dieser Aufgabenbereich (Abbildung 28.9) ist in zwei Abschnitte unterteilt: der obere zeigt eine hierarchische Auflistung der im Dokument *enthaltenen* Elemente sowie ein Kontrollkästchen, womit diese ein- und ausgeblendet werden können; der untere listet die *verfügbaren* Elemente auf. Falls das XML-Dokument mit einem Schema verknüpft ist, wird diese Information dem Schema entnommen, sonst werden (wie hier) nur die sich im Dokument befindenden angezeigt. Das dazugehörige Kontrollkästchen filtert die Liste, um nur kontextgültige Elemente anzubieten. (Da die Gültigkeit nur anhand eines Schemas geprüft werden kann, bleibt die gefilterte Liste leer, wenn kein Schema mit dem aktuellen Dokument verbunden ist.)

Da das abgebildete XML-Dokument mit keinem Schema verbunden ist, können überall im Text beliebig Elemente und Text eingefügt werden. Es kann als Word-Dokument (\*.doc), WordProcessingML-Datei oder als XML-Datei ohne Word-eigene Elemente (*Nur Daten speichern*) gespeichert werden.

**Abbildg. 28.9** Der Aufgabenbereich *XML-Struktur* dient der Verwaltung von XML-Elementen im Word-Dokument.



Während die freie Bearbeitung einer XML-Datei gelegentlich wünschenswert ist, kann ihre Gültigkeit nur mit Hilfe eines Schemas gewährleistet werden.

## Die Word-Schemabibliothek

Schemas werden in der »Schemabibliothek« zentral verwaltet, meistens für den einzelnen Benutzer. Die Schemabibliothek enthält keine Schemas, sondern nur Referenzen dazu. Für jedes eingetragene Schema werden folgende Angaben festgehalten:

- Der Speicherort des Schemas (Pfadangabe oder URL)
- Einen vom Anwender festgelegten URI (falls keiner im Schema vorhanden ist)
- Ein vom Anwender festgelegtes Namensraumpräfix (Alias)

Ein URI darf nur einmal in der Schemabibliothek vorkommen. Um einen URI nach Aufnahme eines Schemas zu ändern, muss das Schema aus der Schemabibliothek entfernt werden. Er kann dann in der XML-Datei geändert werden oder der Anwender weist beim erneuten Laden des Schemas einen anderen zu.

Das gleiche Namensraumpräfix darf zwar mehrmals benutzt werden, es ist jedoch weniger verwirrend, wenn jedem Schema ein eindeutiges Präfix zugewiesen wird.

Word schreibt weder den Speicherort noch das Namensraumpräfix eines Schemas in einem als XML gespeicherten Dokument fest. Nur der URI wird mitgespeichert. Beim Öffnen eines im XML-Format gespeicherten Dokuments vergleicht es die im Dokument vorhandenen URIs mit den URIs der in der Schemabibliothek stehenden Namensräume, um das Dokument mit den passenden Schemas zu verbinden.

Beim Aufnehmen eines Schemas in die Schemabibliothek und beim Verbinden eines Schemas mit einem Dokument führt Word mehrere Prüfungen durch. Diese können zu Fehlermeldungen führen, wenn eine Unstimmigkeit vorliegt.

Word hält Schemas im Arbeitsspeicher (Cache) vor. Deshalb muss Word neu gestartet werden, falls Sie ein in der Schemabibliothek vorhandenes Schema ändern.

Die Benutzerschnittstelle der Schemabibliothek ist einfach zu bedienen. Als Beispiel wird das Schema in Listing 28.19 für das »KML«-Vokabular geladen und mit einem Dokument (im vorgestellten Szenario mit der Beispieldatei *BSP28\_02.xml*) verbunden. (Dieses Schema unterscheidet sich von den vorangehenden durch die Deklaration eines `targetNamespace` und die Unterstützung von gemischtem Inhalt (`mixed="true"`).)

**Listing 28.19** *BSP28\_18.xsd*: Ein XML-Schema für das »KML«-Vokabular

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsd:schema
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.KAP28Beispiel.com/BSP18"
  elementFormDefault="qualified">
  <xsd:element name="Kontinente">
    <xsd:complexType mixed="true">
      <xsd:sequence>
        <xsd:element name="Kontinent" maxOccurs="unbounded" >
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="Name" type="xsd:string" minOccurs="1" />
              <xsd:element name="LängsterStrom" type="xsd:string" minOccurs="0" />
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

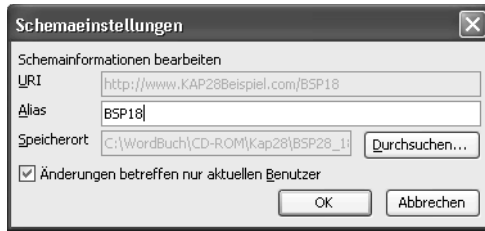


Die Beispieldatei *BSP28\_18.xsd* finden Sie auf der CD-ROM im Ordner *\Beispiele\Kap28*.

Um ein Schema in die Bibliothek aufzunehmen und mit einem XML-Dokument zu verbinden, gehen Sie wie folgt vor.

1. Öffnen Sie das XML-Dokument (in diesem Beispiel, *BSP28\_02.xml* aus Listing 28.2). Klicken Sie rechts neben dem ersten Tag (*Kontinente*).
2. Aktivieren Sie nach Aufruf des Menübefehls *Extras/Vorlagen und Add-Ins* die Registerkarte *XML-Schema*.
3. Klicken Sie auf die Schaltfläche *Schema hinzufügen*, um ein Schema in die Liste aufzunehmen (Abbildung 28.10). Navigieren Sie zum Ordner und wählen Sie die *\*.xsd*-Datei aus.

Abbildg. 28.10 Das Dialogfeld *Schemaeinstellungen*

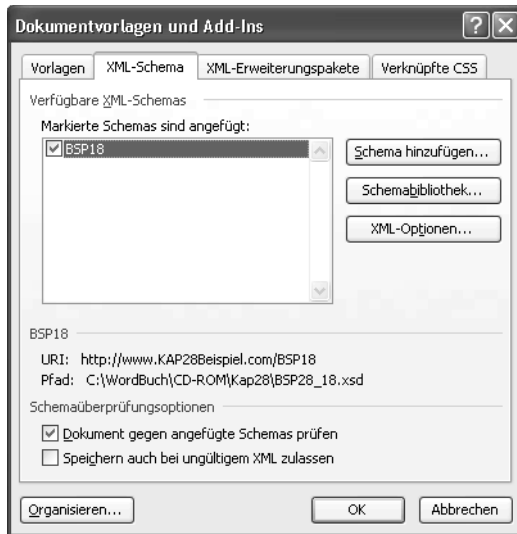


4. Bei der Aufforderung für ein »Alias« (Namensraumpräfix) geben Sie einen kurzen, beschreibenden Ausdruck ein.

**HINWEIS** Word wird beim Speichern des XML-Dokuments das eingegebene Namensraumpräfix nicht berücksichtigen. Es dient nur der Anzeige im Aufgabenbereich *XML-Struktur*.

5. Das Namensraumpräfix wird der Schemabibliothek (Abbildung 28.11) hinzugefügt. Informationen über den URI und den Pfad zum Schema erscheinen darunter.
6. Um das Schema mit dem aktuellen Dokument zu verbinden, aktivieren Sie das Kontrollkästchen neben dem Namensraumpräfix und bestätigen dann mit *OK*.

Abbildg. 28.11 Schemas werden auf der Registerkarte *XML-Schema* mit einem Dokument verbunden.



**HINWEIS** Die eigentliche Schemabibliothek, in der Schemas und Lösungen verwaltet werden, wird durch Anklicken der Schaltfläche *Schemabibliothek* erreicht (Abbildung 28.18). In der oberen Hälfte können Schemas hinzugefügt, entfernt sowie andere Namensraumpräfixe zugewiesen werden.

Beim Betrachten des Aufgabenbereichs *XML-Struktur* wäre zu erwarten, dass der Eintrag *Kontinent* in der unteren Liste steht, da laut Schema nach Kontinente das einzig erlaubte Element Kontinent ist. Dies ist jedoch nicht der Fall. Beim Deaktivieren des Kontrollkästchens *Nur untergeordnete Elemente des aktuellen Elements auflisten* wird der Grund klar, wie in Abbildung 28.12 ersichtlich.

Die Liste führt nämlich zwei Einträge für jedes Element. Die im »KML«-Dokument bereits vorhandenen stellen die Einträge ohne »{BSP18}« dar. Sie befinden sich in einem anderen Namensraum als das Schema (kein Namensraum ist in *BSP28\_02.xml* deklariert).

Das Verbinden eines Schemas mit einem XML-Dokument ändert den Namensraum bestehender Elemente *nicht*. Falls den vorhandenen Elementen ein Namensraum zugewiesen wurde, und Sie genau diesen als *Alias* für das Schema festlegen, wird Word es mit diesen Elementen verbinden. Danach kann der *Alias*-Eintrag geändert werden (beispielsweise von <http://www.KAP28Beispiel.com/BSP18> in *Bsp18*).

Abbildg. 28.12 Liste der verfügbaren Elemente im Aufgabenbereich *XML-Struktur*



Es ist auch möglich, durch ein Makro wie in Listing 28.20 vorhandenen Elementen eines XML-Dokuments den Namensraum eines Schemas zuzuweisen. Es schleift durch alle Element-Knotenpunkte des aktuellen Dokuments, prüft den Namensraum und fügt, wenn dieser nicht dem erwünschten (strURI) entspricht, ein Tag mit dem korrekten Namensraum ein. Anschließend werden die nicht erwünschten Knotenpunkte (alten Tags) gelöscht.

Listing 28.20 Den Namensraum *URI* in einem Dokument ersetzen

```
Sub NamensraumAnpassen()
    Const strURI = "http://www.KAP28Beispiel.com/BSP18"
    Dim objXMLNode As Word.XMLNode

    For Each objXMLNode In ActiveDocument.XMLNodes
        If objXMLNode.NamespaceURI <> strURI Then
            ActiveDocument.XMLNodes.Add Name:=objXMLNode.BaseName, _
                Namespace:=strURI, Range:=objXMLNode.Range
        End If
    Next
    For Each objXMLNode In ActiveDocument.XMLNodes
        If objXMLNode.NamespaceURI <> strURI Then
            objXMLNode.Delete
        End If
    Next
End Sub
```



Die Beispieldatei *BSP28\_19.doc* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap28*.

Wenn Sie nun das XML-Dokument unter einem anderen Namen speichern (beispielsweise *BSP28\_02a.xml*), schließen und dann im Windows-Editor öffnen, erkennen Sie, dass der Namensraum URI *http://www.KAP28Beispiel.com/BSP18* dem Wurzelement *Kontinente* hinzugefügt wurde. Wird die Datei wieder in Word geöffnet, führt die Liste im Aufgabenbereich *XML-Struktur* nur einen Satz Elemente auf, und ein Blick in *Extras/Vorlagen und Add-Ins/XML-Schema* bestätigt, dass Word das Schema mit dem XML-Dokument verbunden hat.

**PROFITIPP**

Steht Ihnen kein Schema zur Verfügung, gibt es einige Werkzeuge, die ein für ein XML-Dokument passendes automatisch erstellen. Innerhalb von Word können Sie beispielsweise die »Word XML Toolbox« benutzen. Es handelt sich um ein im .NET Framework 1.1 geschriebenes Add-In, das auf der Microsoft Webseite bei <http://www.microsoft.com/downloads/details.aspx?FamilyID=a56446b0-2c64-4723-b282-8859c8120db6> herunter geladen werden kann.

Eine Beschreibung der »Toolbox« (in englischer Sprache) befindet sich bei [http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnofftalk/html/ODC\\_office01012004.asp](http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnofftalk/html/ODC_office01012004.asp).

Die »Toolbox« enthält u.a. ein Werkzeug, das ein Schema von einem in Word geöffneten XML-Dokument ableitet (»Inferred Schema«). Es fordert den Anwender auf, einen Namensraum URI sowie einen Dateinamen für das Schema einzugeben. Das Schema wird der Schemabibliothek hinzugefügt; der Inhalt des XML-Dokuments wird in die neue Datei geschrieben, umgeben von einem neuen Wurzelement, das den Namensraum deklariert. Die Datei wird anschließend in Word geöffnet.

Das Schema *BSP28\_20.xsd* wurde mit diesem Werkzeug für das XML-Dokument *BSP28\_02.xml* erstellt. (Es handelt sich um die einzige Beispieldatei dieses Kapitels, die eine UTF-16-Codierung hat.) *BSP28\_21.xml* ist die passende, von der Toolbox erstellte XML-Datei; die WordProcessingML-Version finden Sie in der Datei *BSP28\_22.xml* (alle befinden sich im Ordner *\Beispiele\Kap28*).

Der Aufgabenbereich *XML-Struktur* weist auch auf Probleme mit der Gültigkeit eines XML-Dokuments hin. Steht beispielsweise ein Element am falschen Ort, oder entspricht der Inhalt nicht den im Schema ausgelegten Regeln, erscheint ein gelber Rhombus. Wird dieser mit der rechten Maustaste angeklickt, zeigt das Kontextmenü die entsprechende Fehlermeldung an (Abbildung 28.13).

Standardmäßig kann ein ungültiges XML-Dokument nicht gespeichert werden. Wollen Sie es trotzdem speichern, muss entweder

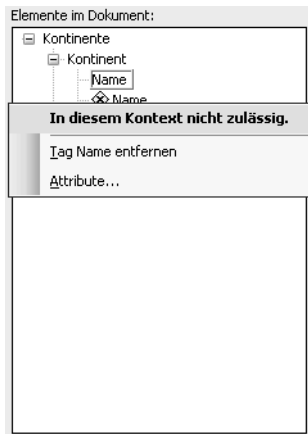
- die Verbindung zum Schema getrennt werden,
- in *Extras/Vorlagen und Add-Ins/XML-Schema* das Kontrollkästchen *Speichern auch bei ungültigem XML zulassen* aktiviert werden oder
- das XML-Dokument als Word-Dokument (\*.doc) gespeichert werden.

**HINWEIS**

Unter Umständen kann Word ein nicht wohlgeformtes XML-Dokument nicht öffnen. In diesem Fall müssen Sie die Fehler zuerst in einer anderen Umgebung (wie dem Windows-Editor) beheben.



Abbildg. 28.13 Kontextfehler, weil ein »Name«-Element nicht in einem »Name« verschachtelt sein darf



## Transformationen und Lösungen (Solutions)

Beim Öffnen oder Speichern eines XML-Dokuments – sowohl eines im Format WordProcessingML wie auch »Nur Daten« – als XML-Dokument kann der Inhalt gleichzeitig transformiert werden. Das geöffnete Dokument sieht damit möglicherweise ganz anders aus als die ursprüngliche Datei, bzw. das gespeicherte Ergebnis weicht eventuell bedeutend von der Version auf dem Bildschirm ab.

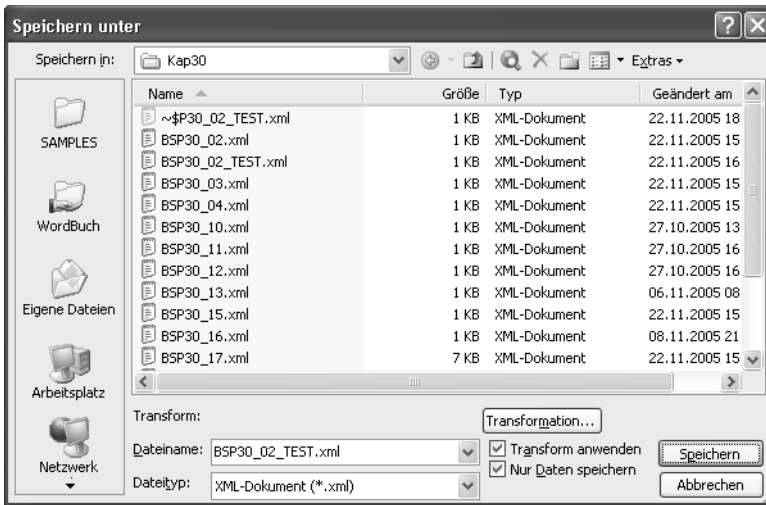
Ein Word-Dokument kann beispielsweise beim Speichern in das XSL-FO-Format transformiert werden, indem Sie

1. über den Menübefehl *Datei/Speichern unter* das zugehörige Dialogfeld einblenden,
2. als Dateiformat *XML-Dokument* wählen,
3. die Kontrollkästchen *Nur Daten speichern* sowie *Transform anwenden* aktivieren,
4. dann die Schaltfläche *Transformation* anklicken (Abbildung 28.14),
5. zur XSL-FO-Transformation (siehe den Abschnitt »XSLT« in diesem Kapitel) navigieren und sie anwählen.

### WICHTIG

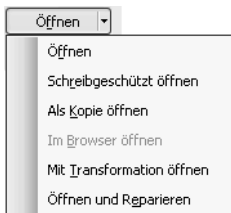
Um Datenverluste zu vermeiden, speichern Sie das Dokument als Word-Dokument ab, bevor Sie eine Transformation durchführen.

Abbildg. 28.14 Im Dialogfeld *Speichern unter* kann ein Dokument als XML gespeichert und gleichzeitig transformiert werden.



Um ein WordProcessingML- oder anderes XML-Dokument beim Öffnen zu transformieren, klicken Sie auf den Pfeil neben der Schaltfläche *Öffnen* im Dialogfeld zum Menübefehl *Datei/Öffnen* und wählen den Eintrag *Mit Transformation öffnen* (Abbildung 28.15). (Die Transformation eines WordProcessingML-Dokuments in ein anderes WordProcessingML-Dokument ist normalerweise recht komplex.)

Abbildg. 28.15 Ein XML-Dokument mit einer Transformation öffnen



Es ist auch möglich, das Aussehen eines XML-Dokuments zu ändern, solange es noch nicht bearbeitet wurde. (Sobald mit der Bearbeitung begonnen wird, betrachtet Word es nicht mehr als XML-, sondern als gewöhnliches Word-Dokument.) Der Aufgabenbereich *XML-Dokument* stellt neben der standardmäßigen Transformation *Nur Daten* auch den Eintrag *Durchsuchen* zur Verfügung, um nachträglich eine gewünschte Transformation auszuwählen:

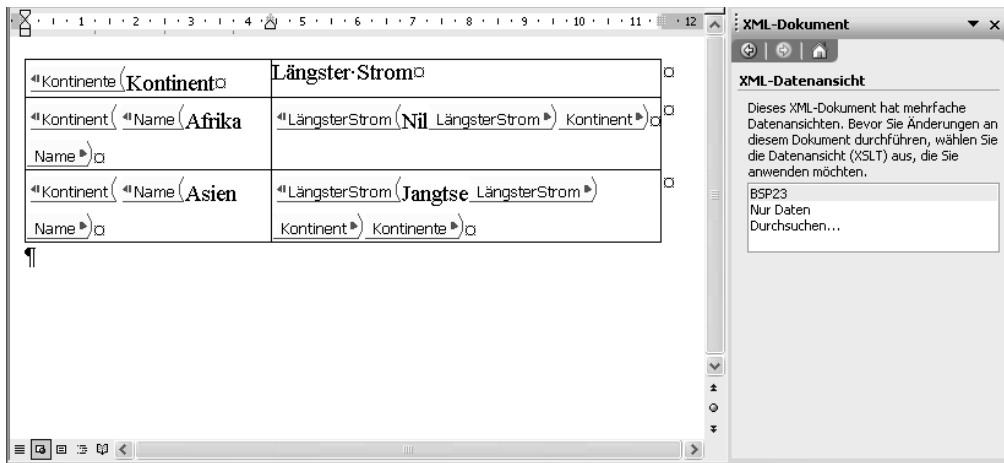
1. Öffnen Sie das XML-Dokument, das im Abschnitt »XML-Vokabulare« in diesem Kapitel vorgestellt wurde (*BSP28\_02.xml*).
2. Im Aufgabenbereich *XML-Dokument* klicken Sie auf den Eintrag *Durchsuchen*.
3. Navigieren Sie zum Ordner mit den CD-Dateien und wählen Sie eine Transformation (*BSP28\_03.xsl* beispielsweise), die im Abschnitt »XML-Daten transformieren« beschrieben wurde.

Sie sollten eine ähnliche Tabelle wie in Abbildung 28.1 sehen. Wenn Sie die gleiche Transformation mit dem im Abschnitt »Die Word-Schemabibliothek« weiter vorne in diesem Kapitel erstellten Dokument (*BSP28\_02a.xml*) ausprobieren, erscheint nur die erste Zeile der Tabelle. Der Grund dafür ist, dass die Transformation den Namensraum URI nicht deklariert und so die Knotenpunkte mit den Daten nicht findet.

Das Listing 28.21 veranschaulicht eine Transformation, die dieses XML-Dokument in ein WordProcessingML-Dokument transformiert, wie in Abbildung 28.16. Sie deklariert sowohl den Namensraum des *BSP28\_18.xsd*-Schemas als auch den für Words Vokabular:

```
xmlns:w="http://schemas.microsoft.com/office/word/2003/wordml"
xmlns:ns2="http://www.KAP28Beispiel.com/BSP18"
```

Abbildg. 28.16 Ein in WordProcessingML transformiertes XML-Dokument



Listing 28.21 Transformation, um aus einem »KML«- ein WordProcessingML-Dokument zu erzeugen

```
<?xml version="1.0" encoding="utf-8" ?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:w="http://schemas.microsoft.com/office/word/2003/wordml"
  xmlns:ns2="http://www.KAP28Beispiel.com/BSP18">
  <xsl:output method="xml" encoding="UTF-8" standalone="yes" />
  <xsl:template match="/">
    <w:wordDocument
      xmlns:w="http://schemas.microsoft.com/office/word/2003/wordml"
      xmlns:ns2="http://www.KAP28Beispiel.com/BSP18"
      xml:space="preserve">
      <w:body>
        <xsl:apply-templates select="ns2:Kontinente" />
      </w:body>
    </w:wordDocument>
  </xsl:template>

  <xsl:template match="/ns2:Kontinente">
    <ns2:Kontinente>
```

**Listing 28.21** Transformation, um aus einem »KML«- ein WordProcessingML-Dokument zu erzeugen (Fortsetzung)

```

<w:tbl>
  <w:tblPr>
    <w:tblW w:w="6500" w:type="dxa"/>
    <w:tblBorders>
      <w:top w:val="single" w:sz="4"/><w:left w:val="single" w:sz="4"/>
      <w:bottom w:val="single" w:sz="4"/><w:right w:val="single" w:sz="4"/>
      <w:insideH w:val="single" w:sz="6"/><w:insideV w:val="single" w:sz="6"/>
    </w:tblBorders>
    <w:tblCellMar>
      <w:left w:w="10" w:type="dxa"/><w:right w:w="10" w:type="dxa"/>
    </w:tblCellMar>
  </w:tblPr>
  <w:tblGrid>
    <w:gridCol w:w="2500"/><w:gridCol w:w="4000"/>
  </w:tblGrid>
  <w:tr>
    <w:tc><w:p><w:r><w:t>Kontinent</w:t></w:r></w:p></w:tc>
    <w:tc><w:p><w:r><w:t>Längster Strom</w:t></w:r></w:p></w:tc>
  </w:tr>
  <xsl:apply-templates select="ns2:Kontinent" />
</w:tbl>
</ns2:Kontinente>
<w:p />
</xsl:template>

<xsl:template match="/ns2:Kontinente/ns2:Kontinent" >
  <ns2:Kontinent>
    <w:tr>
      <ns2:Name>
        <w:tc><w:p><w:r><w:t>
          <xsl:apply-templates select="ns2:Name" />
        </w:t></w:r></w:p></w:tc>
      </ns2:Name>
      <ns2:LängsterStrom>
        <w:tc><w:p><w:r><w:t>
          <xsl:apply-templates select="ns2:LängsterStrom" />
        </w:t></w:r></w:p></w:tc>
      </ns2:LängsterStrom>
    </w:tr>
  </ns2:Kontinent>
</xsl:template>

<xsl:template match="/ns2:Kontinente/ns2:Kontinent/ns2:Name" >
  <xsl:apply-templates />
</xsl:template>

<xsl:template match="/ns2:Kontinente/ns2:Kontinent/ns2:LängsterStrom" >
  <xsl:apply-templates />
</xsl:template>

</xsl:stylesheet>

```



Die Beispieldatei *BSP28\_23.xsl* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap28*.

Folgende Punkte sind noch zu beachten.

- Nicht nur das XSL-Element deklariert die oben erwähnten Namensräume, sie müssen auch im Wurzelement des künftigen WordProcessingML-Dokuments vorkommen:

```
<w:wordDocument
  xmlns:w="http://schemas.microsoft.com/office/word/2003/wordml"
  xmlns:ns2="http://www.KAP28Beispiel.com/BSP18"
  xml:space="preserve">
```

- Die benutzerdefinierten (»KML«) Elemente werden nicht irgendwie in WordProcessingML-Elemente verschachtelt.
- Diese benutzerdefinierten Tags stehen außerhalb der Tabellenstrukturen. Kontinent umgibt beispielsweise die Tabellenzeile und Name die Tabellenzelle.

```
<ns2:Kontinent>
<w:tr>
  <ns2:Name>
    <w:tc><w:p><w:r><w:t>
      <xsl:apply-templates select="ns2:Name" />
    </w:t></w:r></w:p></w:tc>
  </ns2:Name>
```

#### TIPP

Stehen Elemente im beschriebenen Verhältnis zur Tabellenstruktur, werden die Tags für jede in der Word-Benutzerschnittstelle hinzugefügte Tabellenzeile automatisch hinzugefügt. Auf diese Weise kann Word als einfacher XML-Editor für die Dateneingabe eingesetzt werden. Um dies zu testen, führen Sie die Transformation wie oben beschrieben aus, klicken Sie in die letzte Tabellenzelle und drücken Sie Tab. Word müsste eine neue Tabellenzeile generieren, die die Elemente für einen Kontinent enthält.

Um dem Anwender die Auswahl und die Zuweisung einer Transformation zu erleichtern, können Transformationen mit Schemas verbunden werden. Diese werden im Abschnitt *Solution zum Schema* der Schemabibliothek verwaltet. Die mit einem Schema assoziierten Transformationen erscheinen automatisch in der Liste *Datenansicht* des Aufgabenbereichs, wenn ein XML-Dokument mit dem passenden Namensraum geöffnet wird. Eine dieser Transformationen wird als Standard festgelegt und (statt Words standardmäßigem »Nur Daten«-Eintrag) dem gerade geöffneten XML-Dokument zugewiesen.

Eine Transformation darf mit mehr als einem Schema verbunden werden.

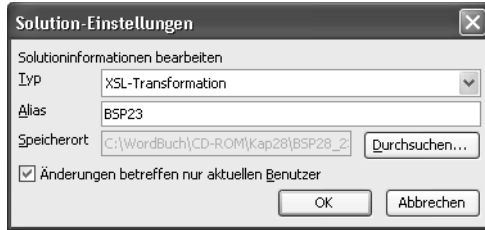
Fügen Sie die Transformation *BSP28\_23.xsl* wie folgt der Schemabibliothek hinzu:

1. Blenden Sie das Dialogfeld *Schemabibliothek* über die Menüfolge *Extras/Vorlagen und Add-Ins/XML-Schema* ein.
2. Wählen Sie das Schema (*Bsp18*) aus der Liste.
3. Klicken Sie die Schaltfläche *Solution hinzufügen* an.

4. Navigieren Sie zum Beispiel-Ordner, wählen Sie die Transformation (*BSP28\_23.xsl*) und betätigen Sie dann *Öffnen*. Das Dialogfeld *Solution Einstellungen* (Abbildung 28.17) wird eingeblendet.
5. Geben Sie ein Namensraumpräfix (»Alias«) in das *Alias*-Feld ein. (Falls nicht, erstellt Word eine GUID und benutzt diese.)

Abbildg. 28.17

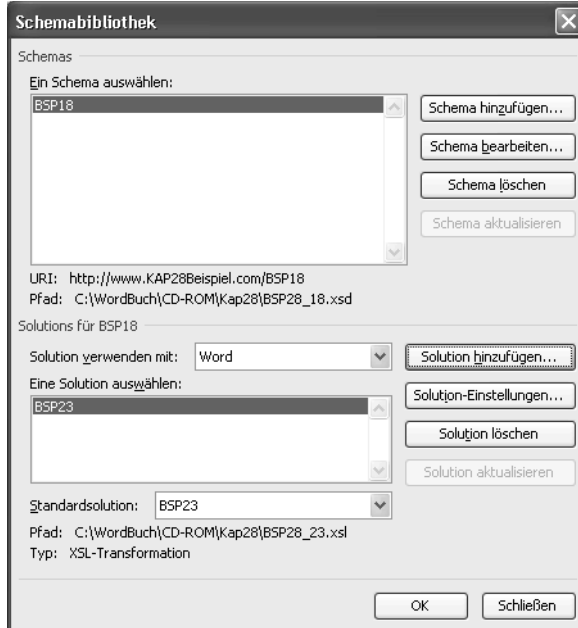
Beim Einbinden einer Transformation in die Schemabibliothek ein Namensraumpräfix zuweisen



Beachten Sie in Abbildung 28.18, wie Word diese Transformation als *Standardsolution* festlegt. Ab diesem Zeitpunkt ist es nicht mehr möglich, Words interne »Nur Daten«-Transformation als die standardmäßige festzulegen. Zudem werden beim Öffnen eines XML-Dokuments mit dem entsprechenden Namensraum die XML-Tags automatisch ausgeblendet.

Abbildg. 28.18

Eine Solution (Transformation) mit einem Schema verbinden und verwalten im Dialogfeld *Schemabibliothek*



## PROFITIPP

XML-Transformationen zu schreiben, ist eine komplexe Angelegenheit. Solche zu schreiben, die WordProcessingML-Dokumente mit Elementen eines benutzerdefinierten Vokabulars erstellen, ist um Einiges schwieriger. Zum Glück enthält das Microsoft Office Word 2003 XML SDK ein Werkzeug, das hilft, WordProcessingML-Transformationen zu erstellen: das »WordProcessingML XSLT Inference Tool«.

Um dieses zu benutzen, gehen Sie wie folgt vor:

1. Laden Sie das Microsoft Office Word 2003 XML SDK herunter und installieren Sie es.
2. Suchen Sie über *Start/Alle Programme* das Tool über die Befehlsfolge *Microsoft Office 2003 Developer Resources/Microsoft Word 2003 XML SDK/Tools/Install the WordProcessingML Transform Inference Tool*.

Erstellen Sie eine Transformation, indem Sie

1. ein XML-Dokument (wie *BSP28\_02a.xml*; also *kein* WordProcessingML) in Word öffnen, das den gleichen Namensraum wie ein in der Schemabibliothek vorhandenes Schema hat,
2. das Dokument formatieren, so dass es wie das transformierte Ergebnis aussieht,
3. die Datei als WordProcessingML speichern (*BSP28\_24.xml* beispielsweise) und schließen,
4. das Fenster der Eingabeaufforderung über die Befehlsfolge *Start/Alle Programme/Zubehör* einblenden
5. und das »Inference Tool« auf die Datei (mit voller Pfadangabe) ausführen.

```
wml2xslt c:\\WordBuch\\Beispiele\\KAP28\\BSP28_24.xml
```

Das Resultat wird eine XSL-Datei sein (im beschriebenen Fall *BSP28\_24.xsl*). Bitte beachten Sie, dass dieses Werkzeug eher als Hilfsmittel zu betrachten ist. Oft werden Sie nicht herumkommen, die Transformation händisch anzupassen, um genau das erwünschte Ergebnis zu erzielen.

## Schemas und Transformationen im Word-Objektmodell

Der Bezug zwischen dem Word-Objektmodell und der Schemabibliothek ist klar erkennbar. Die `Application.XMLNamespaces`-Auflistung stellt den Inhalt (die Namensräume) der Schemabibliothek dar. Mit der `Add`-Methode können weitere Namensräume hinzugefügt werden.

Jedes `XMLNamespace`-Objekt stellt ein Schema der Bibliothek dar. Die Informationen für den Namensraum URI (`URI`), das Namensraumpräfix (`Alias`), der Speicherort des Schemas (`Location`) und die standardmäßige Solution (`DefaultTransform`) werden mit entsprechenden Eigenschaften verwaltet. Das Objekt hat zudem Methoden, um einen Namensraum zu entfernen (`Delete`) und ihn mit einem Dokument zu verbinden (`AttachToDocument`).

Transformationen (`Solutions`) werden durch die Auflistung `XSLTransforms` eines `XMLNamespace`-Objekts dargestellt. Mittels deren `Add`-Methode können weitere Transformationen hinzugefügt werden.

Jedes `XSLTransform`-Objekt stellt eine mit dem Schema verbundene Transformation (`Solution`) dar. Bemerkenswerte Eigenschaften sind `Alias` (Namensraumpräfix) sowie `Location` (Speicherort). Auch diese Objekte verfügen über eine `Delete`-Methode.

Das folgende Codeschnipsel lädt das Beispiel-Schema mit dem Namensraum BSP18 in die Schema-bibliothek:

```
Dim objSchema As Word.XMLNamespace
Set objSchema = Application.XMLNamespaces.Add(
    Path:="C:\CD-ROM\Beispiele\Kap28\BSP28_18.xsd", NamespaceURI="", _
    Alias:="BSP18", InstallForAllUsers:=False)
```

## WordProcessingML außerhalb von Word generieren

Im Abschnitt »Transformationen und Lösungen (Solutions)« weiter vorne in diesem Kapitel wurde gezeigt, wie ein XML-Dokument in ein WordProcessingML-Dokument transformiert wird, indem eine Transformation (XSLT) ausgeführt wurde. Word hatte mit der Transformation eigentlich gar nicht zu tun. Es diente lediglich als Behälter und zeigte das Ergebnis an.

Eigentlich muss Word gar nicht vorhanden sein, um ein XML-Dokument in ein WordProcessingML-Dokument zu transformieren. Dazu braucht man lediglich ein XML-Dokument, eine XSLT-Datei sowie einen Mechanismus, um die Transformation auszuführen. Demzufolge kann die Transformation unabhängig von Word durch einen Browser oder andere Software durchgeführt werden. Ja, sie kann sogar auf einem Server stattfinden.

Als Beispiel zeigen wir, wie die Transformation programmtechnisch mit dem DOM des MSXML-Parsers bewerkstelligt wird. Das Listing 28.22 veranschaulicht den Code eines VBA-Projekts (in Excel oder PowerPoint beispielsweise). Sie müssen im Visual Basic-Editor über den Menübefehl *Extras/Verweise* einen Verweis zur Bibliothek »Microsoft XML, v. 5.0« festlegen. Nach der Ausführung der Prozedur öffnen Sie die erstellte Datei *BSP28\_25.xml*. Sie müsste ebenso aussehen wie das Ergebnis der Transformation im Abschnitt »Transformationen und Lösungen (Solutions)«.

### HINWEIS

Sie können diesen Code auch in einem Visual Basic-Projekt benutzen. Um ihn zu testen, erstellen Sie ein *Standard.exe*-Projekt. Fügen Sie dem Formular eine Schaltfläche hinzu und geben Sie die Codezeilen in deren Ereignisprozedur ein. Vergessen Sie nicht, einen Verweis zur Microsoft XML-Bibliothek zu setzen.

Abbildg. 28.19 XML in WordProcessingML transformieren – ohne Word

```
Sub XML2WML()
    'Passen Sie die Pfadangaben Ihrem System an
    Const strPath = "C:\CD-ROM\Beispiele\Kap28\"
    Dim objXSLTransform As MSXML2.DOMDocument50
    Dim objXMLDocument As MSXML2.DOMDocument50
    Dim objWordMLDocument As MSXML2.DOMDocument50

    Set objXSLTransform = New MSXML2.DOMDocument50
    Set objXMLDocument = New MSXML2.DOMDocument50
    Set objWordMLDocument = New MSXML2.DOMDocument50

    objXSLTransform.async = False
    If Not objXSLTransform.Load(strPath & "\BSP28_23.xsl") Then
        MsgBox "XSL-Transformationsdatei konnte nicht gefunden werden"

```



Abbildg. 28.19 XML in WordProcessingML transformieren – ohne Word (Fortsetzung)

```

Else
  If objXSLTransform.parseError.errorCode <> 0 Then
    MsgBox "Parse error in XSL-Transform: " & objXSLTransform.parseError.reason
  Else
    objXMLDocument.async = False
    If Not objXMLDocument.Load(strPath & "\BSP28_02a.xml") Then
      MsgBox "XML-Datei konnte nicht gefunden werden"
    Else
      If objXMLDocument.parseError.errorCode <> 0 Then
        MsgBox "Parse error in XML-Datei: " & objXMLDocument.parseError.reason
      Else
        objXMLDocument.transformNodeToObject objXSLTransform, objWordMLDocument
        objWordMLDocument.Save strPath & "\BSP28_26.xml"
      End If
    End If
  End If
End If

Set objWordMLDocument = Nothing
Set objXMLDocument = Nothing
Set objXSLTransform = Nothing
End Sub

```



Die Beispieldatei *BSP28\_25.doc* sowie *BSP28\_25.txt* mit dem Code finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap28*.

## Zusammenfassung

In diesem Kapitel wurden die folgenden Themen behandelt:

- XML- und XSL-Standards sowie Zweck und Nutzen vom XML wurden im Abschnitt »Was ist XML und wozu dient es?« (Seiten 862 ff.) vorgestellt.
- Eine Übersicht der XML-Funktionalität in Word 2003 folgte im Abschnitt »Welche Aufgabe erfüllt XML in Word?« (Seiten 868 ff.).
- Die von Word unterstützten Mitglieder der XML-Familie, wie XML, Schemas, Namensräume, Document Object Model (DOM) und Transformationen (XSLT) wurden im Abschnitt »XML-Bestandteile« (Seiten 869 ff.) kurz erläutert.
- Die Grundzüge von WordProcessingML und Words eigenem XML-Vokabular wurden in Abschnitt WordProcessingML (Seiten 890 ff.) präsentiert.
- Wie Schemas und Transformationen in Word integriert werden wurde im Abschnitt »Die Word-Schemabibliothek« (Seiten 896 ff.) behandelt.
- Schließlich wurde im Abschnitt »WordProcessingML außerhalb von Word generieren« (Seiten 908 ff.) veranschaulicht, wie ein WordProcessingML-Dokument unabhängig von der Word-Anwendung erstellt werden kann.



## Kapitel 29

# Smarttags

**In diesem Kapitel:**

Die Arbeit mit Smarttags	912
Die Entwicklung von Smarttags	916
Zusammenfassung	941

Ein »Smarttag« ist ein Mechanismus, um vordefinierte Zeichenfolgen im Text zu erkennen und dem Benutzer dafür bestimmte, sinnvolle »Aktionen« (in Form von Menüeinträgen) anzubieten. Dieses Kapitel stellt die Smarttags vor, mit Schwerpunkt auf der Funktionalität in Word 2003. Zudem wird anhand von Beispielen gezeigt, wie Sie Smarttags für Word 2003 erstellen können.

## Die Arbeit mit Smarttags

In Office XP standen Smarttags Word-Dokumenten, Excel-Arbeitsmappen und Outlook (in begrenztem Umfang) zur Verfügung. In Office 2003 wurde die Unterstützung auf PowerPoint, Access und den Aufgabenbereich *Recherchieren* erweitert.

Es gibt zwei Arten von Smarttags. Die eine Art basiert auf einer XML-Datei – eine »Smart Tag List« (oder MOSTL). Die andere muss in einer DLL programmiert werden, was die Funktionalität effektiver und flexibler macht.

## Smarttags aus dem Blinkwinkel des Benutzers

Falls Ihnen Smarttags völlig unbekannt sind, führt dieser Abschnitt durch die Installation und Anwendung eines einfachen Beispiels, das auf einer Smarttag-Liste basiert. (Smarttag-Listen werden in Abschnitt »MOSTL-Smarttags entwickeln« in diesem Kapitel eingehender vorgestellt.)

Das in Listing 29.1 vorgestellte XML-Dokument enthält die Information für das Smarttag-Beispiel. Speichern Sie es in dem Ordner

*C:\Programme\Gemeinsame Dateien\Microsoft Shared\Smart Tag\LISTS.*

### HINWEIS

Informationen zum Erstellen und Umgang mit XML-Dokumenten finden Sie in Kapitel 28.

**Listing 29.1** Eine einfache Smarttag-Liste

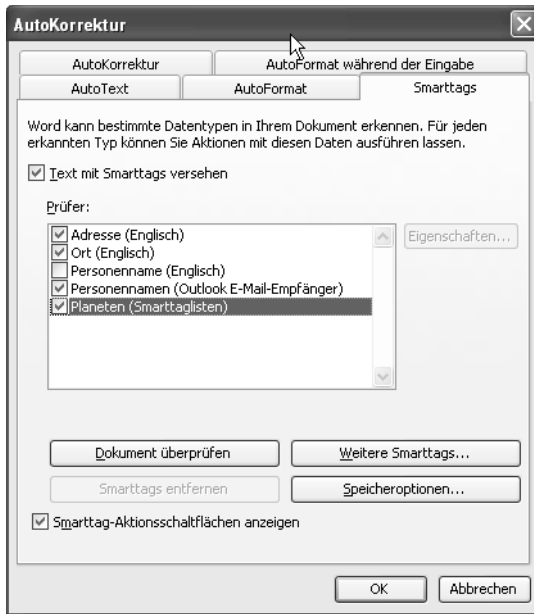
```
<?xml version="1.0" encoding="UTF-8" ?>
<FL:SmartTaglist xmlns:FL="http://schemas.microsoft.com/office/Smart Tags/2003/mostl">
  <FL:name>Planeten</FL:name>
  <FL:lcid>1031,0</FL:lcid>
  <FL:SmartTag type="urn:schemas-beispiel:astronomisch#planet">
    <FL:caption>Planeten</FL:caption>
    <FL:terms>
      <FL:termlist>merkur, venus, erde, mars, jupiter, saturn, uranus, neptun,
        pluto</FL:termlist>
    </FL:terms>
    <FL:actions>
      <FL:action id="wikiinfo">
        <FL:caption>wiki Artikel</FL:caption>
        <FL:url>http://de.wikipedia.org/wiki/{TEXT}_%28Planet%29</FL:url>
      </FL:action>
    </FL:actions>
  </FL:SmartTag>
</FL:SmartTaglist>
```



Sie dürfen die Beispieldatei *planet.xml* auf der CD-ROM im Ordner *\Beispiele\Kap29* benutzen.

Starten Sie Word und stellen Sie sicher, dass ein neues, leeres Dokument geöffnet ist. Rufen Sie dann den Menübefehl *Extras/AutoKorrektur-Optionen* auf und holen Sie die Registerkarte *Smarttags* in den Vordergrund (Abbildung 29.1). Aktivieren Sie den Eintrag »Planeten (Smarttaglisten)« in der Liste *Prüfer* und klicken Sie auf *OK*.

Abbildg. 29.1 Das Dialogfeld, um Smarttag-Optionen zu verwalten



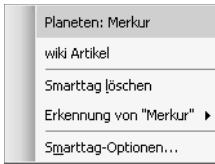
Im Dokument geben Sie den Text »Merkur« ein, gefolgt von einem Leerzeichen. Nach einem kurzen Augenblick wird das Wort von einer feinen, gepunkteten Linie unterstrichen. Wenn Sie den Mauszeiger über dem Wort positionieren oder hineinklicken, erscheint ein Smarttag-Symbol, wie in Abbildung 29.2 ersichtlich.

Abbildg. 29.2 Text, der von einem Smarttag erkannt und markiert wurde



Ein Klick auf das Symbol (oder Drücken der Tastenkombination  $\text{Alt} + \text{Shift} + \text{F10}$ ) blendet die Liste der Aktionen und weitere Optionen für das Smarttag ein (Abbildung 29.3).

Abbildg. 29.3 Ein Smarttag-Kontextmenü



Der erste Eintrag dieses Menüs ist eine benutzerfreundliche Bezeichnung des erkannten Smarttags. Darunter befindet sich eine Liste passender »Aktionen«. Im Beispiel enthält das Menü nur einen solchen Eintrag: »wiki Artikel«. Klicken Sie diesen an und es wird eine Webseite mit Informationen über den Planeten Merkur im Browser eingeblendet – vorausgesetzt, der Rechner ist mit dem Internet verbunden.

## Smarttag »Recognizers« und »Actions«

Jede Textfolge in einem Dokument, die durch ein Smarttag markiert wird, ist eine Instanz eines Smarttag-Objektes eines bestimmten Smarttag-Typs. Im vorangehenden Beispiel ist der Smarttag-Typ *urn:schemas-beispiel:astronomisch#planet*.

Smarttag-Objekt-Instanzen werden von der Word-Anwendung erstellt, wenn sie registrierte »Recognizers« (Prüfer) anwendet, um bestimmte Ausdrücke oder Zeichenfolgen im Dokumenttext zu suchen. Die Hauptaufgabe eines »Recognizers« besteht darin, einen Smarttag-Typ mit jeder Zeichenfolge zu verbinden, die er erkennt. Mehr als ein »Recognizer« kann die gleiche Textstelle kennzeichnen. Es ist sogar möglich, obwohl etwas verwirrend, dass ein »Recognizer« die gleiche Textstelle mehr als einmal markiert.

Wie im letzten Abschnitt erklärt, besteht ein Smarttag-Menü aus folgenden Teilen:

- Eine Überschrift, die den Namen und den erkannten Text anzeigt.
- Ein Eintrag für jede mit dem Smarttag-Typ verbundene Aktion.
- Einige allgemeine Optionen.

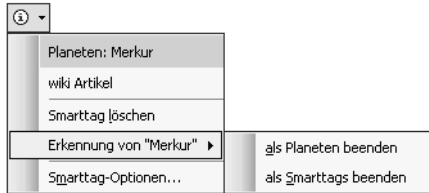
Mehr als ein Satz Aktionen kann mit einem Smarttag-Typ verbunden sein. Würden Sie beispielsweise *planet.xml* kopieren, umbenennen und in den gleichen Ordner mit der ursprünglichen Datei speichern, enthielte das Menü zwei identische Aktionen-Einträge: für jede der XML-Dateien einen.

Zu jeder Instanz eines Smarttag-Objekts gehört ein »Property Bag« (Eigenschaften-Behälter), in dem Namen/Wert-Paare gespeichert werden können. Dieser »Property Bag« kann durch den »Recognizer« oder eine andere Software mit Eigenschaften bestückt werden, die dann den Aktionen zur Verfügung stehen. (Einfache Smarttag-Listen (MOSTL) haben jedoch keine Mechanismen, die es erlauben, dass ihre Werte von Aktionen angesprochen werden.)

## Smarttag-Optionen und Ausnahme-Listen

Die Benutzerschnittstelle bietet dem Anwender die Möglichkeit, eine Textfolge von der Erkennung auszugrenzen. Sie kann entweder nur für einen bestimmten Smarttag-Typ oder gänzlich gesperrt werden, wie in Abbildung 29.4 ersichtlich.

Abbildg. 29.4 Optionen, um eine Textfolge von der Erkennung durch Smarttags auszuschließen



Diese Einstellungen werden in einem XML-Dokument verwaltet, das im Windows Editor bearbeitet werden kann. Es wird unter dem Namen *ignore.xml* im folgenden Ordner gespeichert:

```
<Laufwerk>:\Dokumente und Einstellungen\<Benutzername>\Anwendungsdaten\Microsoft\Smart  
Tags\Exceptions\
```

Im Dialogfeld zum Menübefehl *Extras/Optionen* befinden sich auf der Registerkarte *Speichern* zwei allgemeine Optionen für die Verwaltung von Smarttags:

- *Smarttags einbetten*
- *Smarttags als XML-Eigenschaften in Webseiten speichern*

Smarttags werden standardmäßig im Dokument eingebettet, wenn dieses im Format *\*.doc* oder *WordProcessingML* gespeichert wird. Beim Betrachten einer *WordProcessingML*-Datei in einem Texteditor wird erkennbar, dass Word einen Namensraum für jeden im Dokument benutzten Smarttag-Namensraum erstellt. Zudem enthält es *<st>*-Elemente für jede erkannte Textfolge. (Das RTF-Format unterstützt Smarttags nicht.)

Sind beide der obigen Optionen aktiviert, werden eingebettete Smarttags auch beim Speichern als eine Webseite (*\*.mht*) beibehalten. Der Internet Explorer kann die Liste der Aktionen einblenden, wenn die Sicherheitsoptionen es erlauben.

Beim Einbetten eines Smarttags werden nur der Namensraum sowie Element-Tags gespeichert, nicht aber »Recognizers« oder Aktionen. Diese werden dynamisch nachgeschlagen.

# Die Entwicklung von Smarttags

Es gibt für Smarttags zwei Entwicklungsmethoden. Beide sind im Office 2003 Smart Tag SDK (in englischer Sprache) eingehend beschrieben.

---

**HINWEIS** Sie können das Office Smart Tag SDK von der MSDN-Webseite herunterladen: [http://msdn.microsoft.com/library/default.asp?url=/library/en-us/stagsdk/html/stconWelcomeToSTagSDK\\_HV01074286.asp](http://msdn.microsoft.com/library/default.asp?url=/library/en-us/stagsdk/html/stconWelcomeToSTagSDK_HV01074286.asp). Die Installationsdatei *mstagsdk.msi* befindet sich auch auf der CD-ROM im Ordner *\Beispiele\Kap29*.

---

Das obige Beispiel – eine Microsoft Office Smarttag-Liste (MOSTL = «Microsoft Office Smarttag Liste») – veranschaulicht die erste Methode und ist einfach zu erstellen: Die Angaben werden in eine XML-Datei eingegeben, die anschließend in den dafür bestimmten Ordner unter Windows kopiert wird. Eine sich im Lieferumfang von Office befindende *.dll*-Datei – *MOFL.dll* – sorgt für die Erkennung der Zeichenfolgen, bietet die damit verbundenen, HTTP-basierten Aktionen an und führt sie aus. Die zu erkennenden Zeichenfolgen einer MOSTL können wie folgt definiert werden:

- eine Liste von Ausdrücken (Wörter sowie Zeichengruppierungen)
- eine oder mehrere »Regular Expressions« (Perl-Syntax)
- eine »Context Free Grammar« (CFG)

---

**HINWEIS** Mehr über reguläre Ausdrücke erfahren Sie auf der Webseite [http://de.wikipedia.org/wiki/Perl#Regul.C3.A4re\\_Ausdr.C3.BCcke](http://de.wikipedia.org/wiki/Perl#Regul.C3.A4re_Ausdr.C3.BCcke).

Der Begriff kontextfreie Grammatik wird auf der Seite [http://64.233.183.104/search?q=cache:iQ5T3QcWswkJ:www.bwinf.de/download/183grammatiken.pdf+kontextfreie+Grammatik&hl=de&lr=lang\\_de](http://64.233.183.104/search?q=cache:iQ5T3QcWswkJ:www.bwinf.de/download/183grammatiken.pdf+kontextfreie+Grammatik&hl=de&lr=lang_de) näher erklärt. Informationen zur Syntax für Smarttags finden Sie im Smart Tag SDK.

---

Mehr über MOSTL und ihre Entwicklung lesen Sie im Abschnitt »MOSTL-Smarttags entwickeln« weiter hinten in diesem Kapitel.

Die zweite Methode – eine COM-DLL – bietet erweiterte und flexiblere Funktionalität. Sie ist erforderlich, wenn beispielsweise

- die Kriterien für die Zeichenfolge-Erkennung zu komplex sind für die oben erwähnten Methoden,
- die zu erkennenden Zeichenfolgen in einer Datenbank oder anderen Quelle nachzuschlagen sind oder
- die Aktionen nicht HTTP-basiert sind. (Beispiel: Der erkannte Text soll ersetzt werden.)

Die COM-DLL muss die Smarttag-Interfaces *SmartTagLib.ISmartTagRecognizer* und *SmartTagLib.ISmartTagAction* implementieren. Eine solche *.dll*-Datei kann mit Programmiersprachen wie VB6, VB.NET und C# geschrieben werden, nicht jedoch mit Office-VBA. Das Smarttag SDK enthält einige Beispiele; eines für VB6 finden Sie im Abschnitt »Ein Smarttag mit VB6 entwickeln« in diesem Kapitel.

Zudem ist es möglich, ein Smarttag auf einem vorhandenen zu basieren. Es ist beispielsweise möglich, eine neue MOSTL-Datei zu erstellen, die den gleichen Typ wie eine vorhandene deklariert und andere Aktionen für die »Recognizers« der vorhandenen Liste definiert.



## Unterschiede zwischen den Office-Versionen

Wie schon erwähnt, wurden Smarttags in Office XP eingeführt. Ihre Funktionalität wurde in Office 2003 weiter ausgebaut:

- Die Anzahl der Menüeinträge für Aktionen und ihre Beschriftungen können dynamisch geändert und angepasst werden.
- Die Menüeinträge für Aktionen können auch mit Untermenüs dargestellt werden (*cascading menu*).
- Jede MOSTL-Liste wird separat im Dialogfeld *Smarttag-Optionen* aufgelistet und kann vom Anwender gesperrt bzw. zugelassen werden. In Office XP sind alle MOSTL-Listen in einem einzigen Eintrag enthalten und der Anwender kann entweder alle oder keine zulassen.
- Ausnahme-Listen wurden eingeführt.

Nicht alle für Office XP entwickelten Smarttags laufen unter Office 2003 und müssen unter Umständen angepasst werden. Dieses Kapitel befasst sich lediglich mit der Entwicklung von Smarttags für Word 2003.

## MOSTL-Smarttags entwickeln

Am Kapitelanfang wurde im Abschnitt »Smarttags aus dem Blinkwinkel des Benutzers« ein einfaches Beispiel einer MOSTL vorgestellt. Es veranschaulicht einen Bruchteil der MOSTL-Funktionalität und führt nur einige der im MOSTL XML-Schema definierten Elemente auf. (In diesem Schema sind sowohl Smart Documents als auch Smarttags definiert; in diesem Kapitel werden nur die Smarttag-relevanten Aspekte behandelt.)

### HINWEIS

Das Schema – *mostlAnnotated.xsd* – ist Teil der Microsoft Office 2003 Smart Tag SDK. Die englischsprachige Dokumentation liegt dem SDK bei; eine deutsche Version finden Sie unter der Webadresse <http://www.microsoft.com/germany/msdn/library/data/xml/XMLSchema-FuerSmartTagListen.msp>.

Das in diesem Abschnitt vorgestellte Beispiel baut auf die am Anfang präsentierte »Planeten«-MOSTL auf und veranschaulicht weitere Funktionalität wie

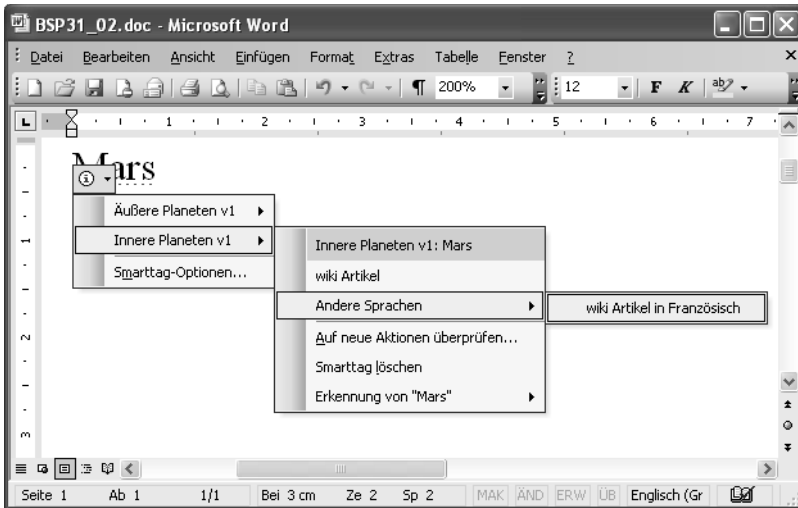
- die automatische Aktualisierung der Aktionen-Einträge
- mehrfache »Recognizers« (Prüfer) und Aktionen
- Untermenüs in Aktionen-Menüs
- verschiedene »Recognizer«-Methoden

Wenn dieser Smarttag aktiv ist, werden Planetennamen wie »Mars« als Smarttag erkannt und markiert. Das Aktionen-Menü wird dem in Abbildung 29.5 ähnlich sein. Das Beispiel umfasst zwei »Recognizers«, einen für »äußere Planeten« sowie einen für »innere Planeten«. (In der Ausgangslage steht der Ausdruck »Mars« absichtlich »irrtümlich« in beiden Listen).

Für jeden »Recognizer« steht ein Menü in der obersten Ebene. Die Einträge für die Aktionen, die in Abbildung 29.3 in der ersten Ebene aufgelistet sind, befinden sich in diesem Fall eine Ebene tiefer. Eine weitere Ebene wurde durch Festlegung einer Aktion als Andere Sprachen//wiki Artikel in Französisch definiert.

Unter-  
menüs

Abbildg. 29.5 Ein Smarttag mit Untermenüs



Automatische  
Aktualisierung

Zudem beinhaltet das Beispiel weitere Dateien, um automatisch das Herunterladen aktualisierter Versionen der Listen auszulösen. Die erste davon wird dem Untermenü *Andere Sprachen* einen weiteren Eintrag hinzufügen. Die zweite korrigiert die Liste der äußeren Planeten (entfernt den Eintrag für »Mars«). Achten Sie beim Laden der Versionen auf die Versionsnummer in den Beschriftungen (v1, v2 bzw. v3).

Der Code der Hauptliste befindet sich in Listing 29.2. Es folgen dann Diskussionen zu den Themen

- wo MOSTL gespeichert werden können,
- Angaben zu Installation und Ausführung des Beispiels,
- weitere Informationen zu den im Beispiel verwendeten Elementen.

Listing 29.2 MOSTL mit zwei »Recognizers« und automatischer Aktualisierung (BSP29\_02.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<FL:smartTagList xmlns:FL="http://schemas.microsoft.com/office/smarttags/2003/mostl">
  <FL:name>Planeten v1</FL:name>
  <FL:caption>Planeten</FL:caption>
  <FL:lcid>0</FL:lcid>
  <FL:description>Planeten unseres Sonnensystem</FL:description>
  <FL:moreInfoURL>http://www.KAP29Bsp.com/planet/BSP29_02_mi.htm</FL:moreInfoURL>
  <FL:downloadURL>http://www.KAP29Bsp.com/planet/BSP29_02_d1.htm</FL:downloadURL>
  <FL:updateable>true</FL:updateable>
  <FL:autoUpdate>true</FL:autoUpdate>
  <FL:lastCheckpoint>1</FL:lastCheckpoint>
  <FL:updateURL>http://www.KAP29Bsp.com/planet/BSP29_02_up_v2.xml</FL:updateURL>
  <FL:updateFrequency>1</FL:updateFrequency>
  <FL:smartTag type="urn:schemas-Bsp:astronomisch#innereplaneten">
    <FL:propertyPage>http://www.KAP29Bsp.com/planet/BSP29_02_ip.htm</FL:propertyPage>
    <FL:caption>Innere Planeten v1</FL:caption>
    <FL:terms>
      <FL:termList>merkur,venus,erde,mars</FL:termList>
    </FL:terms>
  </FL:smartTag>
</FL:smartTagList>
```

Listing 29.2 MOSTL mit zwei »Recognizers« und automatischer Aktualisierung (*BSP29\_02.xml*) (Fortsetzung)

```

</FL:terms>
<FL:actions>
  <FL:action id="wikide">
    <FL:caption>wiki Artikel</FL:caption>
    <FL:url>http://de.wikipedia.org/wiki/{TEXT}_%28Planet%29</FL:url>
  </FL:action>
  <FL:action id="wikifr">
    <FL:caption>Andere Sprachen//wiki Artikel in Französisch</FL:caption>
    <FL:url>http://fr.wikipedia.org/wiki/{TEXT}_%28plan%C3%A8te%29</FL:url>
  </FL:action>
</FL:actions>
</FL:smartTag>
<FL:smartTag type="urn:schemas-Bsp:astronomisch#aussereplaneten">
  <FL:propertyPage>http://www.KAP29Bsp.com/planet/BSP29_02_äp.htm</FL:propertyPage>
  <FL:caption>Äußere Planeten v1</FL:caption>
  <FL:terms>
    <FL:termListWithProps>
      <FL:t><FL:prop monde="2" typ="gasriese"/>mars</FL:t>
      <FL:t><FL:prop monde="48" typ="gasriese"/>saturn</FL:t>
      <FL:t><FL:prop monde="27" typ="gasriese"/>uranus</FL:t>
      <FL:t><FL:prop monde="13" typ="gasriese"/>neptun</FL:t>
      <FL:t><FL:prop monde="1" typ="terrestrisch"/>pluto</FL:t>
    </FL:termListWithProps>
  </FL:terms>
  <FL:actions>
    <FL:action id="wikide">
      <FL:caption>wiki Artikel</FL:caption>
      <FL:url>http://www.wikipedia.org/wiki/{TEXT}_%28Planet%29</FL:url>
    </FL:action>
    <FL:action id="wikifr">
      <FL:caption>Andere Sprachen//wiki Artikel in Französisch</FL:caption>
      <FL:url>http://fr.wikipedia.org/wiki/{TEXT}_%28plan%C3%A8te%29</FL:url>
    </FL:action>
  </FL:actions>
</FL:smartTag>
</FL:smartTagList>

```

## Speicherorte für MOSTL-Listen

Die Speicherorte für MOSTL werden im SDK eingehend vorgestellt. Hier wird nur die Frage der Spracherkennung diskutiert.

Word durchsucht vier Ordner nach Smarttag-Listen:

*C:\Programme\Gemeinsame Dateien\Microsoft Shared\Smart Tag\*

*C:\Programme\Gemeinsame Dateien\Microsoft Shared\Smart Tag\1031*

*C:\Dokumente und Einstellungen\<Benutzername>\Anwendungsdaten\Microsoft\Smart Tag Lists*


*C:\Dokumente und Einstellungen\<Benutzername>\Anwendungsdaten\Microsoft\Smart Tag Lists\1031*

Die mit den vierstelligen Zahlen bezeichneten Ordner sind sprachspezifisch; Word wird im Ordner für die gegenwärtig aktive Sprache (LCID) suchen. »1031« ist die LCID für Deutsch (Deutschland).

**TIPP**

Um die LCID der gegenwärtigen Installation zu ermitteln, geben Sie

```
?Word.Application.LanguageSettings.LanguageID(msoLanguageIDUI)
```

in das Direktfenster des Visual Basic-Editors ein und drücken die -Taste. Eine Liste aller LCID finden Sie, wenn Sie in der Word-Hilfe nach dem Begriff »LCID« suchen.

Sollen die Ausdrücke einer Liste in allen Sprachen erkannt werden, muss die Liste also in einem der beiden Ordner ohne LCID gespeichert werden.

Ausdrücke für mehrere Sprachen können sich in der gleichen Liste im sprachenlosen Ordner befinden und separat erkannt werden. Nicht nur der Text eines Dokuments wird geprüft; auch die LCID-Eigenschaft des Absatzes wird an den »Recognizer« weitergegeben. Er vergleicht diese Information mit dem Inhalt des MOSTL-Elements `<lcid>`. Das Element kann eine kommagetrennte Auflistung mehrerer LCID enthalten sowie den Eintrag »0« oder »\*« (bedeutet »alle Sprachen«).

Beträgt der Wert des Elements `<FL:lcid>1031</FL:lcid>`, wird ein Ausdruck nur erkannt, wenn er in einem mit der LCID 1031 formatierten Absatz steht. Nicht einmal andere deutsche Dialekte wie Deutsch (Schweiz) oder Deutsch (Österreich) werden als gültig betrachtet.

Um einen Ausdruck zu erkennen, egal mit welcher Sprache er formatiert ist, muss das Element weggelassen werden oder »alle Sprachen« festlegen: `<FL:lcid>0</FL:lcid>`

## Das Beispiel installieren

Anhand der Erläuterungen des Abschnitts »Speicherorte für MOSTL-Listen« können Sie entscheiden, wo die Hauptliste (*Bsp29\_02.xml*) gespeichert werden soll. Es bleiben noch die Dateien, die die weiteren Elemente dieses Beispiels veranschaulichen.



Alle in diesem Abschnitt aufgelisteten Beispieldateien finden Sie auf der CD-ROM zum Buch im Ordner `\Beispiele\Kap29`.

Darauf wird momentan in der Hauptliste mit dem Fantasie-URL – <http://www.KAP29Bsp.com/planet/> – hingewiesen. Um der Diskussion zu folgen, sollten die folgenden Dateien auf einem Webserver gespeichert werden:

*BSP29\_02\_mi.htm*

*BSP29\_02\_dl.htm*

*BSP29\_02\_ip.htm*

*BSP29\_02\_äp.htm*

*BSP29\_02\_up\_v2.xml*

*BSP29\_02\_up\_v3.xml*

*BSP29\_02\_v2.xml*

*BSP29\_02\_v3.xml*

Die URLs in diesen Dateien sind entsprechend anzupassen und wieder mit der Codierung UTF-8 zu speichern.

*BSP29\_02.xml* (die Hauptliste)

*BSP29\_02\_v2.xml*

*BSP29\_02\_v3.xml*

#### HINWEIS

Nur die *\*.xml*-Dateien für die automatische Aktualisierung *müssen* sich auf einem Webserver befinden. Die anderen dürfen als normale Dateien lokal oder im Netzwerk gespeichert werden dürfen. In diesem Fall muss der Pfad mit einem Datei-URL festgelegt werden: *file://[/[Laufwerk]:\[Pfadangabe]*.

#### PROFITIPP

Falls kein Webserver zur Verfügung steht, kann IIS (»Internet Information Services«) unter Windows installiert und als »localhost« eingesetzt werden. Um IIS zu installieren, öffnen Sie in der Systemsteuerung *Software* und klicken Sie auf *Windows-Komponenten hinzufügen/entfernen*; es öffnet sich der *Assistent für Windows-Komponenten*. Nach Aktivieren des Kontrollkästchens *Internet-Informationdienste (IIS)* klicken Sie auf *Weiter*. Die Komponente wird installiert.

Auf Laufwerk C: wird dadurch ein neuer Ordner *Inetpub* mit dem Unterordner *wwwroot* erstellt. Fügen Sie *wwwroot* einen neuen Unterordner für die Beispieldateien hinzu (beispielsweise *Kap29ST*) und kopieren Sie die Dateien dort hinein. In diesem Fall lautet der URL: *http://localhost/Kap29ST/*.

## Die automatische Aktualisierung einer MOSTL

Damit die Smarttag-Funktionalität prüft, ob die Liste zu aktualisieren ist, werden der XML-Datei die Elemente aus Tabelle 29.1 hinzugefügt.

Tabelle 29.1 MOSTL-Elemente für die automatische Aktualisierung der Liste

Elemente für die Aktualisierung	Beschreibung
<updateable>	Die MOSTL kann aktualisiert werden, wenn der Wert des Elements »true« beträgt.
<autoupdate>	Die Prüfung und Aktualisierung erfolgt automatisch, wenn der Wert des Elements »true« beträgt.
<updateURL>	Enthält den URL zur Spezifikation für die Aktualisierung (im Beispiel handelt es sich um die Datei <i>Bsp29_02_up_v2.xml</i> , in Listing 29.3 ersichtlich).
<updateFrequency>	Legt in Minuten fest, wie oft die Prüfung durchzuführen ist. Fehlt dieses Element, wird wöchentlich geprüft.
<lastCheckpoint>	Legt die Versionsnummer der gegenwärtigen Datei fest. Befindet sich eine niedrigere Zahl im Element <checkpoint> der Spezifikation oder im Element <lastCheckpoint> der neuen MOSTL ( <i>Bsp29_02_v2.xml</i> ), wird keine Aktualisierung durchgeführt.

**Listing 29.3** Die Spezifikation verweist auf den Speicherort der neuen MOSTL, die sich im gleichen Ordner befinden muss.




```
<FLUP:smarttaglistupdate xmlns:FLUP="urn:schemas-microsoft-com:smarttags:listupdate">
  <FLUP:checkpoint>2</FLUP:checkpoint>
  <FLUP:smarttaglistdefinition>BSP29_02_v2.xml</FLUP:smarttaglistdefinition>
</FLUP:smarttaglistupdate>
```

Die Prüfung, ob Aktualisierungen vorliegen, erfolgt jedoch nicht »einfach so«. Der Benutzer muss zuerst eine Aktion ausführen, um Word »einen Stups zu geben«. Auch dann erscheint die Aktualisierung meistens nicht sofort; oft muss Word beendet und neu gestartet werden. Oder die Aktualisierung kann über das Objektmodell erzwungen werden:

```
Application.SmartTagTypes.ReloadAll
```

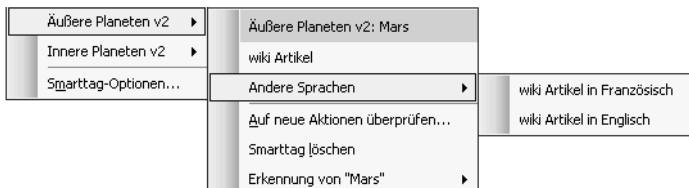
Bei erfolgreicher Aktualisierung ersetzt der Inhalt der neuen MOSTL den der gegenwärtigen. Der Dateiname bleibt jedoch gleich (*Bsp29\_02.xml*).

Um die Aktualisierung nachzuvollziehen, müssen alle URL der Beispieldateien angepasst und diese in die passenden Ordner kopiert werden. Führen Sie dann folgende Schritte aus:

1. Starten Sie Word und öffnen Sie die Beispieldatei *Bsp29\_02.doc*. (Beachten Sie die *MacroButton*-Feldfunktion mit der Beschriftung *ReloadAllSmartTagTypes*).
2. Geben Sie »Mars« am Dokumentanfang ein und drücken Sie die -Taste.
3. Führen Sie den Mauszeiger über das Wort und klappen Sie, sobald das Smarttag-Symbol erscheint, das Menü auf. Es sollten Einträge für die Listen »Äußere Planeten« sowie »Innere Planeten« vorhanden sein.
4. Wählen Sie »Innere Planeten«. Als Aktionen werden die Einträge »wiki Artikel« sowie »Andere Sprache« erscheinen. Letzterer öffnet ein weiteres Menü zu einem französischen Artikel.
5. Führen Sie eine der Aktionen aus, um die Aktualisierung auszulösen. (Sie können die Handlung bestätigen, indem Sie die Datei *Bsp29\_02.xml* öffnen und nach dem <FL:caption>-Element suchen. Es sollte wie folgt aussehen: <FL:caption>Innere Planeten v2</FL:caption>.)
6. Da Word die Smarttag-Informationen im Arbeitsspeicher zwischenspeichert, öffnen Sie den VB-Editor ( + ), blenden Sie das Direktfenster ein, und führen Sie `Application.SmartTagTypes.ReloadAll` aus. Es könnte sogar notwendig sein, den Ausdruck zu löschen und nochmals einzugeben.

Sobald die neue MOSTL von Word korrekt erkannt und eingesetzt wird, erscheinen andere Einträge, wie in Abbildung 29.6 ersichtlich. Die Beschriftungen führen »v2« auf, und ein englischer Artikel wird neu angeboten.

**Abbildg. 29.6** Die aktualisierte MOSTL: die Beschriftungen unterscheiden sich, und ein zusätzlicher Eintrag erscheint unter *Andere Sprachen*.



**HINWEIS**

Bitte beachten Sie, dass die Verknüpfungen für französische und englische Artikel nur zu brauchbaren Ergebnissen führen, wenn der Planet in diesen Sprachen gleich benannt wird. Sonst zeigt *Wikipedia* eine »nicht vorhanden«-Meldung an.

Nach zwei Minuten können Sie die obigen Schritte wiederholen, und das Smarttag wird mit Version 3 aktualisiert. Diese entfernt »Mars« aus der Liste der äußeren Planeten und fügt »Jupiter« hinzu.

**HINWEIS**

Es fällt auf, dass der Eintrag *Auf neue Aktionen überprüfen* dem ersten Untermenü hinzugefügt wird. Dies wird durch das Element `<downloadURL>` verursacht. Beim Anwählen wird lediglich die sich im URL befindende Webseite im Browser geöffnet.

Die HTML-Datei des sich im Element `<propertyPage>` befindenden URLs wird durch Betätigung der Schaltfläche *Eigenschaften* im Dialogfeld *Extras/AutoKorrektur-Optionen/Smarttag* im Browser geöffnet.

## Erkennungsausdrücke festlegen

termList-  
Element

Das Listing 29.2 lässt erkennen, dass zwei Arten der Definition für die »Recognizer«-Ausdrücke zur Verfügung stehen. Die inneren Planeten werden in einem `<termList>` Element aufgelistet:

```
<FL:terms>
  <FL:termList>merkur,venus,erde,mars</FL:termList>
</FL:terms>
```

In diesem Element wird auf die Groß-/Kleinschreibung nur bei akzentuierten Zeichen geachtet. Falls Sie ausdrücklich Groß-/Kleinschreibung spezifizieren wollen, müssen die Ausdrücke mit Anführungszeichen umgeben werden. Sind Anführungszeichen Teil eines solchen Ausdrucks, sind drei aufeinander folgende Anführungszeichen notwendig: »""Merkur""« erkennt »Merkur« in einem Dokument (»merkur« jedoch nicht).

Im Smart Tag SDK steht, dass mit dem Element `<terms>` Werte dem »Property Bag« zugewiesen werden können, indem sie in einem `<termsWithProperty>` Element aufgeführt werden:

```
<FL:terms>
  <FL:termListWithProps>
    <FL:t><FL:prop monde="2" typ="gasriese"/>mars</FL:t>
    <FL:t><FL:prop monde="48" typ="gasriese"/>saturn</FL:t>
    <FL:t><FL:prop monde="27" typ="gasriese"/>uranus</FL:t>
    <FL:t><FL:prop monde="13" typ="gasriese"/>neptun</FL:t>
    <FL:t><FL:prop monde="1" typ="terrestrisch"/>pluto</FL:t>
  </FL:termListWithProps>
</FL:terms>
```

Es ist jedoch nicht offensichtlich, dass dies tatsächlich funktioniert, da sie von MOSTL-Aktionen nicht angesprochen werden können. Sie können jedoch durch das Word-Objektmodell angesprochen werden, wie der Beispielcode im Abschnitt »Smarttags im Word-Objektmodell« veranschaulicht.

<re>-  
Element

Anstelle einer Liste von Ausdrücken können mit regulären Ausdrücken Suchmuster definiert werden, ähnlich wie in Words *Suchen und Ersetzen*-Funktionalität. Smarttags unterstützen Perl-Dialekt, leicht modifiziert für den Gebrauch mit XML (die Zeichen »<« und »>« werden mit den Zeichenfol-

gen &lt; bzw. &gt; festgelegt). Die Liste der Planeten könnte beispielsweise wie folgt definiert werden:

```
<FL:re>
  <FL:exp switches="i">merkur|venus|erde|mars|jupiter|saturn|uranus|neptun|pluto</FL:exp>
</FL:re>
```

oder

```
<FL:re>
  <FL:exp switches="i">merkur|venus|erde|mars</FL:exp>
  <FL:exp switches="i">jupiter|saturn|uranus|neptun|pluto</FL:exp>
</FL:re>
```

(Das Attribut `switches="i"` legt fest, dass der Vergleich ohne Beachtung der Groß-/Kleinschreibung zu erfolgen hat.)

Im SDK finden Sie noch weitere Möglichkeiten beschrieben, unter anderem:

- Das Erstellen einer binären Datei mit einer komprimierten, sortierten Liste von Ausdrücken. Diese Methode ist nützlich, wenn die Liste viele Ausdrücke enthält.
- Auf eine externe Datei mit regulären Ausdrücken in einem `<includeFile>`-Element hinweisen.
- Den Gebrauch einer kontextfreien Grammatik.

## MOSTL-Aktionen festlegen

Das Beispiel zeigt die meisten zur Verfügung stehenden Elemente für Aktionen auf. Zusammengefasst sind die Hauptpunkte:

- Jede für ein bestimmtes Smarttag-Element (`SmartTagType`) definierte Aktion muss eine eindeutige ID haben.
- Das Element `<caption>` legt die Beschriftung im Aktion-Menü fest. Um ein Untermenü zu erstellen, wird `///` zwischen den Beschriftungen eingefügt:

```
<FL:caption>Andere Sprachen///wiki Artikel in Französisch</FL:caption>
```

- Die einzige Handlung, die eine Aktion ausführen kann, ist, den URL im Element `<URL>` aufzurufen. Als Parameter können dem URL einzig die LCID sowie der erkannte Text hinzugefügt werden.

## Ein Smarttag mit VB6 entwickeln

Wie schon erwähnt, ist es möglich, mit einer Smarttag-DLL mehr Funktionalität anzubieten. Um dies zu veranschaulichen, stellen wir ein Beispiel vor, das auf eine Datenbank zugreift und Informationen aus Datensätzen in das Dokument einfügt – beides Handlungen, zu denen eine MOSTL nicht fähig ist.

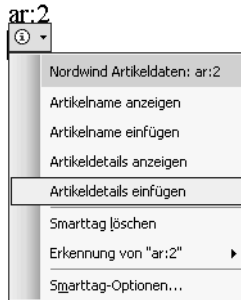
Angenommen, die Anwender arbeiten hauptsächlich in Word, müssen aber gelegentlich Artikel- und Kundendaten nachschlagen und in Dokumente einfügen. Statt in die Datenbankumgebung



umzuschalten, wäre es vorteilhaft, wenn sie direkt von Word aus Daten effizient nachfragen könnten. Smarttags bieten eine benutzerfreundliche Schnittstelle an: Der Anwender muss sich nicht mit Dialogfeldern abgeben; alles spielt sich innerhalb eines kleinen Bereichs auf dem Bildschirm ab.

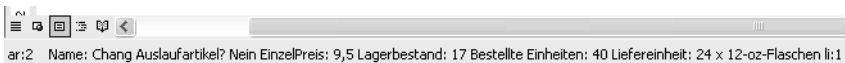
Wie die Abbildung 29.7 veranschaulicht, wird ein Kürzel eingegeben und vom Smarttag erkannt. Das Aktionen-Menü bietet mehrere Optionen an.

Abbildg. 29.7 Das Smarttag erkennt das Kürzel und zeigt entsprechende Optionen an.

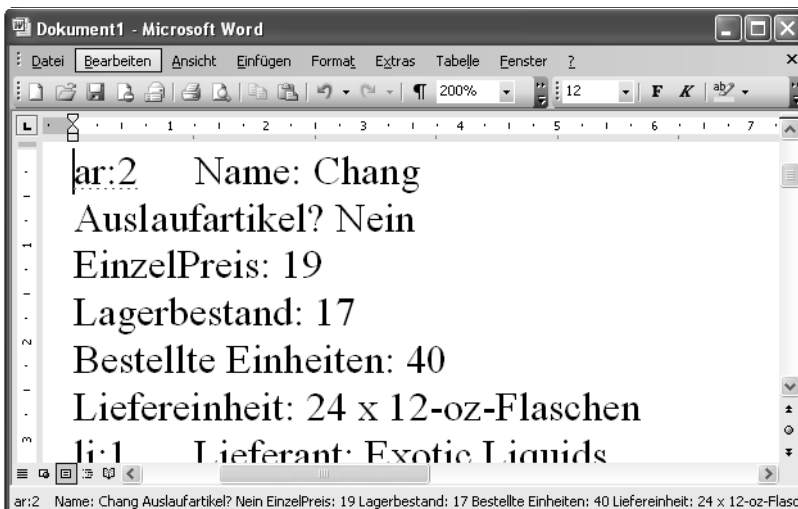


Informationen zum Artikelnamen oder die Details können in der Statusleiste angezeigt (Abbildung 29.8) oder ins Dokument eingefügt (Abbildung 29.9) werden. Der Anwender kann nicht benötigte Informationen löschen oder nach Ausführen des Befehls *Rückgängig machen* die Aktion erneut starten, wenn er einen anderen Artikel will.

Abbildg. 29.8 Artikeldetails werden in der Statusleiste angezeigt



Abbildg. 29.9 Die gleichen Informationen können auch direkt ins Dokument eingefügt werden.



**HINWEIS** Um die Beispiel-Smarttag-DLL zu installieren, beachten Sie die Angaben im Abschnitt »Eine Smarttag-DLL erstellen und installieren« in diesem Kapitel.

## Die Projekt-Spezifikationen

Dieses Smarttag soll Daten wie folgt in der Access-Nordwind-Datenbank nachschlagen:

- **Artikel-Daten:** basiert auf dem Kürzel »ar:« plus Artikel-Code (Artikel-Nr) oder den ersten Buchstaben des Artikelnamens.
- **Kunden-Daten:** basiert auf dem Kürzel »ku:« plus Kunden-Code (Kunden-ID) oder den ersten Buchstaben des Firmen- oder des Kontaktnamens.

Das Artikel-Smarttag soll vier Aktionen anbieten: Artikeldetails oder -namen anzeigen sowie Artikeldetails oder -namen einfügen. Gleiches gilt für die Kundendaten.

Bei der Auswahl einer Anzeige-Option werden so viel Detail-Informationen in der Statusleiste angezeigt, wie darin Platz vorhanden ist. Das Ausführen einer Einfügen-Option fügt pro Feld eine Zeile ins Dokument ein; jede Zeile enthält den Feldnamen, gefolgt von einem Doppelpunkt und dem Feldinhalt. Damit kann der Anwender problemlos die Daten löschen, die er nicht benötigt.

Da der Primärschlüssel vieler Datenbanktabellen eine Ganzzahl ist, muss der Anwender eine Bezeichnung (Kürzel) eingeben, um die gewünschte Tabelle anzugeben. Unmittelbar darauf muss er einen Doppelpunkt, gefolgt von einem Code oder Namen eingeben. Um beispielsweise den Artikel mit der ID 55 nachzuschlagen, wird *ar:55* eingegeben (wobei auf Groß-/Kleinschreibung nicht geachtet wird). Der Recognizer wird demzufolge Werte erkennen wie

*ar:1, AR:ch, ku:ALFKI, KU:123* usw.

Theoretisch wäre es möglich, eine Zeichenfolge in der Datenbank nachzuschlagen, bevor sie als Smarttag markiert wird. Da die Erkennung jedoch laufend ausgeführt wird, könnte dies die System- und Netzwerkressourcen arg strapazieren. Deshalb wird in der Datenbank erst nachgeschlagen, wenn der Benutzer eine Aktion auswählt. Ist kein passender Datensatz vorhanden, wird eine entsprechende Meldung eingeblendet.

Dieses Smarttag-Beispiel wurde mit klassischem Visual Basic (VB6) entwickelt. Um ein neues Smarttag zu erstellen, starten Sie VB6 und wählen als Projektart *ActiveX DLL*. Setzen Sie einen Verweis auf die Bibliothek »Microsoft Smart Tag 2.0«. Dieses Beispiel verfügt zusätzlich über Verweise auf die Bibliotheken »Microsoft VBScript Regular Expression 5.5« und »Microsoft ActiveX Data Objects 2.8«.

Das Projekt ist in vier Module gegliedert:

- Das Klassenmodul `clsRecognizer` implementiert die Interface `SmartTagLib.ISmartTagRecognizer`. (Falls das Projekt eine »Property Page« oder »Tokenizer« benötigt (was hier nicht der Fall ist), muss auch die Interface `SmartTagLib.ISmartTagRecognizer2` implementiert werden.)
- Das Klassenmodul `clsActions` implementiert die Interface `SmartTagLib.ISmartTagAction`. (Falls das Projekt dynamische Beschriftungen oder die Fähigkeit, den Smarttag-Indikator (die gepunktete Linie) auszuschalten benötigt, muss auch die Interface `SmartTagLib.ISmartTagAction2` implementiert werden.)
- Das Modul `SharedCode` mit einigen allgemeinen, von allen Modulen aufgerufenen Prozeduren.
- Das Modul `DBAccess`, das alle Prozeduren enthält, die mit der Datenbank kommunizieren.

Der Hauptteil des Codes ist routinemäßig; nur die Implementierung der Recognize-Methode des »Recognizer« wird hier in Listing 29.4 wiedergegeben.



Die Dateien *Bsp29\_03\_Actions.cls*, *Bsp29\_03\_DBAccess.bas*, *Bsp29\_03\_Recognizer.cls* sowie *Bsp29\_03\_SharedCode.bas* finden Sie auf der CD-ROM im Ordner *\Beispiele\Kap29\Bsp29\_03\_VB.zip*, zusammen mit den restlichen VB6-Projektdateien. Es handelt sich um reine Textdateien, die im Windows-Editor sowie in VB6 geöffnet werden können.

Listing 29.4

»Recognizer«-Code dieses VB6-Beispiels

```
Private Sub ISmartTagRecognizer.Recognize(ByVal Text As String, _
    ByVal DataType As SmartTagLib.IF_TYPE, ByVal LocaleID As Long, _
    ByVal RecognizerSite As SmartTagLib.ISmartTagRecognizerSite)

    Dim intIndex As Integer
    Dim intLength As Integer
    Dim intPreviousIndex As Integer
    Dim objSmartTagProperties As SmartTagLib.ISmartTagProperties

    Dim objRegExp As VBScript_RegExp_55.RegExp
    Dim objMatches As VBScript_RegExp_55.MatchCollection
    Dim objMatch As VBScript_RegExp_55.Match
    Dim objSpaceMatches As VBScript_RegExp_55.MatchCollection
    Dim objSpacematch As VBScript_RegExp_55.Match

    On Error GoTo Err_Handler
    Select Case DataType

        ' Office gibt ein PARA (Absatz) oder ein CELL (Zelle) zurück
        ' Dieser Smarttag wird nur für Word entwickelt, weshalb nur
        ' der Typ PARA bearbeitet wird
        Case IF_TYPE_PARA
            Set objRegExp = New VBScript_RegExp_55.RegExp
            With objRegExp
                ' Groß-/Kleinschreibung ignorieren, so dass alle Variationen
                ' des Präfix erkannt werden, wie AR:, Ar:, aR: and ar:
                .IgnoreCase = True
                ' Global = False würde bedeuten, die Suche durch RegExp endet nach
                ' dem ersten Treffer. Auf True festlegen, um alles zu durchsuchen
                .Global = True
                ' Treffer wie ku:1 werden erlaubt, obwohl die Kundencodes nicht numerisch sind
                .Pattern = "\b(AR|KU):"
                Set objMatches = .Execute(Text)
            End With

            intPreviousIndex = 0
            For Each objMatch In objMatches
                With objMatch
                    If .FirstIndex < intPreviousIndex Then
                        ' Dieser Treffer steht mitten im vorhergehenden,
                        ' weshalb nichts unternommen wird
                    Else
                        ' Möglicher Neutreffer,
                        ' das Leerzeichen am Anfang fallen lassen
                        intIndex = .FirstIndex + 1
                    End If
                End With
            Next objMatch
        End Select
    End Sub
```

**Listing 29.4** »Recognizer«-Code dieses VB6-Beispiels (Fortsetzung)

```

' Nochmals RegExp ausführen, um das darauf folgende Leerzeichen zu suchen
With objRegExp
.Global = False
.Pattern = "\s"
' MsgBox Mid(Text, intIndex + 3)
Set objSpaceMatches = .Execute(Mid(Text, intIndex + 3))
End With
'Mindestens ein Treffer muss vorliegen ...
If Not objSpaceMatches Is Nothing Then
Set objSpacematch = objSpaceMatches.Item(0)
' MsgBox objSpacematch.FirstIndex & " " & objSpacematch.Length
' ... aber ein dem Kolon direkt folgendes Leerzeichen gilt nicht
If objSpacematch.FirstIndex > 0 Then
' Ein Treffer liegt vor
intLength = objMatch.Length + objSpacematch.FirstIndex
' Die neue Position im Text speichern
intPreviousIndex = intIndex + intLength - 1
Set objSmartTagProperties = RecognizerSite.GetNewPropertyBag
' Diese Suche ist relativ einfach. Falls Ihr Recognizer schwer
' arbeiten muss, um Treffer zu erzielen, wäre es u.U. nützlich,
' die Ergebnisse im Property Bag zu speichern (folgende Zeile),
' so dass die Aktionen nicht nochmals durchgeführt werden müssen.
objSmartTagProperties.Write _
Key:="LOOKUP", _
Value:=Mid(Text, intIndex + 3, intLength - 3)
Select Case Left(UCase(Trim(objMatch.Value)), 2)
Case "AR"
RecognizerSite.CommitSmartTag _
SharedCode.ST_Type(1), intIndex, intLength, objSmartTagProperties
Case "KU"
RecognizerSite.CommitSmartTag _
SharedCode.ST_Type(2), intIndex, intLength, objSmartTagProperties
Case Else
End Select
Set objSmartTagProperties = Nothing
End If
Set objSpacematch = Nothing
End If
Set objSpaceMatches = Nothing
End If
End With
Next
Set objMatches = Nothing
Set objRegExp = Nothing
Case Else
End Select
Exit Sub

Err_Handler:
MsgBox Err.Number & vbCrLf & Err.Description
Exit Sub
End Sub

```

## Implementierung des *SmartTagLib.ISmartTagRecognizer*-Interface

Jeder »Recognizer« kann mehrere Smarttag-Typen erkennen. Einige der Eigenschaften gelten für den »Recognizer«, andere für jeden Smarttag-Typ, womit der »Recognizer« arbeitet.

Das SmartTagLib.ISmartTagRecognizer-Interface dieses Projekts richtet die nur lesbaren Eigenschaften in der Tabelle 29.2 ein.

Tabelle 29.2 *ISmartTagRecognizer*-Eigenschaften

Eigenschaft	Beschreibung
ProgID	Legt die Bezeichnung für die Windows-Registrierung fest (siehe Abschnitt »Eine Smarttag-DLL erstellen und installieren«), in diesem Fall <b>Bsp29_03.clsRecognizer</b> .
Name	Ergibt den zweiten Teil der Smarttag-Beschreibung, die im Dialogfeld <i>Smarttags</i> (Menübefehl <i>Extras/AutoKorrektur-Optionen</i> ) erscheint. Kann sprachenspezifisch (nach LCID) sein.
Desc	Ein beschreibender Name für den »Recognizer«. Erscheint nicht in der Word-Benutzerschnittstelle.
SmartTagCount	Die Anzahl der <b>SmartTagType</b> -Objekte, die der »Recognizer« zur Verfügung stellt, bestimmt die Anzahl der Einträge im Smarttag-Dialogfeld. In diesem Beispiel sind es zwei: je einer für Artikel und Kunden.
SmartTagName	Der URI (Name) jedes <b>SmartTagType</b> . Im Beispiel werden <b>urn:bsp2903#Artikel</b> sowie <b>urn:bsp2903#Kunden</b> festgelegt.

Die einzige implementierte Methode ist *Recognize*, mit den Parametern in Tabelle 29.3.

Tabelle 29.3 Parameter der *ISmartTagRecognizer.Recognize*-Methode

Parameter	Beschreibung
Text	Die Zeichenkette, die der »Recognizer« erkennen soll. Diese Information wird von der aufrufenden Anwendung übergeben.
DataType	Die Textart im Text-Parameter. Die Smarttag-Technologie erkennt zwei Arten: <b>Para</b> (Word-Absatz) sowie <b>Cell</b> (Excel-Zelle). Da dieses Smarttag nur für Word bestimmt ist, behandelt das Projekt nur Text der Art <b>Para</b> . <b>Select Case DataType</b> <b>Case IF_TYPE_PARA</b>
LocaleID	Die Sprache der gegenwärtigen Umgebung; im Fall von Word die Sprache des Absatzes, in dem sich der Text befindet. Die Information wird als LCID übergeben. In diesem Beispiel wird dieser Parameter nicht ausgewertet.
RecognizerSite	Ein <b>ISmartTagRecognizerSite</b> -Objekt. Das Beispiel benutzt die Interface-Methode <b>CommitSmartTag</b> , um Informationen einer im Text erkannten Zeichenfolge zurück an die rufende Anwendung zu geben. Dies sind: der URI des <b>SmartTagType</b> , der erkannt wurde; die Position im Text des ersten Zeichens der erkannten Zeichenkette; die Länge der erkannten Zeichenkette. Über diese Schnittstelle können auch Name/Wert-Paare des »Property Bag« zugewiesen werden.

Wie schon erwähnt, setzt dieses Projekt die Regular Expressions-Funktionalität aus VBScript ein. Das bedeutet, dass Scripting Version 5.5 auf dem Rechner installiert sein muss. Da mit dieser Version Nicht-ANSI-Zeichen (wie Umlaute) schwer zu spezifizieren sind, wird das Präfix (beispielsweise `\bAR:`) zuerst gesucht und dann das erste darauf folgende Leerzeichen.

Wurde eine Textstelle erkannt, erstellt der »Recognizer« eine »Property« im »Property Bag« für den Smarttag mit dem Namen »Lookup« und den Wert des Textes nach dem Kürzel (»ar:« bzw. »ku:«). Somit muss der gleiche Datensatz nicht mehrmals in der Datenbank nachgeschlagen werden.

```
objSmartTagProperties.Write _
    Key:="LOOKUP", _
    Value:=Mid(Text, intIndex + 3, intLength - 3)
Select Case Left(UCase(Trim(objMatch.Value)), 2)
    Case "AR"
        RecognizerSite.CommitSmartTag _
            SharedCode.ST_Type(1), intIndex, intLength, objSmartTagProperties
    Case "KU"
        RecognizerSite.CommitSmartTag _
            SharedCode.ST_Type(2), intIndex, intLength, objSmartTagProperties
    Case Else
    End Select
Set objSmartTagProperties = Nothing
```

## Implementierung des *SmartTagLib.ISmartTagAction*-Interface

Der Code im Klassenmodul für die Smarttag-Aktionen implementiert Aktionen für alle Smarttag-Typen (in diesem Fall sind es zwei). Es gibt vier *Verben* (Verbs) für jeden SmartTagType. Intern stellen eindeutige numerische ID-Werte diese dar, gegen außen können sie mit sprachenunabhängigen Zeichenketten – *Verbnamen* (Verb names) – angesprochen werden. Verbnamen können wie Funktionen in einer Programmiersprache wie VB betrachtet werden.

Viele der Prozeduren des Action-Interface dienen der Festlegung und dem Lesen der Werte dieser Verben.

Das SmartTagLib.ISmartTagAction-Interface dieses Projekts richtet die nur-lesbaren Eigenschaften in der Tabelle 29.4 ein.

Tabelle 29.4 *ISmartTagAction*-Eigenschaften

Eigenschaft	Beschreibung
ProgID	Legt die Bezeichnung für die Windows-Registrierung fest, in diesem Fall <code>Bsp29_03.clsActions</code> .
Name	Ein Name für die Aktion. Wird in der Word-Benutzerschnittstelle nicht verwendet.
Desc	Ein beschreibender Name für die Aktion. Wird in der Word-Benutzerschnittstelle nicht verwendet.
SmartTagCount	Die Anzahl <b>SmartTagType</b> , die diese <b>Action</b> -Klasse unterstützt; in diesem Fall sind es zwei.
SmartTagName	Der URI (Name) jedes <b>SmartTagType</b> . Im Beispiel werden <code>urn:bsp2903#Artikel</code> sowie <code>urn:bsp2903#Kunden</code> festgelegt.

Tabelle 29.4 *ISmartTagAction*-Eigenschaften

Eigenschaft	Beschreibung
SmartTagCaption	Die oberste Beschriftung des Aktionen-Menüs sowie der erste Teil der Beschreibung im Dialogfeld <i>Smarttags</i> unter <i>Extras/AutoKorrektur-Optionen</i> .
VerbCount	Die Anzahl der Verben für den spezifizierten <b>SmartTagType</b> .
VerbID	Die eindeutige Ganzzahl, die ein Verb (für diese <b>Action</b> -Klasse) darstellt.
VerbCaptionFromID	Die Beschriftung im Aktionen-Menü für die mit der ID verbundene Aktion (kann sprachenspezifisch, nach LCID, definiert werden).
VerbNameFromID	Der sprachenunabhängige Verbnamen für eine bestimmte Verb-ID.

Das Action-Interface implementiert eine einzige Methode – **InvokeVerb**. Sie hat die in Tabelle 29.5 aufgelisteten Parameter.

Tabelle 29.5 Die Parameter der Methode *ISmartTagAction.InvokeVerb*

Parameter	Beschreibung
VerbID	Die Ganzzahl ID des Verbs, das ausgeführt wird.
ApplicationName	Die <b>ProgID</b> der rufenden Anwendung. Diese ist nützlich, wenn die rufende Anwendung keine Information für den Parameter <b>Target</b> übergibt.
Target	Ein Objekt, worüber die rufende Anwendung angesprochen werden kann, um Daten zurück zu schreiben oder Handlungen auszuführen. Um Text in der Statusleiste von Word anzuzeigen, beispielsweise: <b>Target.Application.Statusbar</b> .
Properties	Der »Property Bag« des Smarttags.
Text	Der Text, der von der anrufenden Anwendung übergeben wurde (nur lesbar).
XML	Der XML-Text, der von der anrufenden Anwendung übergeben wurde (nur lesbar).

## Eine Smarttag-DLL erstellen und installieren

Die Datei *Bsp29\_03.dll* sowie die Ressourcendatei *Bsp29\_03.udl* befinden sich auf der CD-ROM im Ordner *\Beispiele\Kap29\Bsp29\_03\_VB.zip*. Es ist unwesentlich, wo sie gespeichert werden, solange die beiden im gleichen Ordner stehen. Die Beispieldatenbank *Nordwind.mdb* befindet sich auf der CD-ROM zum Buch im Ordner *Datenbank* und wird in der UDL-Datei über den Pfad *C:\WordBuch\CD-ROM\Datenbank\Nordwind.mdb* geladen. Falls Sie das Beispiel in einem anderen Pfad installieren, muss dieser angepasst werden.

Um die DLL zu installieren, gehen Sie wie folgt vor:

1. Schließen Sie alle Anwendungen, die Smarttags unterstützen.
2. Registrieren Sie die DLL, indem Sie in *Start/Ausführen* folgende Kommandozeile (wenn nötig, dem Speicherpfad auf dem Rechner angepasst) eingeben und ausführen:

```
regsvr32 c:\WordBuch\CD-ROM\Kap29\Bsp29_03.d11
```

- Die folgenden Schlüssel werden in der Windows-Registrierung erstellt:

```
HKEY_CLASSES_ROOT\Bsp29_03.clsActions
HKEY_CLASSES_ROOT\Bsp29_03.clsRecognizer
```


- Zeigen Sie den Inhalt des Unterschlüssels *CLSID* für *Bsp29\_03.clsActions* an, klicken Sie rechts auf den *Standard*-Eintrag, wählen Sie *Ändern* und kopieren Sie den Wert (er soll aussehen wie {85920372-54DC-4280-9D9D-31B89ECF42AE}). Anschließend betätigen Sie *Abbrechen*, um das Dialogfeld auszublenden.
- Suchen Sie den Schlüssel

```
HKEY_CURRENT_USER\Software\Microsoft\Office\Common\Smart Tag\Actions
```

- Klicken Sie mit der rechten Maustaste auf den Schlüssel *Actions*. Wählen Sie aus dem Kontextmenü den Eintrag *Neu/Schlüssel* und geben Sie den kopierten CLSID-Wert ein.
- Wiederholen Sie die Schritte 4 bis 6 für den »Recognizer«, um einen neuen Schlüssel mit dem Wert unter *HKEY\_CLASSES\_ROOT\Bsp29\_03.clsRecognizer* zu erstellen:

```
HKEY_CURRENT_USER\Software\Microsoft\Office\Common\Smart Tag\Recognizer
```

**HINWEIS** Sie können alternativ die Datei *Bsp29\_03.reg* doppelt anklicken, um die Registrierungseinträge zu erstellen. Falls Sie jedoch das Projekt in VB6 öffnen, ändern und die DLL neu kompilieren, ohne die Projektkompatibilität beizubehalten, werden Sie nicht drum herum kommen, die beschriebenen Schritte auszuführen.

Prüfen Sie, ob die beiden Smarttags (*Nordwind Artikeldaten* sowie *Nordwind Kundendaten*) auf der Registerkarte *Smarttags* unter *Extras/AutoKorrektur-Optionen* aufgelistet sind. Geben Sie im Dokument ein Kürzel wie »ar:5« ein, drücken Sie die -Taste und blenden Sie das Aktionen-Menü ein. Wählen Sie dann eine der Aktionen aus.

## Smarttags im Word-Objektmodell

In Word 2002 wurden die Objekte *SmartTag* und *CustomProperty* zusammen mit den zugehörigen Auflistungen *SmartTags* sowie *CustomProperties* eingeführt. Auch neu waren Eigenschaften, die den Speicheroptionen im Abschnitt »Smarttag-Optionen und Ausnahme-Listen« entsprechen.

Weitere Objekte wurden in Word 2003 eingeführt. Einige davon (in Tabelle 29.6 aufgelistet) sind nur relevant, wenn mit Smarttags im Aufgabenbereich *Dokumentaktionen* einer Smart Document-Lösung gearbeitet wird; diese werden im folgenden Kapitel 30 vorgestellt.

Tabelle 29.6 Die *SmartTag*-Objekte des Word 2003-Objektmodells

Name	Beschreibungen und Bemerkungen
SmartTagType	Stellt einen bestimmten Smarttag-Namensraum-URI dar. Eigenschaft des Word <b>Application</b> -Objekts.



Tabelle 29.6 Die *SmartTag*-Objekte des Word 2003-Objektmodells (Fortsetzung)

Name	Beschreibungen und Bemerkungen
SmartTagAction	Stellt eine einzige <b>Action</b> eines Smarttags dar. Element der <b>SmartTagActions</b> -Auflistung.
SmartTagRecognizer	Stellt installierte Komponenten dar, die Text mit Typinformationen bezeichnen.
SmartTag	Stellt eine Zeichenfolge in einem Dokument oder Bereich dar, die bekannte Typinformationen enthält. Ein Element der <b>SmartTags</b> -Auflistung.
CustomProperty	Repräsentiert eine einzelne Instanz einer Benutzereigenschaft für ein Smarttag. Element der <b>CustomProperties</b> -Auflistung. Nicht zu verwechseln mit den <b>CustomDocumentProperties</b> des <b>Document</b> -Objekts von Word.

Ein VBA-Beispiel im Abschnitt »Smarttag-Objekte in Word-VBA verwenden« in diesem Kapitel zeigt, wie diese Objekte eingesetzt werden. Für weitere Informationen schlagen Sie bitte in den VBA-Hilfdateien von Word nach. Eine Zusammenfassung der wichtigsten Informationen folgt.

## SmartTagTypes

Es ist nicht möglich, über das Objektmodell ein SmartTagType-Objekt der SmartTags-Auflistung hinzuzufügen.

Alle Aktionen und »Recognizers« können mit der ReloadAll-Methode dieser Auflistungen aktualisiert werden.

## SmartTagActions

Ein SmartTagAction-Objekt hat nur eine Methode, Execute, die die Aktion ausführt. Ein Action-Objekt wird über seinen Verbnamen (siehe den Abschnitt »Implementierung des SmartTagLib.ISmartTagAction-Interface« in diesem Kapitel) angesprochen:

```
ActiveDocument.SmartTags(1).SmartTagActions("ArtikelNameAnzeigen").Execute
```

Alle Aktionen eines bestimmten SmartTag- oder SmartTagType-Objekts können mit der ReloadActions-Methode dieser Auflistung aktualisiert werden.

Die Eigenschaft SmartTagControlType gibt zurück, welche Art von SmartTag vorliegt. Für Smarttags in einem Dokument beträgt der Wert immer wdControlSmartTag (=1). Alle anderen SmartTagControlType-Werte sind nur im Smart Document-*Dokumentaktionen*-Aufgabenbereich relevant.

## SmartTagRecognizers

Ein »Recognizer« kann über die Enabled-Eigenschaft des Objekts gesperrt oder freigestellt werden.

Alle »Recognizers« einer Installation können mit der ReloadRecognizers-Methode dieser Auflistung aktualisiert werden.

## SmartTags

Sie können eine Auflistung der Smarttags eines Dokuments, eines Bereichs sowie einer Markierung abfragen. Um eine Auflistung der Smarttags eines bestimmten Typs zu bekommen, wird die Methode `SmartTagsByType` der `SmartTags`-Auflistung eingesetzt.

Name/Wert-Paare werden dem »Property Bag« eines `SmartTag`-Objekts mit der `Add`-Methode und dessen `CustomProperties`-Eigenschaft hinzugefügt und über die `Delete`-Methode entfernt.

## Namen, Beschriftungen und ID

Es werden reichlich Namen, Beschriftungen und nicht numerische ID-Werte im Word-Objektmodell, dem `SmartTag`-Interface und der Benutzerschnittstelle verwendet. Um diese Überhäufung etwas zu entwirren und einen besseren Überblick zu ermöglichen, sind diese in Tabelle 29.7 und Tabelle 29.8 aufgelistet, gruppiert und kommentiert.

**Tabelle 29.7** Gleichwertige Smarttag-Namen, Beschriftungen und nicht numerische ID-Werte

Word-Objekt.Eigenschaft	Recognizer- (R) oder Action- (A) Eigenschaft	MOSTL-List-Element	Notiz
<code>SmartTagType.Name</code> , <code>SmartTag.Name</code>	<code>Recognizer.SmartTagName</code> , <code>A.SmartTagName</code>	Type-Attribut des <code>SmartTag</code> -Elements	1
<code>SmartTagType.FriendlyName</code>	<code>A.Caption</code>	Caption-Element innerhalb des <code>SmartTag</code> -Elements	2
<code>SmartTagAction.Name</code>	<code>A.VerbNameFromID</code>	ID-Attribut des <code>Action</code> -Elements	3
[nicht verfügbar]	<code>A.VerbCaptionFromID</code>	Caption-Element innerhalb des <code>Action</code> -Elements	4
[nicht verfügbar]	<code>A.ProgID</code>	[nicht verfügbar]	7
[nicht verfügbar]	<code>A.Name</code>	[nicht verfügbar]	7
[nicht verfügbar]	<code>A.Desc</code>	[nicht verfügbar]	7
<code>SmartTagRecognizer.FullName</code>	[nicht verfügbar]	[nicht verfügbar]	5
<code>SmartTagRecognizer.Caption</code>	<code>A.Caption</code> sowie <code>R.Name</code>	Caption-Element innerhalb des <code>SmartTag</code> -Elements bzw. ein LCID-abhängiger Name, durch die MOSTL-implementierende MOFL.DLL zur Verfügung gestellt	6
[nicht verfügbar]	<code>R.ProgID</code>	[nicht verfügbar]	7
[nicht verfügbar]	<code>R.Desc</code>	[nicht verfügbar]	7
[nicht verfügbar]	[nicht verfügbar]	Name-Element des <code>smartTagList</code> -Elements	7

Tabelle 29.8 Smarttag-Namen, Beschriftungen und nicht numerische ID-Werte

Notiz	Beschreibung
1	Der Namensraum-URI für den <b>SmartTagType</b> . Beispiel: <b>urn:schemas-beispiel:astronomisch#planet</b> . Erscheint nicht in der Word-Benutzeroberfläche.
2	Der erste Teil des Smarttagnamens, wie er im Dialogfeld zum Menübefehl <i>Extras/AutoKorrektur-Optionen</i> auf der Registerkarte <i>Smarttags</i> erscheint. Bildet auch die Überschrift des Aktionen-Menüs.
3	Ein <b>Action</b> -ID-Wert. Er ist logisch nicht gleich der <b>Caption</b> -Eigenschaft des <b>Action</b> -Objekts (die dem Word-Objektmodell nicht zur Verfügung steht), obwohl ihre Werte gleich sein könnten. Er entspricht dem <b>ID</b> -Attribut eines MOSTL- <b>Action</b> -Elements. Er kann auch als ein sprachenunabhängiger »Verbname« betrachtet werden. Erscheint nicht in der Word-Benutzeroberfläche.
4	Beschriftung einer Smarttag-Aktion, wie sie im Aktion-Menü erscheint.
5	Die vollständige Pfadangabe der DLL, die das »Recognizer« implementiert. Für MOSTL-»Recognizers« heißt die DLL <i>MOFL.DLL</i> . Erscheint nicht in der Word-Benutzeroberfläche.
6	Name des »Recognizers« im Dialogfeld zum Menübefehl <i>Extras/AutoKorrektur-Optionen</i> auf der Registerkarte <i>Smarttags</i> . Beispiel: <i>Planeten (Smarttaglisten)</i>
7	Erscheint nicht in der Word-Benutzeroberfläche.

## Weitere Methoden und Eigenschaften

- Weitere Methoden und Eigenschaften, die einen Zusammenhang mit Smarttags haben, sind in Tabelle 29.9 aufgelistet.

Tabelle 29.9 Weitere Word-Methoden und -Eigenschaften

Objekt.Name	Beschreibung/Bemerkungen
<code>Document.CheckNewSmartTags</code>	Methode. Greift auf die Microsoft Office-Website zu, um verfügbare Smarttag-Erkennungs- und Aktionsdateien abzurufen. Entspricht der Schaltfläche <i>Weitere Smarttags</i> im Dialogfeld zum Menübefehl <i>Extras/AutoKorrektur-Optionen</i> (Registerkarte <i>Smarttags</i> ).
<code>Document.EmbedSmartTags</code>	Boolesche Eigenschaft. Legt fest, ob Smarttags beim Speichern im Dokument einzubetten sind. Entspricht der Option <i>Smarttags einbetten</i> im Dialogfeld zum Menübefehl <i>Extras/Optionen</i> (Registerkarte <i>Speichern</i> ).
<code>Document.RecheckSmartTags</code>	Methode. Entfernt vom Grammatikprüfer eingefügte Smarttags und prüft das Dokument erneut unter Verwendung aller aktiven »Recognizers«. Entspricht der Schaltfläche <i>Dokument überprüfen</i> im Dialogfeld zum Menübefehl <i>Extras/AutoKorrektur-Optionen</i> (Registerkarte <i>Smarttags</i> ).
<code>Document.RemoveSmartTags</code>	Methode. Entfernt alle Smarttag-Informationen aus einem Dokument. Entspricht der Schaltfläche <i>Smarttags entfernen</i> im Dialogfeld zum Menübefehl <i>Extras/AutoKorrektur-Optionen</i> (Registerkarte <i>Smarttags</i> ).
<code>Document.SmartTagsAsXMLProps</code>	Boolesche Eigenschaft. Legt fest, ob Smarttag-Informationen beim Speichern in HTML- oder XML-Format beizubehalten ist. Entspricht der Option <i>Smarttags als XML Eigenschaften in Webseiten speichern</i> im Dialogfeld zum Menübefehl <i>Extras/Optionen</i> (Registerkarte <i>Speichern</i> ).

**Tabelle 29.9** Weitere Word-Methoden und -Eigenschaften (Fortsetzung)

Objekt.Name	Beschreibung/Bemerkungen
EmailOptions.EmbedSmartTag	Boolesche Eigenschaft. Legt fest, ob Smarttag-Informationen eingebettet werden sollen beim Versand eines Dokuments als HTML-E-Mail.
Options.DispaySmartTagOptions	Boolesche Eigenschaft. Legt fest, ob Word die Smarttag-Aktionsschaltfläche einblenden soll, wenn der Mauszeiger über als Smarttag erkanntem Text gehalten wird. Entspricht der Option <i>Smarttag-Aktionsschaltflächen anzeigen</i> im Dialogfeld zum Menübefehl <i>Extras/AutoKorrektur-Optionen</i> (Registerkarte <i>Smarttags</i> ).
Options.LabelSmartTags	Boolesche Eigenschaft. Legt fest, ob Word Smarttags im Dokument erkennen soll. Entspricht der Option <i>Text mit Smarttags versehen</i> im Dialogfeld zum Menübefehl <i>Extras/AutoKorrektur-Optionen</i> (Registerkarte <i>Smarttags</i> ).
Window.View.DisplaySmartTags	Boolesche Eigenschaft. Legt fest, ob Smarttags im Dokumentfenster gekennzeichnet werden sollen (Anzeige der gepunkteten Unterstreichung).

## Smarttag-Objekte in Word-VBA verwenden

Das in diesem Abschnitt vorgestellte Beispiel benutzt die Smarttag-Objekte des Word 2003-Objektmodells, um Informationen über die sich im gegenwärtigen Dokument befindenden Smarttags zu sammeln und in ein XML-Dokument auszugeben. Anschließend wird dieses in der Word-Anwendung geöffnet und transformiert. Das Ergebnis sollte ähnlich aussehen wie in Abbildung 29.10.

**Abbildg. 29.10** Ein transformiertes XML-Dokument mit Informationen über alle sich im Quelldokument befindenden Smarttags

Smarttags verwendet in \BSP29\_04.doc

SmartTag.Name: → urn:bsp3103#Artikel	SmartTag.FriendlyName: → Artikel
SmartTag.Text: ar:8x	Download-URL: x
Eigenschaften:	
Name: Value	
LOOKUP: 8x	
SmartTag.Name: → urn:schemas-beispiel:astronomisch#aussereplaneten	SmartTag.FriendlyName: → aussereplaneten
SmartTag.Text: marsx	Download-URL: file:///c:/KAP31/BSP31_02_download.htm
Eigenschaften:	
Name: Value	
type: gasrieser	
monder: 2x	
SmartTag.Name: → urn:schemas-beispiel:astronomisch#innereplaneten	SmartTag.FriendlyName: → innereplaneten
SmartTag.Text: marsx	Download-URL: file:///c:/KAP31/BSP31_02_download.htm
Eigenschaften:	
Name: Value	
LOOKUP: 8x	
SmartTag.Name: → urn:bsp3102#Artikel	SmartTag.FriendlyName: → Artikel
SmartTag.Text: ar:8x	Download-URL: x
Eigenschaften:	
Name: Value	
LOOKUP: 8x	



Die Beispieldatei *Bsp29\_04.doc* mit dem VBA-Code sowie die Transformationsdatei *Bsp29\_04.xsl* finden Sie auf der CD-ROM im Ordner *\Beispiele\Kap29*. Die Konstantwerte für die Pfadangaben in der Prozedur *ListSmartTagsByType* müssen vor der ersten Ausführung angepasst werden.

Der Beispielcode in Listing 29.5, Listing 29.6 sowie Listing 29.7 zeigt zudem auf, wie ein XML-Dokument durch das MSXML DOM erstellt wird. Eine Microsoft XML-Bibliothek muss daher auf dem Rechner installiert sein.

Der Beispielcode wurde mit einem Verweis auf Microsoft XML, v. 5.0 kompiliert, die Teil des Lieferumfangs von Office 2003 ist. Falls Sie diesen Code in eigenen Word-VBA-Projekten verwenden, vergessen Sie nicht, einen Verweis auf diese Bibliothek zu setzen.

Die XSLTransformation wurde mit dem in Kapitel 28 vorgestellten »Microsoft Office 2003 WordProcessingML Transform Inference Tool« erstellt. Das Ergebnis lieferte weder XSLT-Code für die Kopfzeile noch für die verschachtelten Tabellen und musste entsprechend händisch angepasst werden. Dieser Code ist in der XSL-Datei entsprechend kommentiert.

Die Hauptprozedur *ListSmartTagsByType* in Listing 29.5 führt folgende Handlungen aus:

- Erstellt ein DOM-Dokument-Objekt.
- Fügt ihm Knotenpunkte und Text für die sich im gegenwärtigen Dokument befindenden Smarttag-»Recognizers« und -Aktionen hinzu.
- Speichert das DOM-Dokument als eine XML-Datei, öffnet diese in Word und transformiert sie, um die Information ansprechend zu formatieren.

Das Listing 29.6 enthält eine Unterprozedur, *ListSmartTagActions*, die die Informationen für alle Smarttag-Aktionen zusammenstellt. Im VBA-Projekt befindet sich eine Unterprozedur für »Recognizers«, die ähnlich strukturiert ist.

Noch eine Unterprozedur führt Listing 29.7 auf. Diese fügt einen Knotenpunkt für ein Kinderelement mit Text dem DOM-Dokument hinzu. Nicht hier aufgelistet ist eine weitere Unterprozedur, *InsertEmptyNode*, die leere Knotenpunkte für fehlenden Inhalt ins DOM-Dokument einfügt.

Dazu gibt es im VBA-Projekt noch eine Unterprozedur, die boolesche Werte in »1« bzw. »0« konvertiert.

#### HINWEIS

Die Beispieldatei *Bsp29\_04.doc* enthält zudem eine alternative Version dieses Codes – *ListSmartTagByTypeDebug* – die die Smarttag-Informationen in das Direktfenster statt in eine XML-Datei ausgibt.

Listing 29.5

Beispielcode, um alle Smarttag-Information eines Dokuments aufzulisten

```
Sub ListSmartTagsByType()
' Dieses Makro läuft in der Word-Anwendung.
' Es benutzt das MSXML DOM, um eine XML-Datei
' mit Informationen zu den Smarttags im aktuellen
' Dokument zu erstellen, und öffnet anschließend
' das Resultat mit Anwendung einer Transformation.

Const strNamespace = "http://www.KAP29.com/BSP04"

' Passen Sie die folgenden Konstantwerte wo nötig an
Const strXMLPathName = "C:\WordBuch\Beispiele\KAP29\BSP29_04.xml"
```

**Listing 29.5** Beispielcode, um alle Smarttag-Information eines Dokuments aufzulisten (Fortsetzung)

```

Const strXSLTPathName = "c:\WordBuch\Beispiele\KAP29\BSP29_04.xsl"

Dim bIsSmartDocumentSmartTag As Boolean
Dim intIndex As Integer

Dim objSmartTags As Word.SmartTags
Dim objSmartTagActions As Word.SmartTagActions
Dim objSmartTagRecognizers As Word.SmartTagRecognizers
Dim objSmartTagType As Word.SmartTagType

Dim objNamespaceAttributeNode As IXMLDOMAttribute
Dim objNode As IXMLDOMNode
Dim objRootNode As IXMLDOMElement
Dim objSmartTagTypeNode As IXMLDOMElement
Dim objSmartTagTypesNode As IXMLDOMElement
Dim objXMLDocument As DOMDocument

On Error GoTo Err_Handler

' DOM-Dokument erstellen
Set objXMLDocument = New DOMDocument
With objXMLDocument
    .async = False
    .validateOnParse = False
    .resolveExternals = False
    .preserveWhiteSpace = True

    ' Kopfzeile und Wurzelemente erstellen; Namensraum festlegen
    Set objNode = objXMLDocument.createProcessingInstruction("xml", "version='1.0'")
    .appendChild newchild:=objNode
    Set objNode = Nothing
    Set objRootNode = objXMLDocument.createElement("root")
    objRootNode.setAttribute "xmlns", strNamespace
    .appendChild objRootNode
    Call InsertNode(objXMLDocument, objRootNode, "Dokumentpfad", ActiveDocument.Path)
    Call InsertNode(objXMLDocument, objRootNode, "Dokumentname", ActiveDocument.Name)

    Set objSmartTagTypesNode = InsertEmptyNode(objXMLDocument, objRootNode, _
                                                "ActiveSmartTagTypes")

    Set objRootNode = Nothing
    Call InsertNode(objXMLDocument, objSmartTagTypesNode, "SmartTagTypeCount", _
                    CStr(Word.Application.SmartTagTypes.Count))

    For Each objSmartTagType In Word.Application.SmartTagTypes
        With objSmartTagType
            Set objSmartTags = ActiveDocument.SmartTags.SmartTagsByType(.Name)
            If objSmartTags.Count > 0 Then

                Set objSmartTagTypeNode = InsertEmptyNode(objXMLDocument, _
                                                            objSmartTagTypesNode, "SmartTagType")
                Call InsertNode(objXMLDocument, objSmartTagTypeNode, "SmartTagTypeName", .Name)
                Call InsertNode(objXMLDocument, objSmartTagTypeNode, "FriendlyName", _
                                .FriendlyName)

                Set objSmartTagActions = .SmartTagActions
                Call ListSmartTagActions(objSmartTagActions, objXMLDocument, _

```

Listing 29.5 Beispielcode, um alle Smarttag-Information eines Dokuments aufzulisten (Fortsetzung)

```

        objSmartTagTypeNode, bIsSmartDocumentSmartTag)
    Set objSmartTagActions = Nothing

    If Not bIsSmartDocumentSmartTag Then
        Set objSmartTagRecognizers = .SmartTagRecognizers
        Call ListSmartTagRecognizers(objSmartTagRecognizers, objXMLDocument, _
            objSmartTagTypeNode)
        Set objSmartTagRecognizers = Nothing

        Call ListSmartTags(objSmartTags, objXMLDocument, objSmartTagTypeNode)
    End If
    End If
    Set objSmartTags = Nothing
End With
Set objSmartTagTypeNode = Nothing
Next
Set objSmartTagTypesNode = Nothing

'Das XML-Dokument speichern
.Save strXMLPathName
End With
Set objXMLDocument = Nothing

' Abschließend das gespeicherte XML-Dokument unter
' Anwendung einer Transformation öffnen
Documents.Open FileName:=strXMLPathName, xmltransform:=strXSLTPathName

Exit Sub

Err_Handler:
    MsgBox Err.Number & vbCr & Err.Description
End Sub

```

Listing 29.6 Untergeordnete Prozedur, die die Aktionen eines Smarttags zusammenträgt

```

Sub ListSmartTagActions(objSmartTagActions As Word.SmartTagActions, _
    objXMLDocument As DOMDocument, objSmartTagTypeNode As IXMLDOMElement, _
    bIsSmartDocumentSmartTag As Boolean)

    Dim intIndex As Integer
    Dim objSmartTagAction As Word.SmartTagAction
    Dim objActionNode As IXMLDOMElement
    Dim objActionsNode As IXMLDOMElement

    On Error GoTo Err_Handler
    bIsSmartDocumentSmartTag = False
    If objSmartTagActions.Count > 0 Then
        Set objActionsNode = InsertEmptyNode(objXMLDocument, objSmartTagTypeNode, "Actions")
        Call InsertNode(objXMLDocument, objActionsNode, "ActionCount", _
            Str(objSmartTagActions.Count))
        For intIndex = 1 To objSmartTagActions.Count
            On Error Resume Next
            Set objSmartTagAction = objSmartTagActions.Item(intIndex)
            If Err.Number = 6116 Then
                Err.Clear
            End If
        Next
    End If

```

**Listing 29.6** Untergeordnete Prozedur, die die Aktionen eines Smarttags zusammenträgt (Fortsetzung)

```

        Set objSmartTagAction = Nothing
        Set objActionNode = InsertEmptyNode(objXMLDocument, objActionsNode, "Action")
        Call InsertNode(objXMLDocument, objActionNode, "ActionName", _
            "[Smart Document Action]")
        Call InsertNode(objXMLDocument, objActionNode, "ActionType", "Task Pane Control")
        Set objActionNode = Nothing
        bIsSmartDocumentSmartTag = True
        On Error GoTo Err_Handler
        Exit For
    Else
        On Error GoTo Err_Handler
        If Err.Number = 0 Then
            Set objActionNode = InsertEmptyNode(objXMLDocument, objActionsNode, "Action")
            With objSmartTagAction
                Call InsertNode(objXMLDocument, objActionNode, "ActionName", .Name)
                Select Case .Type
                    Case wdControlSmartTag
                        Call InsertNode(objXMLDocument, objActionNode, "ActionType", "Smart Tag")
                    Case Else ' Diese Aktionen kommen nur in Smarttags als Teil einer
                        ' Smart Document-Lösung vor (Smarttags im
                        ' Aufgabenbereich "Dokumentaktionen").
                        Call InsertNode(objXMLDocument, objActionNode, "ActionType", _
                            "Task Pane Control")
                        bIsSmartDocumentSmartTag = True
                End Select
                Set objActionNode = Nothing
                Set objSmartTagAction = Nothing
            End With
        End If
    End If
Next
Set objActionsNode = Nothing
End If
Exit Sub

Err_Handler:
    MsgBox Err.Number & vbCrLf & Err.Description
End Sub

```

**Listing 29.7** Untergeordnete Prozedur, die einen Knotenpunkt für ein Kinderelement mit Textinhalt in dem DOM-Dokument erstellt

```

Private Sub InsertNode(objDOMDocument As DOMDocument, objParentNode As IXMLDOMNode, _
    strName As String, strValue As String)

    Dim objNode As IXMLDOMNode
    Set objNode = objDOMDocument.createElement(strName)
    objNode.Text = strValue
    objParentNode.appendChild objNode
    Set objNode = Nothing
End Sub

```



**WICHTIG** Falls Sie versuchen, dieses Beispiel auf alle installierten Smarttags für den gegenwärtigen Anwender zu erweitern, könnten Sie Problemen begegnen, wenn XML Expansion Packs für Smart Document-Lösungen installiert sind (siehe Kapitel 30).

Die folgenden Probleme tauchen auf, wenn Sie die Smarttag-»Recognizers« oder -Aktionen eines Smarttags ansprechen, das einer Smart Document-Lösung gehört:

- Beim Ansprechen von Elementen der SmartTagActions-Auflistung wird der Laufzeitfehler 6116 (Anwendungs- oder objektdefinierter Fehler) ausgelöst.
- Beim Ansprechen von Eigenschaften der SmartTagRecognizers-Auflistung (.Count, etwa) bleibt die Word-Anwendung hängen.

Da dieses Beispiel nur Smarttags im Dokument anspricht, kann das Problem nur vorkommen, wenn ein XML Expansion Pack mit dem Dokument verbunden ist und Smarttags, die Teil der Smart Document-Lösung sind, sich im Dokument befinden. In diesem Fall ist es möglich, zuerst die Actions-Auflistung zu bearbeiten, den Laufzeitfehler abzufangen und dann eine Bearbeitung dieser »Recognizers« zu unterlassen.

## Zusammenfassung

Dieses Kapitel hat Smarttags aus Office 2003 beschrieben und stellte u.a. vor:

- Smarttags in der Benutzeroberfläche (Seite 912 ff.)
- Konzeptgrundlagen (Seite 914 ff.)
- Die Entwicklung von MOSTL-Smarttags (Seite 917 ff.)
- Die Entwicklung einer Smarttag-DLL in VB6 (Seite 924 ff.)
- Die Smarttag-Funktionalität im Word-Objektmodell (Seite 932 ff.)



## Kapitel 30

# Smart Documents

**In diesem Kapitel:**

Was ist ein Smart Document?	944
Ein MOSTL-Smart Document	945
Eine Smart Document-DLL	966
Das Erweiterungspaket-Manifest digital signieren	994
Zusammenfassung	995

Mit Smart Documents (»intelligente Dokumente«) steht Entwicklern eine Funktionalität zur Verfügung, bei der kontextbezogene Werkzeuge eine ganz neue Art der Interaktion zwischen Anwender und Dokument ermöglichen. Sie basieren auf den Technologien XML und *Aufgabenbereiche (Task Panes)*.

Microsoft stellt die Smart Document-Technologie bislang für Word 2003 sowie Excel 2003 bereit. In diesem Buch stellen wir jedoch nur die Smart Document-Technologie für Word 2003 vor.

#### HINWEIS

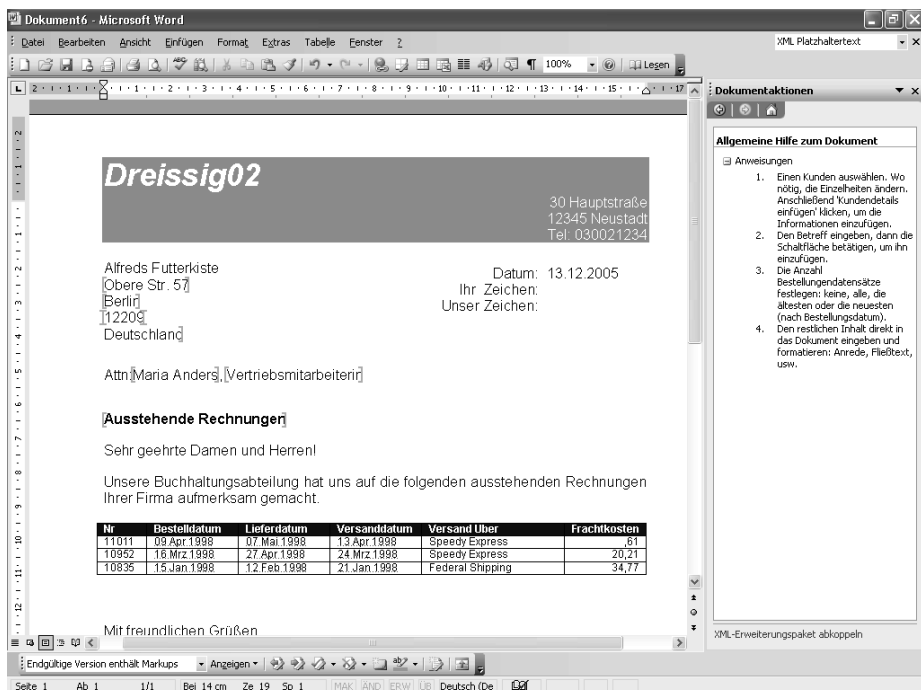
Smart Documents sind die einzige Möglichkeit, eigene Aufgabenbereiche mit COM-Anwendungen zu erstellen. Visual Studio for Office 2005 (VSTO 2005) besitzt zwar eine verbesserte Funktionalität zur Erstellung von Aufgabenbereichen, steht jedoch nur .NET-Entwicklern zur Verfügung. VSTO 2005 wird in diesem Buch nicht behandelt.

## Was ist ein Smart Document?

In Kapitel 29 wurden die Smarttags vorgestellt und näher beschrieben. Sie bieten Aktionsmenüs für Text an, der durch einen »Recognizer« erkannt wurde. Zweck dieser Smarttags ist es, dem Anwender für den so erkannten Text bestimmte vordefinierte Aktionen bereitzustellen, ohne dass er den Mauszeiger bewegen oder den ganzen Bildschirm durchsuchen muss.

Hinter Smart Documents steckt eine ähnliche Idee. Jedoch werden hier nicht Textstellen im Dokument, sondern XML-Elemente (auch als XML-Tags bezeichnet – siehe Kapitel 28) erkannt, an denen der Anwender gegenwärtig arbeitet bzw. in denen sich die Einfügemarke gerade befindet.

**Abbildg. 30.1** Eine Smart Document-Lösung bietet dem Anwender kontextbezogene Hilfe und Werkzeuge im Aufgabenbereich *Dokumentaktionen*.



Die Hauptunterschiede zwischen beiden Techniken können wie folgt zusammengefasst werden:

- Die Aktionen eines Smart Documents sind mit XML-Tags im Dokument verbunden, anstatt mit den vom »Recognizer« erkannten Textstellen.
- Die für das gerade markierte XML-Element zur Verfügung stehenden Smart Document-Aktionen sind im Aufgabenbereich immer direkt sichtbar; der Anwender muss kein Menü öffnen, um die Aktionen zu sehen.
- Es steht dem Entwickler ein breiteres Spektrum an Aktionen mit dazu passenden Steuerelementen zur Verfügung. Diese umfassen beispielsweise Schaltflächen, Hilfetexte, Optionsschaltflächen, Kontrollkästchen, Listenfelder, Comboboxen, Grafiken, Dokumentausschnitte, Trennlinien, Hyperlinks sowie Unterstützung für weitere, selbst definierte ActiveX-Steuerelemente.

Der folgende Abschnitt zeigt anhand eines einfachen Beispiels, wie ein Smart Document mit der in Kapitel 29 vorgestellten MOSTL-Technologie (*Microsoft Office Smart Tag List*) entwickelt werden kann. Die Möglichkeiten einer MOSTL-Lösung sind jedoch begrenzt gegenüber denen einer COM-DLL, die mit einer Programmiersprache wie Visual Basic Classic (VB6), VB.NET oder C# erstellt werden kann (siehe den Abschnitt »Eine Smart Document-DLL« weiter hinten in diesem Kapitel).

### Das Smart Document SDK

Das Smart Document SDK (auf der CD-ROM zum Buch im Ordner *\Beilagen\Smart Document SDK* oder im Internet unter <http://www.microsoft.com/downloads/details.aspx?FamilyID=24a557f7-eb06-4a2c-8f6c-2767b174126f&DisplayLang=en>) ist für das Verständnis und die Entwicklung von Smart Documents unentbehrlich, bislang aber nur in englischer Sprache erhältlich und enthält:

- Einführungs- sowie konzeptuelle Erläuterungen,
- die Dokumentation der ISmartDocument- sowie ISmartDocProperties-Interfaces,
- die Dokumentation mehrerer XML-Schemata, u.a. das »XML Expansion Pack Manifest Schema« sowie das »MOSTL Smart Tag List/Smart Document Schema«,
- Beispiele in mehreren Programmiersprachen, u.a. VB6, VB.NET sowie C#,
- eine Dokumentation zum Thema Verteilen und Installation,
- Werkzeuge, um die XML Expansion Pack Manifest-Sicherheit während der Entwicklungsphase auszuschalten und für Testzwecke eigene Zertifikate zu erstellen.

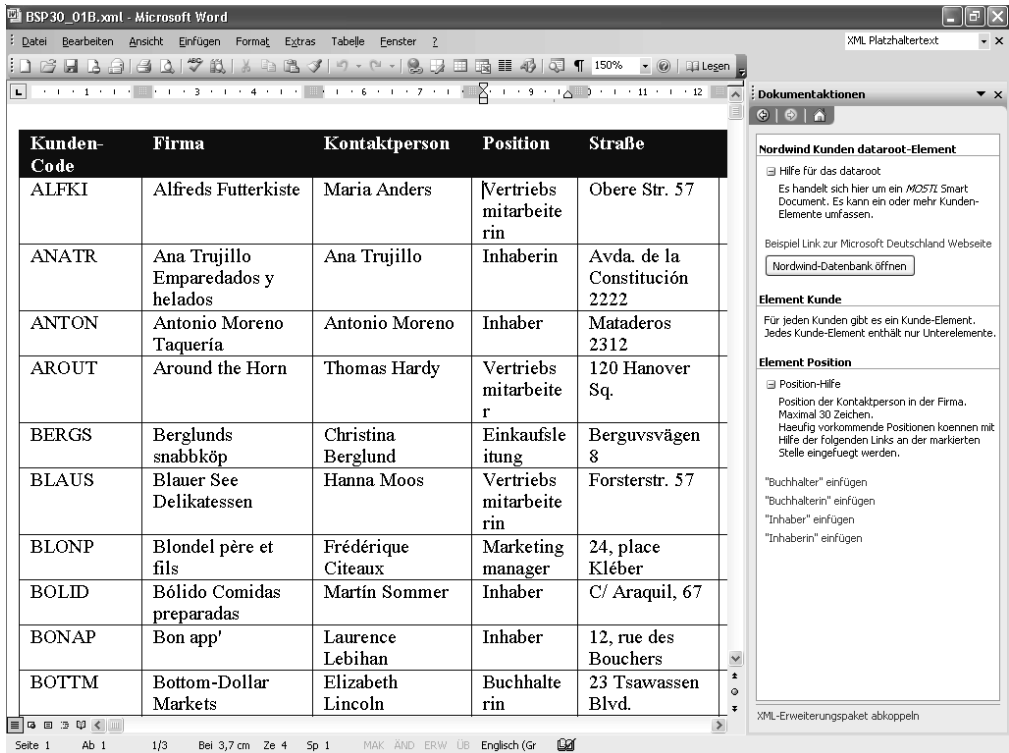
## Ein MOSTL-Smart Document

Wie bei MOSTL-Smarttags ist auch die Funktionalität der MOSTL-Smart Documents eher begrenzt. Sie können im Aufgabenbereich *Dokumentaktionen* nur

- kontextbezogene Hilfe anbieten,
- Links zu Dateien und Webseiten bereitstellen,
- über Links häufig verwendete Werte und Textfragmente ins Dokument einfügen (anstelle der gegenwärtigen Markierung).

Das Beispiel in diesem Abschnitt zeigt, wie Datensätze einer Datenbank für die Bearbeitung in der dem Anwender vertrauten Word-Umgebung angezeigt werden können (Abbildung 30.2). Dank des XML-Dateiformats ist es einfach, die Daten zuerst aus der Datenbank zu exportieren und das bearbeitete Ergebnis anschließend wieder zu importieren. Wie in Kapitel 28 beschrieben, werden XML-Elemente in Word-Tabellen beim Einfügen neuer Zeilen automatisch dupliziert, so dass der Anwender problemlos Datensätze hinzufügen kann.

**Abbildg. 30.2** In eine Word-Tabelle exportierte XML-Daten aus einer Access-Datenbank



The screenshot shows a Microsoft Word document titled 'BSP30\_01B.xml - Microsoft Word'. The main content is a table with 5 columns: Kunden-Code, Firma, Kontaktperson, Position, and Straße. The table contains 12 rows of customer data. On the right side, there is a 'Dokumentaktionen' (Document Actions) task pane. It contains sections for 'Nordwind Kunden dataroot-Element', 'Element Kunde', and 'Element Position', each with instructions and links for inserting data into the document.

Kunden-Code	Firma	Kontaktperson	Position	Straße
ALFKI	Alfreds Futterkiste	Maria Anders	Vertriebsmitarbeiterin	Obere Str. 57
ANATR	Ana Trujillo Emparedados y helados	Ana Trujillo	Inhaberin	Avda. de la Constitución 2222
ANTON	Antonio Moreno Taquería	Antonio Moreno	Inhaber	Mataderos 2312
AROUT	Around the Horn	Thomas Hardy	Vertriebsmitarbeiter	120 Hanover Sq.
BERGS	Berglunds snabbköp	Christina Berglund	Einkaufsleitung	Berguvsvägen 8
BLAUS	Blauer See Delikatessen	Hanna Moos	Vertriebsmitarbeiterin	Forsterstr. 57
BLONP	Blondel père et fils	Frédérique Citeaux	Marketing manager	24, place Kléber
BOLID	Bólido Comidas preparadas	Martin Sommer	Inhaber	C/ Araquil, 67
BONAP	Bon app'	Laurence Lebihan	Inhaber	12, rue des Bouchers
BOTTM	Bottom-Dollar Markets	Elizabeth Lincoln	Buchhalterin	23 Tsawassen Blvd.

## Das MOSTL-Beispiel installieren

In diesem Abschnitt wird beschrieben, wie Sie ein MOSTL-Smart Document zu Testzwecken installieren.

### WICHTIG

Wir empfehlen unbedingt, die auf der CD-ROM zum Buch enthaltenen Beispieldateien zu installieren, um den Angaben in diesem Abschnitt besser folgen zu können.



Alle in Tabelle 30.1 benötigten Beispieldateien befinden sich auf der CD-ROM zum Buch im Ordner `\Beispiele\Kap30`. Die Beispiel-Datenbank `Nordwind.mdb` finden Sie im Ordner `\Beispiele\Datenbank`.

Tabelle 30.1 Beispieldateien für den Abschnitt über MOSTL-Smart Documents

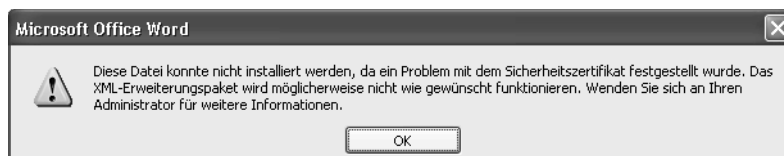
Beispieldatei	Beschreibung
<i>Bsp30_01A.xml</i>	Ein Word-Dokument mit einem <b>Kunden</b> -Datensatz in einem benutzerdefinierten XML-Vokabular.
<i>Bsp30_01B.xml</i>	Eine erweiterte Version von <i>Bsp30_01A.xml</i> , die mehr Datensätze enthält und mit einem Erweiterungspaket verbunden ist.
<i>Bsp30_01.xsd</i>	Das Schema für <i>Bsp30_01A.xml</i> und <i>Bsp30_01B.xml</i> .
<i>Bsp30_01.xsl</i>	Eine Transformation für <i>Bsp30_01A.xml</i> und <i>Bsp30_01B.xml</i> .
<i>Bsp30_01S.xml</i>	Enthält eine Smart Document-Solution (Lösung); definiert, welche Elemente des Schemas im Aufgabenbereich <i>Dokumentaktionen</i> repräsentiert sind, und mit welchen Steuerelementen.
<i>Bsp30_01M.xml</i>	Ein XML Erweiterungspaket-Manifest, das die zu einer Lösung gehörenden Dateien auflistet; enthält in diesem Fall nur die Informationen zur MOSTL-Datei <i>Bsp30_01S.xml</i> .
<i>Bsp30_01N.xml</i>	Enthält Informationen zur MOSTL-Datei, zum Schema sowie zur Transformation.
<i>Bsp30_01.reg</i>	Schaltet die Sicherheit für Erweiterungspakete aus.

Gehen Sie zur Installation folgendermaßen vor:

1. Kopieren Sie alle genannten Beispieldateien – mit Ausnahme der *Nordwind.mdb* – in einen Ordner auf Ihrem Rechner. Im Beispiel wird der Ordner *C:\Beispiele\Kap30* verwendet, den Sie auch anlegen sollten. Die Datenbank *Nordwind.mdb* kopieren Sie bitte in den Ordner *C:\Beispiele\Datenbank*.
2. Starten Sie Word, so dass ein neues leeres Dokument angezeigt wird, oder erstellen Sie ein neues Dokument. Öffnen Sie anschließend im Menü *Extras/Vorlagen und Add-Ins* das gleichnamige Dialogfeld und wechseln zur Registerkarte *XML-Schema*.
3. Klicken Sie auf die Schaltfläche *Schema hinzufügen* und wählen Sie aus dem Ordner *C:\Beispiele\Kap30* (bzw. dem von Ihnen verwendeten Ordner) die Datei *Bsp30\_01.xsd*. Verwenden Sie als *Alias* *Bsp01* (oder eine andere Bezeichnung, wenn der vorgeschlagene Name bereits einem anderen Schema zugewiesen wurde). Bestätigen Sie die Angaben mit *OK* und überprüfen Sie unbedingt, ob der Schemaeintrag auf der Registerkarte *XML-Schema* aktiviert ist.
4. Wechseln Sie zur Registerkarte *XML-Erweiterungspakete*. Klicken Sie dort auf die Schaltfläche *Hinzufügen* und öffnen Sie die Datei *Bsp30\_01M.xml*.

**ACHTUNG** Word wird vermutlich die in Abbildung 30.3 angezeigte Meldung einblenden, dass die Datei wegen Problemen mit der Sicherheitsprüfung nicht geöffnet werden konnte. Bestätigen Sie den Warnhinweis über *OK*.

Abbildg. 30.3 Das XML-Erweiterungspaket ist nicht digital signiert



Die Meldung erscheint, weil das verwendete XML-Erweiterungspaket (»XML Expansion Pack«) nicht digital signiert ist. Um das Paket zu installieren, muss es entweder ein *Code Signing Digital Certificate* (siehe den Abschnitt »Das Erweiterungspaket-Manifest digital signieren« in diesem Kapitel) besitzen, oder ein Eintrag in der Windows-Registrierung muss geändert werden, um die Sicherheit zu Testzwecken auszuschalten.

Um den Eintrag in der Windows-Registry zu ändern, wechseln Sie im Explorer in den Ordner mit den Beispieldateien (das Dialogfeld in Word kann geöffnet bleiben) und führen per Doppelklick die Datei *Bsp30\_01.reg* aus. Bestätigen Sie die Frage, ob die Informationen in der Datei *Bsp30\_01.reg* der Registrierung hinzugefügt werden sollen, über die Schaltfläche *Ja*.

Mit dieser Datei wird folgender Eintrag der Registrierung hinzugefügt bzw. geändert:

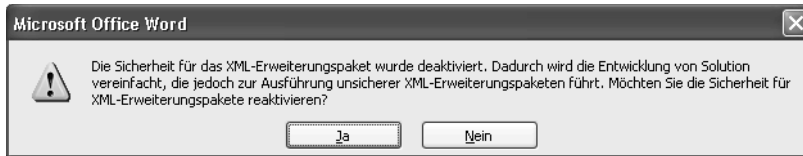
`[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Office\Common\Smart Tag]`

Der Schlüssel *DisableManifestSecurityCheck* wird dabei auf den Wert »1« festgelegt.

5. Wechseln Sie zurück zu dem geöffneten Dialogfeld in Word und klicken Sie noch einmal auf die Schaltfläche *Hinzufügen*. Nach Auswahl der Datei *Bsp30\_01M.xml* weist eine Warnmeldung (Abbildung 30.4) darauf hin, dass die Sicherheit für XML-Erweiterungspakete ausgeschaltet ist, und fragt, ob die Sicherheit wieder eingeschaltet werden soll.

Abbildg. 30.4

Standardmäßig ist die Schaltfläche *Ja* aktiviert, was die Festlegung des Registry-Schlüssels rückgängig machen würde.



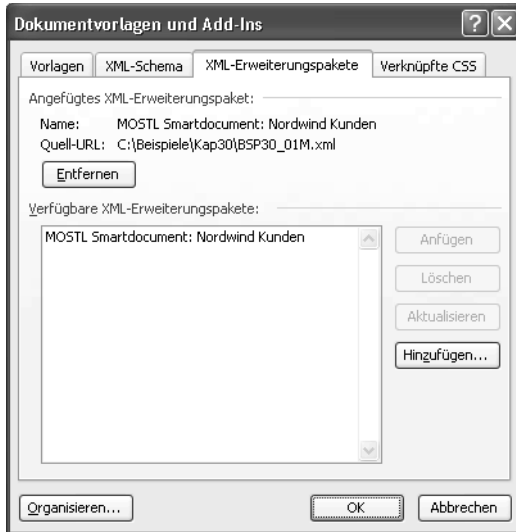
6. Lehnen Sie die Reaktivierung der Sicherheit unbedingt über die Schaltfläche *Nein* ab.
7. Um das Erweiterungspaket anschließend dem aktuellen Dokument anzufügen, klicken Sie auf die Schaltfläche *Anfügen*. Dies ist nur möglich, weil das Schema mit dem gleichen Namensraum in der Registerkarte *XML-Schema* aktiviert ist. Nachdem Sie das Erweiterungspaket der Datei angefügt haben, sollte das Dialogfeld wie in Abbildung 30.5 aussehen: Die Felder hinter *Name*: und *Quell-URL*: enthalten nun die entsprechenden Angaben für dieses Erweiterungspaket. Über die Schaltfläche *Entfernen* kann das Erweiterungspaket ggf. entfernt werden.

#### PROFITIPP

Während der Entwicklung einer Smart Document-Lösung können Sie eine neue Version der Lösung laden, indem das XML-Erweiterungspaket in diesem Dialogfeld aus der Liste gelöscht, dann neu hinzu- und angefügt wird. Es ist nicht notwendig, wie bei Smarttags, Word zu beenden und neu zu starten. Die Schaltfläche *Aktualisieren* ist in diesem Zusammenhang **nicht** wirksam.



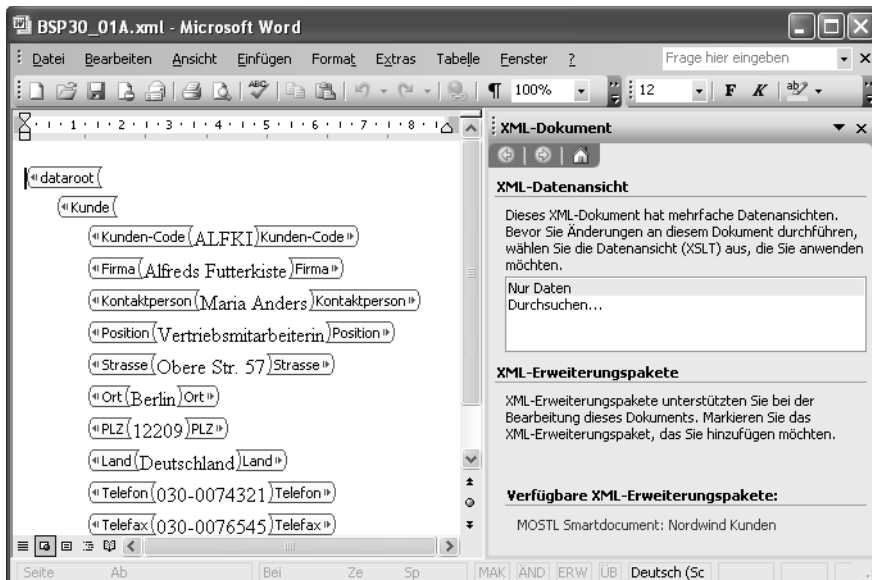
Abbildg. 30.5 Das XML-Erweiterungspaket wurde dem aktuellen Dokument angefügt.



8. Schließen Sie das Dialogfeld *Dokumentvorlagen und Add-Ins* mit **OK**.

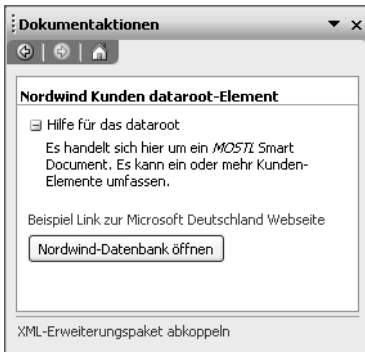
Wird nun eine XML-Datei (*Bsp30\_01A.xml*) geöffnet, die durch ihren URI auf das Schema (aber nicht das Erweiterungspaket) verweist, erscheint sie in Word wie in Abbildung 30.6 ersichtlich. Ein Link zum Erweiterungspaket befindet sich im unteren Teil des Aufgabenbereichs *XML-Dokument*. Wird er angeklickt, erscheint der Aufgabenbereich *XML-Aktionen*.

Abbildg. 30.6 XML-Dokument – verbunden mit dem Schema, jedoch nicht mit dem Erweiterungspaket



Falls die Einfügemarke sich außerhalb des XML-Teils des Dokuments befindet, ist dieser leer. Wird sie rechts des Elements `dataroot` gesetzt, erscheinen die diesem Element zugewiesenen Smart Document-Elemente, siehe Abbildung 30.7. Die Angaben in diesem Dokumentaktionen-Abschnitt gelten für das ganze XML-Dokument und bleiben sichtbar, egal, wo man sich im XML befindet.

**Abbildg. 30.7** Der Smart Document-Abschnitt für das Wurzelement `dataroot`



Es handelt sich um:

- einen Abschnitt für die Hilfe. Durch Klicken auf das Symbol links kann der Inhalt dieses Abschnitts ausgeblendet bzw. wieder eingeblendet werden.
- eine »Link action«. Der Hyperlink in diesem Beispiel öffnet die Webseite von Microsoft Deutschland in einem Browser-Fenster.
- eine Schaltfläche (»Button action«) *Nordwind-Datenbank öffnen*. Diese funktioniert wie ein Hyperlink und öffnet in diesem Beispiel die Beispieldatenbank *Nordwind.mdb* in einem Access-Fenster. (Falls die Datenbank sich nicht im erwarteten Speicherort *file:///C:/Beispiele/Datenbank* befindet, wird eine entsprechende Fehlermeldung eingeblendet.)

Die nächste Hierarchie-Ebene dieses XML-Dokuments bildet das Element `Kunde`. Befindet sich die Einfügemarke rechts neben diesem Tag, sieht der Aufgabenbereich wie in Abbildung 30.8 aus. Unter dem Abschnitt für das Wurzelement findet sich einer für dieses Element. Er enthält lediglich Informationen zum erwarteten Elementinhalt.

Wenn Sie der Reihe nach die weiteren, sich in der nächsten Ebene befindenden Elemente anwählen, wird im Aufgabenbereich ein dritter Abschnitt eingeblendet. Dessen Inhalt variiert je nach Element; meistens handelt es sich um Hilfe-Texte oder Beschriftungen. Die Abbildung 30.9 zeigt die Angaben für das Datenfeld `Position`.

Die hier aufgelisteten Links fügen Text an der aktuellen Position der Einfügemarke ein, eventuell markierter Text wird ersetzt. Die Links im Beispiel sollen nur die entsprechende Möglichkeit aufzeigen; ansonsten wird Text selten über einen Link ins Dokument eingefügt.

Abbildg. 30.8 Der Aufgabenbereich *Dokumentaktionen* mit zwei eingeblendeten Hierarchie-Ebenen

**HINWEIS** Die Abbildung 30.9 veranschaulicht zudem, welche Probleme die Technologie noch mit Unicode hat. Umlaute werden allgemein, aber nicht überall unterstützt. Im Hilfetext für die Position wird UTF-8 nicht korrekt interpretiert, weshalb die Umlaute mit den Hilfszeichenfolgen angegeben werden mussten.

Abbildg. 30.9 Der Aufgabenbereich mit einer dritten Hierarchie-Ebene für das Element *Position*

Nach dem vorherigen Überblick zu den Möglichkeiten einer MOSTL-Smart Document-Lösung wird im folgenden Abschnitt dargelegt, wie das Ergebnis in Abbildung 30.2 erreicht wurde und wie der Anwender die Lösung nutzen kann, um neue Datensätze zu erfassen.

## Eine benutzerfreundlichere Lösung


Die eingangs beschriebene Lösung stellt Daten in einer Tabelle zur weiteren Bearbeitung bereit; die XML-Tags sind nicht sichtbar. Falls Sie das Kapitel 28 gelesen haben, werden Sie – zu Recht – denken, dafür sei eine XSL-Transformation notwendig, wie im dortigen Abschnitt »Transformationen und Lösungen (Solutions)« beschrieben.

### TIPP

Hier eine Zusammenfassung der Aussage in Kapitel 28: Die benutzerdefinierten Tags müssen sich außerhalb der WordProcessingML-Elemente für die entsprechenden Tabellenteile befinden und diese umgeben:

```
<Kunde>
  <w:tr>
    <Kunden-Code>
      <w:tc>
        </w:tc>
      </Kunden-Code>
    </w:tr>
  </Kunde>
```

Jede Tabellenzeile enthält einen Kunde-Datensatz; jede Zelle ein Datenfeld (Kindelement).

Beim automatischen Erstellen einer neuen Tabellenzeile durch Drücken der -Taste am Tabellenende wird ein zusätzlicher Satz des Elements Kunde erzeugt, mit allen Unterelementen, wie in der vorherigen Zeile definiert.

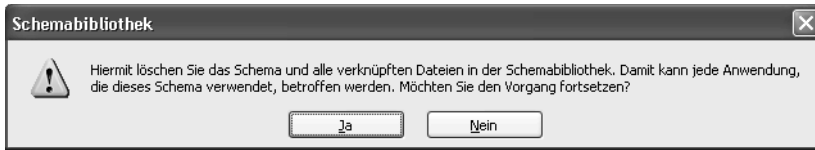
Nach den bisherigen Kenntnissen müsste der Benutzer, um die Lösung zu installieren, neben dem Schema und Erweiterungspaket auch die XSLT über *Extras/Vorlagen- und Add-Ins/XML-Schema/Schema-Bibliothek* importieren. Ist das Ganze nicht eher mit mehr Aufwand als mit Arbeiterleichterung verbunden?

Die Antwort liegt im Erweiterungspaket. Das Dokument kann so mit einem Erweiterungspaket verbunden werden, dass sich dieses, zusammen mit allen weiteren benötigten Bestandteilen, beim Öffnen des Dokuments installiert. Somit muss sich der Benutzer mit den Registerkarten unter *Extras/Vorlagen und Add-Ins* gar nicht erst abgeben, sondern nur eine einzige Frage beantworten, wie im folgenden Vorgang beschrieben. Um diesen mitzuverfolgen, gehen Sie wie folgt vor:

1. Entfernen Sie das im Abschnitt »Das MOSTL-Beispiel installieren« installierte Schema und Erweiterungspaket. Öffnen Sie dazu die Schema-Bibliothek über die Befehlsfolge *Extras/Vorlagen und Add-Ins/XML-Schema*, klicken Sie auf die Schaltfläche *Schemabibliothek*, markieren Sie im daraufhin geöffneten Dialogfeld *Schemabibliothek* das Schema und wählen Sie *Schema löschen*. Die Nachfrage aus Abbildung 30.10 bestätigen Sie mit *Ja*. Damit werden auch die mit dem Schema verbundenen Transformationen und Erweiterungspakete gelöscht.

Abbildg. 30.10

Alle mit dem Schema verbundenen Lösungsdateien werden ebenfalls gelöscht.



2. Bestätigen Sie das Dialogfeld *Schemabibliothek* mit OK. Stellen Sie in der Registerkarte *XML-Schema* sicher, dass das Kontrollkästchen für das Schema entweder entfernt wurde oder als »Nicht verfügbar« bezeichnet ist, klicken Sie dann auf OK.
3. Schließen Sie eventuell geöffnete XML-Dateien.
4. Öffnen Sie nun die Beispieldatei *Bsp30\_01B.xml*. Word blendet das Dialogfeld aus Abbildung 30.11 ein.

Abbildg. 30.11

Erweiterungspaket, um alle verbundenen Schemas und Transformationen in einem Schritt zu installieren




5. Aktivieren Sie das Kontrollkästchen *Als Standardauswahl für dieses Dokument festlegen* und bestätigen Sie die Frage mit *Ja*. Die Sicherheitsmeldung in Abbildung 30.4 müsste erscheinen; beantworten Sie diese mit *Nein*. Es werden kurz verschiedene Dialogfelder eingeblendet, bevor das Dokument schließlich auf dem Bildschirm neben dem Aufgabenbereich *XML-Dokument* bereitsteht.

#### HINWEIS

Eine Smart Document-Lösung wird oft auch zentral auf einem Server bereitgestellt. Bei der Installation des Erweiterungspakets werden die Komponenten vom Server heruntergeladen und lokal auf dem Rechner installiert.

6. Klicken Sie unten im Aufgabenbereich auf den Link *MOSTL Smartdocument:Nordwind Kunden*. Sie sehen das Dokument, wie es in Abbildung 30.2 vorliegt, mit dem Aufgabenbereich *Dokumentaktionen* in Abbildung 30.7.

Wenn Sie in die Tabelle klicken und die Einfügemarke durch die Zelle bewegen, werden Sie die verschiedenen Inhalte des Aufgabenbereichs wieder erkennen.

Bewegen Sie die Einfügemarke in die letzte Tabellenzelle und drücken Sie die -Taste: Word erstellt eine neue Tabellenzelle mit den gleichen Dokumentaktionen wie in den anderen Zeilen. Sofern das Kontrollkästchen *XML-Tag im Dokument anzeigen* des Aufgabenbereichs *XML-Struktur* aktiviert ist, wird beim Betrachten der Zeile ersichtlich, dass diese auch die gleichen Tags enthält.

**HINWEIS** Bis Word die Tabellenzeile anzeigt, können einige Sekunden vergehen. Je größer die Tabelle ist, desto länger muss gewartet werden.

Versuchen Sie nun das Dokument zu schließen, wird eine Warnung eingeblendet, dass das Dokument wegen einer Verletzung der Schema-Struktur nicht als XML gespeichert werden kann. Bei näherer Betrachtung des Aufgabenbereichs *XML-Struktur* wird erkennbar, dass das Problem im Kunden-Code-Element der letzten Tabellenzeile liegt; entsprechend erscheint eine feine, rot gepunktete Linie am linken Zellenrand. Ein Rechtsklick auf den fehlbaren Eintrag im Aufgabenbereich teilt mit, dass das Element einen Eintrag eines bestimmten Musters enthalten muss, was dem Anwender wahrscheinlich eher rätselhaft vorkommen würde.

**HINWEIS** Falls die rote Linie neben der Tabellenzelle nicht sichtbar ist, klicken Sie auf den Link *XML-Optionen* zuunterst im Aufgabenbereich *XML-Struktur* und deaktivieren die Option *Schemaverletzungen in diesem Dokument ausblenden*. Erscheinen im Aufgabenbereich keine Fehlersymbole, stellen Sie sicher, dass die Option *Dokument gegen angefügte Schemas prüfen* aktiviert ist.

Im Aufgabenbereich *Dokumentaktionen* steht für das Kunden-Code-Element beschrieben, dass eine aus fünf Buchstaben bestehende Bezeichnung erforderlich ist. Geben Sie einen entsprechenden Eintrag, wie »AAAAA«, ein und versuchen Sie nochmals, das Dokument zu speichern. Die Meldung in Abbildung 30.12 wird eingeblendet.

Abbildg. 30.12 Die XML-Datei kann auch in einem anderen Format als WordProcessingML gespeichert werden.

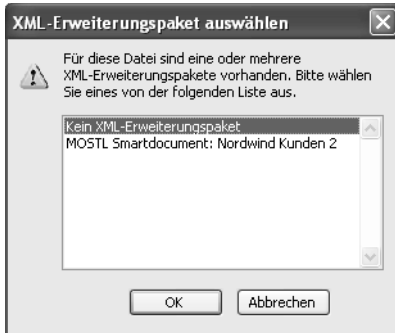


Laienhaft ausgedrückt: Das Dokument wird im Format WordProcessingML gespeichert, wenn Sie die Meldung mit *Ja* beantworten. Klicken Sie stattdessen auf *Nein*, erscheint das Dialogfeld *Speichern unter*, in dem Sie das Kontrollkästchen *Nur Daten speichern* aktivieren können.

Wird ein Dokument mit angefügtem Erweiterungspaket in WordProcessingML oder als Word-Dokument (\*.doc) gespeichert, müsste das Erweiterungspaket beim erneuten Öffnen nicht wieder angefügt werden. Meist geht jedoch diese Verknüpfung beim Speichern mit aktiviertem Kontrollkästchen *Nur Daten speichern* verloren und das Erweiterungspaket muss explizit (über das Dialogfeld) wieder angefügt werden.

**HINWEIS** Gelegentlich wird beim Öffnen eines Word-Dokuments das Dialogfeld aus Abbildung 30.13 eingeblendet. Das Microsoft Office 2003 Smart Document SDK erklärt, unter welchen Umständen Word ein Erweiterungspaket nicht automatisch anfügt.

**Abbildg. 30.13** Dieses Dialogfeld erscheint, wenn mehr als ein installiertes Erweiterungspaket für ein Dokument benutzt werden kann.



## Wie es funktioniert

Die Daten sowie das Schema für das Beispiel wurden in Access über *Datei/Exportieren* im XML-Format angelegt. Durch Anpassen des Ergebnisses wurden die Beispieldateien erstellt.

*Bsp30\_01A.xml*

Für *Bsp30\_01A.xml* wurden alle Datensätze außer einem entfernt. Die Elementnamen Kunden sowie Straße wurden in Kunde bzw. Strasse geändert. Letztlich wurden die von Access angelegten Namensräume durch eigene ersetzt und das Layout ansprechender gestaltet. Das Listing 30.1 veranschaulicht den Inhalt der Datei.

**Listing 30.1** Inhalt der Beispieldatei *Bsp30\_01A.xml* mit einem *Kunde*-Datensatz

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<dataroot
  xmlns="http://www.KAP30Beispiel.com/BSP01">
  <Kunde>
    <Kunden-Code>ALFKI</Kunden-Code>
    <Firma>Alfreds Futterkiste</Firma>
    <Kontaktperson>Maria Anders</Kontaktperson>
    <Position>Vertriebsmitarbeiterin</Position>
    <Strasse>Obere Str. 57</Strasse>
    <Ort>Berlin</Ort>
    <PLZ>12209</PLZ>
    <Land>Deutschland</Land>
    <Telefon>030-0074321</Telefon>
    <Telefax>030-0076545</Telefax>
  </Kunde>
</dataroot>
```

*Bsp30\_01B.xml*

Das zweite Beispieldokument *Bsp30\_01B.xml* enthält mehr Datensätze sowie eine Verarbeitungsanweisung. Diese weist Word an, das Erweiterungspaket (das sich in diesem Beispiel im gleichen Ordner mit dem XML-Dokument befinden muss) auch zu öffnen.

```
<?mso-solutionextension
  URI="http://www.KAP30Beispiel.com/BSP01" manifestPath="BSP30_01N.xml"?>
```

Bsp30\_01.xsd  
(das Schema)

Wie erwähnt, wurde das Schema durch die Export-Funktionalität von Access automatisch generiert. Da es eine zweidimensionale Tabelle beschreibt, ist es recht unkompliziert; der Code wird hier also nicht reproduziert. Das Schema wurde für diese Lösung wie folgt angepasst:

- Alle Referenzen zum Namensraum »od:« wurden entfernt.
- Der Namensraum für dieses Projekt wurde eingefügt.
- Die Namen der Elemente Kunden sowie Straße wurden in Kunde bzw. Strasse geändert.
- Ein Mustervergleich für das Element Kunden-Code (fünf Buchstaben) wurde hinzugefügt, so dass die Angaben für den Primärschlüssel den Anforderungen der Nordwind-Datenbank entsprechen.
- Das Layout wurde ansprechend gestaltet.

Bsp30\_01.xsl  
(die Transform)

Die XSL-Transformation wurde mit dem »Microsoft Office 2003 WordProcessingML Transform Inference Tool« des »Microsoft Office Word 2003 XML SDK« (siehe Kapitel 28) erstellt. Das Ergebnis musste angepasst werden, weil es annimmt, dass jedes Feld in jedem Datensatz über Inhalt verfügt, so dass jedes Kunde-Element gleichviel Kindelemente hätte. Dies ist jedoch nicht der Fall. Beispielsweise fehlen in vielen Datensätzen Angaben für das Feld Region; dafür wurden beim Export der Daten nach XML keine Elemente generiert. Bei der Transformation hätten diese Tabellenzeilen weniger Zellen als jene, die ein Region-Element haben.

Das Listing 30.2 enthält den vom »Inference Tool« erstellten Code, der für die Verarbeitung des Kunde-Elements zuständig ist. Wie der `<xsl:apply-template>`-Anweisung und dem Code in Listing 30.3 zu entnehmen ist, werden alle Kindelemente identisch verarbeitet. Fehlt in der XML-Datei ein Element dieser Liste, wird dafür keine Tabellenzeile erstellt.

**Listing 30.2** Der vom »Inference Tool« erstellte `<xml-template>`-Code für das *Kunde*-Element

```
<xsl:template match="/ns2:dataroot/ns2:Kunde">
  <ns2:Kunden>
    <xsl:for-each select="@ns2:*|@*[namespace-uri()='']">
      <xsl:attribute name="{name()}" namespace="{namespace-uri()}">
        <xsl:value-of select="." />
      </xsl:attribute>
    </xsl:for-each>
    <w:tr>
      <xsl:apply-template
        select="ns2:Kontaktperson|ns2:Telefon|ns2:Ort|ns2:Telefax|ns2:Position|
              ns2:Firma|ns2:Kunde-Code|ns2:Land|ns2:Region|ns2:Strasse|ns2:PLZ"
      />
    </w:tr>
  </ns2:Kunden>
</xsl:template>
```

**Listing 30.3** Der vom »Inference Tool« erstellte `<xml-template>`-Code für alle Kindelemente des *Kunde*-Elements

```
<xsl:template match="/ns2:dataroot/ns2:Kunde/ns2:Kunden-Code">
  <ns2:Kunden-Code>
    <xsl:for-each select="@ns2:*|@*[namespace-uri()='']">
      <xsl:attribute name="{name()}" namespace="{namespace-uri()}">
        <xsl:value-of select="." />
      </xsl:attribute>
    </xsl:for-each>
```



**Listing 30.3** Der vom »Inference Tool« erstellte `<xml-template>`-Code für alle Kindelemente des *Kunde*-Elements (Fortsetzung)

```

<w:tc>
  <w:tcPr>
    <w:tcW w:w="1247" w:type="dxa" />
  </w:tcPr>
  <w:p>
    <w:pPr>
      <w:rPr>
        <w:sz w:val="20" />
        <w:sz-cs w:val="20" />
      </w:rPr>
    </w:pPr>
    <w:r>
      <w:rPr>
        <w:sz w:val="20" />
        <w:sz-cs w:val="20" />
      </w:rPr>
      <w:t><xsl:value-of select="." /></w:t></w:r>
    </w:p>
  </w:tc>
</ns2:Kunden-Code>
</xsl:template>

```

Der für diese Lösung angepasste Code für das Kunde-Element (ersetzt den in Listing 30.2 und Listing 30.3) befindet sich in Listing 30.4. Im Gegensatz zur vorangehenden Transformation wird die Tabellenzeile zuerst erstellt und danach die Zellen gefüllt, Element für Element (nur den Teil für das Element Kunden-Code wird angezeigt; alle weiteren Elemente werden ähnlich behandelt).

**Listing 30.4** Die angepasste Transformation erzeugt, unabhängig von der Anzahl der Kindelemente eines *Kunde*-Elements, eine gleichmäßige Tabelle.

```

<xsl:template match="/ns2:dataroot/ns2:Kunde">
  <ns2:Kunde>
    <xsl:for-each select="@ns2:*|@*[namespace-uri()='']">
      <xsl:attribute name="{name()}" namespace="{namespace-uri()}">
        <xsl:value-of select="." />
      </xsl:attribute>
    </xsl:for-each>
    <w:tr>
      <ns2:Kunden-Code>
        <w:tc>
          <w:tcPr>
            <w:tcW w:w="1247" w:type="dxa" />
          </w:tcPr>
          <w:p>
            <w:pPr>
              <w:rPr>
                <w:sz w:val="20" />
                <w:sz-cs w:val="20" />
              </w:rPr>
            </w:pPr>
            <w:r>
              <w:rPr>
                <w:sz w:val="20" />
                <w:sz-cs w:val="20" />
              </w:rPr>
              <w:t><xsl:value-of select="." /></w:t>
            </w:r>
          </w:p>
        </w:tc>
      </ns2:Kunden-Code>
    </w:tr>
  </ns2:Kunde>
</xsl:template>

```

**Listing 30.4** Die angepasste Transformation erzeugt, unabhängig von der Anzahl der Kindelemente eines *Kunde*-Elements, eine gleichmäßige Tabelle. (Fortsetzung)

```

        </w:rPr>
        <w:t><xsl:apply-templates select="ns2:Kunden-Code" /></w:t>
    </w:r>
</w:p>
</w:tc>
</ns2:Kunden-Code>

```

*Bsp30\_01S.xml*  
(die MOSTL-Solution)

Der Code dieser Datei befindet sich in Listing 30.5. Eine Smart Document MOSTL-Solution muss das gleiche Schema respektieren wie eine Smarttag-MOSTL (siehe Kapitel 29). Einige Unterschiede gibt es dennoch:

- Die Datei muss ein `solutionID`-Element enthalten, das die Lösung eindeutig identifiziert. Diese Angabe muss mit der Angabe im `solutionID`-Element des Manifests genau übereinstimmen. Allgemein wird hierfür eine GUID (»Globally Unique Identifier«) benutzt (mehr zum Thema GUID und deren Erstellung findet sich im Abschnitt »Globally Unique Identifiers (GUIDs)« in diesem Kapitel).
- Es muss sich in der Datei für jedes Element des Schemas, dem Aktionen zugewiesen werden, je ein `smartDoc`-Element befinden.
- Das Attribut des Smartdoc-Typs besteht aus dem URI des Schemas, gefolgt vom #-Zeichen und dem Elementnamen. Um beispielsweise Aktionen für das Kunde-Element zu definieren:

```
<Kunde type="http://www.KAP30Beispiel.com/BSP01#Kunde">
```

- Diese URI werden als »Smart Document Type Names« oder auch gelegentlich als »namensraum#element Namen« bezeichnet. Damit erkennt der Smart Document-Mechanismus, welche Aktionen welchen Elementen zugewiesen sind. Da XML auf die Groß-/Kleinschreibung achtet, müssen die Bezeichnungen mit den Elementnamen *exakt* übereinstimmen.

---

**HINWEIS** Die »Smart Document Type Names« sind im Grunde genommen die »Recognizers« der Smart Document-Technologie, die Stellen im Dokument mit Aktionen verbinden.

---

Da »Smart Document Type Names« nur auf Namensraum plus Elementnamen basieren, kann der Mechanismus nicht zwischen Elementen aus verschiedenen Hierarchien unterscheiden, die den gleichen Namen haben. Gäbe es beispielsweise zwei Adressenarten, Rechnungs- sowie Versandadresse, und beide hätten ein Kindelement `Ort`, müssten beide `Ort`-Elemente die gleichen Aktionen haben.

---

**WICHTIG** Der Smart Document MOSTL-Mechanismus kann Sonderzeichen wie Umlaute und das scharfe S »ß« in »namensraum#element Namen« nicht korrekt erkennen. Deshalb müssen für dieses Beispiel Feldnamen wie »Straße« in »Strasse« geändert werden. Dieses Problem besteht in Smart Documents, die auf einer COM-DLL basieren, nicht.

---

- Wie bei Smarttags können für jedes `smartDoc`-Element mehrere Aktionen bereitstehen. Die Smart Document-Technologie stellt jedoch mehrere Arten von Aktionen bereit, wie Schaltflächen und Hilfetext.

- Das aktuelle smartDoc-Element wird auch durch eine Beschriftung, gefolgt von den zur Verfügung gestellten Aktionen, angekündigt. Diese befinden sich jedoch im Aufgabenbereich *Dokumentaktionen* statt im Smarttag-Kontextmenü.

Die einer Smart Document MOSTL-Lösung zur Verfügung stehenden Aktionen sind in Tabelle 30.2 aufgelistet.

Tabelle 30.2 Aktionen für Smart Document MOSTL-Lösungen

Aktion	Beschreibung/Bemerkung
<i>Separator</i>	Waagerechte Trennlinie. Wurde in diesem Beispiel nicht eingesetzt, da die Trennlinien zwischen Elementen genügen.
<i>Label</i>	Beschriftung von maximal 256 Zeichen
<i>Help</i>	Hilfetext in HTML-Format; nur einfaches HTML wird unterstützt. Es besteht aus einer Überschrift, gefolgt vom Fließtext, der durch Anklicken eines Symbols unsichtbar gemacht werden kann. Ist einer Beschriftung vorzuziehen, wenn der Text aus mehr als 256 Zeichen besteht oder zwecks <b>Hervorhebung</b> formatiert werden soll. Der HTML-Code kann nicht in einer externen Datei ausgelagert werden.
<i>Link</i> sowie <i>Button</i>	Die zwei Aktionen sind sich ähnlich. Beide zeigen ihre Beschriftung an und werden durch Anklicken ausgeführt. Entweder öffnen sie einen URL in einem neuen Fenster oder sie führen das definierte <code>&lt;elementAction&gt;</code> -Element aus.
<i>elementAction</i>	Obwohl zwei <code>&lt;elementAction&gt;</code> -Arten dokumentiert sind – <code>&lt;insertXML&gt;</code> und <code>&lt;removeXML&gt;</code> –, scheint nur die erstere zu funktionieren. Der Inhalt des <code>&lt;insertXML&gt;</code> -Elements wird an die aktuelle Stelle im Dokument eingefügt. Dieser Inhalt muss ein gültiger WordProcessingML sein und deshalb auch auf den WordProcessingML-Namensraum des <code>smartTagList</code> -Elements verweisen (siehe Listing 30.5).
<i>Image</i>	Eine Grafikdatei wird in das Element geladen. Klickt der Benutzer darauf, wird der zugehörige <code>&lt;url&gt;</code> -Element-URL in einem neuen Browserfenster geöffnet.

Listing 30.5 Inhalt der MOSTL-Solution-Datei *Bsp30\_015.xml*

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<smartTagList
  xmlns="http://schemas.microsoft.com/office/smarttags/2003/mostl"
  xmlns:w="http://schemas.microsoft.com/office/word/2003/wordml">
  <solutionID>{9CB3539A-890D-4C50-A07E-3FF847449D27}</solutionID>
  <name>MOSTL Smart Document - Kunden</name>
  <lcid>1031,0</lcid>
  <description>MOSTL Smart Document Beispiel für den Unterhalt der Nordwind-
    Kundendaten</description>

  <smartDoc type="http://www.KAP30Beispiel.com/BSP01#dataroot">
    <caption>Nordwind Kunden dataroot-Element</caption>
    <actions>
      <action id="dataroot_help">
        <actionType>Help</actionType>
        <caption>Hilfe für das dataroot</caption>
        <help><html><body>
          <p>Es handelt sich hier um ein <i>MOSTL</i> Smart Document.
            Es kann ein oder mehr Kunden-Elemente umfassen.</p>
        </body></html></help>
      </action>
```

**Listing 30.5** Inhalt der MOSTL-Solution-Datei *Bsp30\_01S.xml* (Fortsetzung)

```

    <action id="dataroot_hyperlink">
      <actionType>Link</actionType>
      <caption>Beispiel Link zur Microsoft Deutschland Webseite</caption>
      <url>http://www.microsoft.com/germany</url>
    </action>

    <action id="dataroot_button">
      <actionType>Button</actionType>
      <caption>Nordwind-Datenbank öffnen</caption>
      <url>file:///c:/Beispiele/datenbank/Nordwind.mdb</url>
    </action>
  </actions>
</smartDoc>

<smartDoc type="http://www.KAP30Beispiel.com/BSP01#Kunde">
  <caption>Element Kunde</caption>
  <actions>
    <action id="Kunde_label">
      <actionType>Label</actionType>
      <caption>Für jeden Kunden gibt es ein Kunde-Element. Jedes Kunde-Element enthält
        nur Unterelemente.</caption>
    </action>
  </actions>
</smartDoc>

<smartDoc type="http://www.KAP30Beispiel.com/BSP01#Kunden-Code">
  <caption>Element Kunden-Code</caption>
  <actions>
    <action id="Kunden-Code_help">
      <actionType>Help</actionType>
      <caption>Kunden-Code-Hilfe</caption>
      <help><html><body>
        <p>In der Nordwind-Datenbank stellt der Kunden-Code eine eindeutige Bezeichnung
          des Kunden dar. Eindeutiger Code aus 5 Buchstaben, basierend auf dem
          Kundennamen. <b>Erforderlich.</b></p>
      </body></html></help>
    </action>
  </actions>
</smartDoc>

<smartDoc type="http://www.KAP30Beispiel.com/BSP01#Firma">
  <caption>Element Firma</caption>
  <actions>
    <action id="Firma_help">
      <actionType>Help</actionType>
      <caption>Firma-Hilfe</caption>
      <help><html><body>
        <p>Firmenname des Kunden. Maximal 40 Zeichen. <b>Erforderlich.</b></p>
      </body></html></help>
    </action>
  </actions>
</smartDoc>

<smartDoc type="http://www.KAP30Beispiel.com/BSP01#Kontaktperson">
  <caption> Element Kontaktperson</caption>
  <actions>

```

Listing 30.5 Inhalt der MOSTL-Solution-Datei *Bsp30\_01S.xml* (Fortsetzung)

```

    <action id="Kontaktperson_label">
      <actionType>Label</actionType>
      <caption>Name der Hauptkontaktperson der Firma. Maximal 30 Zeichen.</caption>
    </action>
  </actions>
</smartDoc>

<smartDoc type="http://www.KAP30Beispiel.com/BSP01#Position">
  <caption> Element Position</caption>
  <actions>
    <action id="Position_help">
      <actionType>Help</actionType>
      <caption>Position-Hilfe</caption>
      <help><html><body>
        <p>Position der Kontaktperson in der Firma.
        Maximal 30 Zeichen.</p><p>Haeufig vorkommende Positionen koennen mit Hilfe der
        folgenden Links an der markierten Stelle eingefuegt werden.</p>
      </body></html></help>
    </action>

    <action id="Position_1">
      <actionType>Link</actionType>
      <caption>"Buchhalter" einfügen</caption>
      <elementAction>
        <insertXML><w:p><w:r><w:t>Buchhalter</w:t></w:r></w:p></insertXML>
      </elementAction>
    </action>

    <action id="Position_2">
      <actionType>Link</actionType>
      <caption>"Buchhalterin" einfügen</caption>
      <elementAction>
        <insertXML><w:p><w:r><w:t>Buchhalterin</w:t></w:r></w:p></insertXML>
      </elementAction>
    </action>

    <action id="Position_3">
      <actionType>Link</actionType>
      <caption>"Inhaber" einfügen</caption>
      <elementAction>
        <insertXML><w:p><w:r><w:t>Inhaber</w:t></w:r></w:p></insertXML>
      </elementAction>
    </action>

    <action id="Position_4">
      <actionType>Link</actionType>
      <caption>"Inhaberin" einfügen</caption>
      <elementAction>
        <insertXML><w:p><w:r><w:t>Inhaberin</w:t></w:r></w:p></insertXML>
      </elementAction>
    </action>
  </actions>
</smartDoc>

<smartDoc type="http://www.KAP30Beispiel.com/BSP01#Strasse">
  <caption> Element Strasse</caption>

```

Listing 30.5 Inhalt der MOSTL-Solution-Datei *Bsp30\_01S.xml* (Fortsetzung)

```

    <actions>
      <action id="Strasse_label">
        <actionType>Label</actionType>
        <caption>Strasse oder Postfach der Firma. Maximal 60 Zeichen.</caption>
      </action>
    </actions>
  </smartDoc>

  <smartDoc type="http://www.KAP30Beispiel.com/BSP01#Ort">
    <caption>Element Ort</caption>
    <actions>
      <action id="Ort_label">
        <actionType>Label</actionType>
        <caption>Ortsname der Firma. Maximal 15 Zeichen.</caption>
      </action>
    </actions>
  </smartDoc>

  <smartDoc type="http://www.KAP30Beispiel.com/BSP01#PLZ">
    <caption>Element PLZ</caption>
    <actions>
      <action id="PLZ_label">
        <actionType>Label</actionType>
        <caption>Postleitzahl der Firma. Maximal 10 Zeichen.</caption>
      </action>
    </actions>
  </smartDoc>

  <smartDoc type="http://www.KAP30Beispiel.com/BSP01#Region">
    <caption>Element Region</caption>
    <actions>
      <action id="Region_label">
        <actionType>Label</actionType>
        <caption>Bundesland oder Provinz der Firma. Maximal 15 Zeichen.</caption>
      </action>
    </actions>
  </smartDoc>

  <smartDoc type="http://www.KAP30Beispiel.com/BSP01#Land">
    <caption>Element Land</caption>
    <actions>
      <action id="Land_label">
        <actionType>Label</actionType>
        <caption>Land, wo der Firmensitz sich befindet. Maximal 15 Zeichen.</caption>
      </action>
    </actions>
  </smartDoc>

  <smartDoc type="http://www.KAP30Beispiel.com/BSP01#Telefon">
    <caption>Element Telefon</caption>
    <actions>
      <action id="Telefon_label">
        <actionType>Label</actionType>
        <caption>Telefonnummer mit (internationaler) Vorwahl.
          Maximal 24 Zeichen.</caption>
      </action>
    </actions>
  </smartDoc>

```

Listing 30.5 Inhalt der MOSTL-Solution-Datei *Bsp30\_01S.xml* (Fortsetzung)

```

    </action>
  </actions>
</smartDoc>

<smartDoc type="http://www.KAP30Beispiel.com/BSP01#Telefax">
  <caption> Element Telefax</caption>
  <actions>
    <action id="Telefax_label">
      <actionType>Label</actionType>
      <caption>Telefaxnummer mit (internationaler) Vorwahl.
        Maximal 24 Zeichen.</caption>
    </action>
  </actions>
</smartDoc>
</smartTagList>

```

*Bsp30\_01M.xml*  
sowie  
*Bsp30\_01N.xml*  
(Manifest-Dateien)

Eine Manifest-Datei enthält Informationen über die Lösung als Einheit sowie jede Datei, die einen Teil davon bildet. Ihr Aufbau muss dem »XML Expansion Pack Schema« (im SDK beschrieben) entsprechen.

Wie im Abschnitt »Das MOSTL-Beispiel installieren« weiter vorne in diesem Kapitel angedeutet, muss ein Manifest normalerweise digital signiert werden. Um das Entwickeln zu vereinfachen und zu beschleunigen, ist es jedoch möglich, die Sicherheitsmaßnahme durch Festlegen eines Registry-Schlüssels teilweise auszuschalten. Mehr Informationen über das Signieren eines Manifests enthält der Abschnitt »Das Erweiterungspaket-Manifest digital signieren«.

Wir zeigen hier in Listing 30.6 nur den Inhalt des Manifests *Bsp30\_01N.xml* an, da dieses auf dem Grundgerüst von *Bsp30\_01M.xml* aufbaut. Weitere Anmerkungen zu diesem Manifest:

- Das Element `<uri>` bezieht sich auf den URI des Schemas (*Bsp30\_01.xsd*).
- Es beinhaltet drei `<solution>`-Elemente, die alle das gleiche `<solutionID>`-Element haben. Einer der Solution-Abschnitte weist auf das Schema, einer auf die Transformation und einer auf die Solution-Datei. Obwohl die Dokumentation andeutet, dass alle Dateien sich in einem einzigen `<solution>`-Element befinden können, darf ein Schema nur in einem `<solution>`-Element des Typs Schema festgelegt werden; und das `<solution>`-Element einer Transformation muss ein `<context>`-Element enthalten (was in einem `<solution>`-Element für andere Dateiartern nicht erlaubt ist).
- Der Inhalt des `<solutionID>`-Elements muss eindeutig sein und dem gleichen Element in der Smart Document MOSTL-Solution entsprechen. Die Dokumentation empfiehlt, eine GUID zu benutzen (siehe den Abschnitt den »Globally Unique Identifiers (GUIDs)« in diesem Kapitel).
- Die `<alias>`-Elemente legen die Beschriftungen fest, die in mehreren Dialogfeldern unter *Extras/Vorlagen und Add-Ins* zu finden sind. Es ist möglich, für jede Sprache, die die Lösung unterstützen soll, `<alias>`-Elemente mit entsprechender LCID zu definieren.
- Steht keine Pfadangabe in einem `<filePath>`-Element, muss sich die Solution-Datei im gleichen Ordner mit dem Manifest befinden. Bitte beachten Sie jedoch, dass ein Smart Document MOSTL-Solution (*Bsp30\_01S.xml*) standardmäßig in den folgenden Ordner kopiert wird:

*C:\Dokumente und Einstellungen\<Benutzername>\Lokale Einstellungen\Anwendungsdaten\Microsoft\Smart Tag Lists*

Die weiteren Dateien werden in einen Ordner kopiert, den vom Installer erstellt wird, meistens handelt es sich um

*C:\Dokumente und Einstellungen\<Benutzername>\Lokale Einstellungen\Anwendungsdaten\Microsoft\Schemas\<Solution URI>\<Solution ID>*

Falls <Solution URI> oder <Solution ID> für einen Ordernamen ungültigen Zeichen enthalten werden diese automatisch ersetzt. Die Dateien für diese Solution werden (mit Ausnahme der MOSTL-Datei) beispielsweise in den folgenden Ordner kopiert:

*C:\Dokumente und Einstellungen\<Benutzername>\Lokale Einstellungen\ Anwendungsdaten\Microsoft\Schemas\http\_\_\_www\_KAP30Beispiel\_com\_BSP01\{9CB3539A-890D-4C50-A07E-3FF847449D27}*

(Nach *http* folgen drei Unterstriche.)

Sie können mit Hilfe des Elements <installPath> Dateien in Unterordner kopieren lassen (mehr Informationen finden Sie im SDK).

- Das Element <targetApplication> legt fest, welche Anwendungen und Anwendungsversionen eine Solution anbieten. Dieses Beispiel soll nur in Microsoft Word 2003 (Version 11) verfügbar sein. Soll eine Lösung in allen Word-Versionen sowie Excel 2003 angeboten werden, braucht es zwei dieser Elemente:

```
<SD:targetApplication>Word.Application</SD:targetApplication>
<SD:targetApplication>Excel.Application.11</SD:targetApplication>
```

**Listing 30.6** Die Manifest-Datei *Bsp30\_01N.xlm* des MOSTL-Smart Document-Beispiels

```
<SD:manifest xmlns:SD="http://schemas.microsoft.com/office/xmlexansionpacks/2003">
  <SD:version>1.0</SD:version>
  <SD:updateFrequency>20160</SD:updateFrequency>
  <SD:uri>http://www.KAP30Beispiel.com/BSP01</SD:uri>
  <SD:solution>
    <SD:solutionID>{9CB3539A-890D-4C50-A07E-3FF847449D27}</SD:solutionID>
    <SD:type>schema</SD:type>
    <SD:alias lcid="0">BSP01</SD:alias>
    <SD:file>
      <SD:type>schema</SD:type>
      <SD:version>1.0</SD:version>
      <SD:filePath>BSP30_01.xsd</SD:filePath>
    </SD:file>
  </SD:solution>
  <SD:solution>
    <SD:solutionID>{9CB3539A-890D-4C50-A07E-3FF847449D27}</SD:solutionID>
    <SD:type>transform</SD:type>
    <SD:alias lcid="0">BSP01T</SD:alias>
    <SD:context>http://schemas.microsoft.com/office/word/2003/wordml</SD:context>
    <SD:file>
      <SD:type>primaryTransform</SD:type>
      <SD:version>1.0</SD:version>
      <SD:filePath>BSP30_01.xsl</SD:filePath>
    </SD:file>
  </SD:solution>
  <SD:solution>
    <SD:solutionID>{9CB3539A-890D-4C50-A07E-3FF847449D27}</SD:solutionID>
    <SD:type>smartDocument</SD:type>
    <SD:alias lcid="0">MOSTL Smartdocument: Nordwind Kunden 2</SD:alias>
    <SD:targetApplication>Word.Application.11</SD:targetApplication>
    <SD:file>
```



Listing 30.6 Die Manifest-Datei *Bsp30\_01N.xml* des MOSTL-Smart Document-Beispiels (Fortsetzung)

```

<SD:type>solutionList</SD:type>
<SD:version>1.0</SD:version>
<SD:filePath>BSP30_01S.xml</SD:filePath>
</SD:file>
</SD:solution>
</SD:manifest>

```

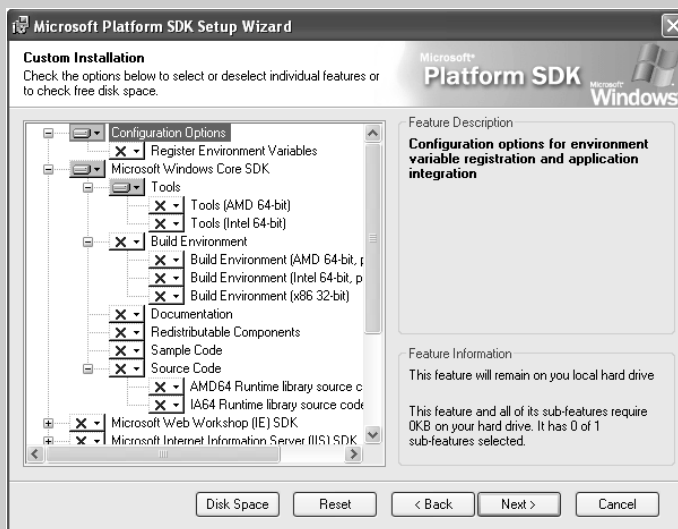
## Globally Unique Identifiers (GUIDs)

Wie im letzten Abschnitt erwähnt, müssen Smart Document Solutions eine eindeutige SolutionID haben. Meistens wird eine GUID (»Globally Unique Identifier«) verwendet.

Microsoft stellt ein Werkzeug – *GUIDGEN.EXE* – zur Verfügung, das eine GUID erstellen kann. Dieses befindet sich im Lieferumfang der .NET-Versionen von Visual Studio und ist auch Bestandteil des »Windows® Server 2003 SP1 Platform SDK«. Letzteres kann kostenlos von der MSDN Webseite heruntergeladen werden bei <http://www.microsoft.com/downloads/details.aspx?FamilyID=A55B6B43-E24F-4EA3-A93E-40C0EC4F68E5&displaylang=en>.

Da dieses SDK recht groß ist, empfiehlt es sich, nur den Teil mit diesem Werkzeug herunterzuladen und installieren. Gehen Sie wie folgt vor:

1. Navigieren Sie zur Webseite und suchen Sie den Abschnitt »Files in this download«. Klicken Sie auf den Link *PSDK-x86.exe* und öffnen oder speichern Sie diesen (es handelt sich um eine Setup-Datei).
2. Führen Sie die Datei aus. Nach den üblichen Setup-Bildschirmen, wie Einführungstext und Lizenzvertrag, wird nach der Installationsart gefragt; wählen Sie »Custom«.
3. Es wird eine Liste der Komponenten entsprechend der Abbildung 30.14 eingeblendet. Deaktivieren Sie alle Optionen außer den Haupteinträgen »Configuration Tools« und »Microsoft Windows Core SDK/Tools«.

Abbildg. 30.14 Installation des Teils des Platform SDK mit dem *GUIDGEN*-Werkzeug

4. Führen Sie die Installation durch. Bitte beachten Sie, dass die Komponenten über das Internet heruntergeladen werden; die Installation kann also dauern.

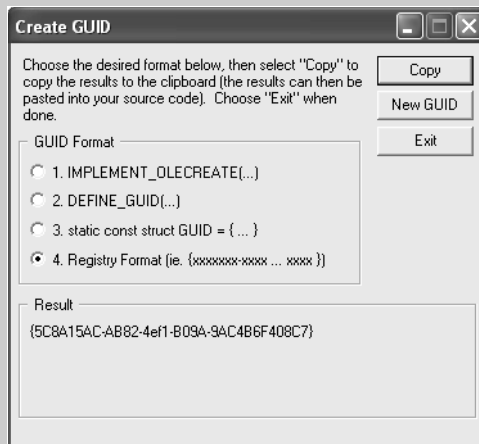
Suchen Sie nun die Datei *GUIDGEN.EXE* auf Ihrem Rechner. Sie müsste sich im festgelegten Speicherort, im Unterordner *Bin* befinden. Führen Sie sie aus; das Dialogfeld aus Abbildung 30.15 sollte erscheinen.

Um eine GUID für eine Smart Document MOSTL-Solution zu generieren, aktivieren Sie die Option *Registry Format*. Aktivieren Sie zunächst *New GUID*, um es zu generieren, und dann *Copy*, um das Ergebnis in die Zwischenablage zu übernehmen.

#### HINWEIS

Falls Sie *GUIDGEN.EXE* nicht installieren möchten, finden Sie im Knowledge Base-Artikel »How To Use CoCreateGUID API to Generate a GUID with VB« unter der Webadresse <http://support.microsoft.com/kb/176790/en-us> Visual Basic-Code (der auch Office-VBA-tauglich ist) und eine Anleitung, um eine GUID zu erstellen.

Abbildg. 30.15 Das SDK-Werkzeug *GUIDGEN.EXE*



## Eine Smart Document-DLL

MOSTL Smart Document-Lösungen sind, was den unterstützten Zeichensatz und die verfügbaren Aktionen angeht, begrenzt. Wird die Lösung in einer COM-DLL geschrieben, erhalten Sie eine erweiterte Funktionalität:

- Es stehen mehr Steuerelemente zur Verfügung, beispielsweise Optionsschaltflächen, Kontrollkästchen, List- sowie Comboboxen und Dokumentfragmente. Sogar selbst definierte ActiveX-Steuerelemente lassen sich in den Aufgabenbereich einbauen.
- HTML-Texte für Hilfe- und Dokumentfragment-Steuerelemente können in externen Dateien ausgelagert werden, was die Flexibilität erhöht und die Pflege erleichtert.
- Die Steuerelemente haben, ähnlich wie die Steuerelemente eines Formulars, Ereignisse. Dieser Code kann jegliche Art von Handlungen ausführen und sogar die Anwendung (Word) automatisieren.

Für den Anwender ist kein Unterschied zu einem MOSTL-Smart Document erkennbar. Er sieht den gleichen Aufgabenbereich mit kontextbezogenem Inhalt.

## Mit VB6 eine Smart Document-DLL erstellen

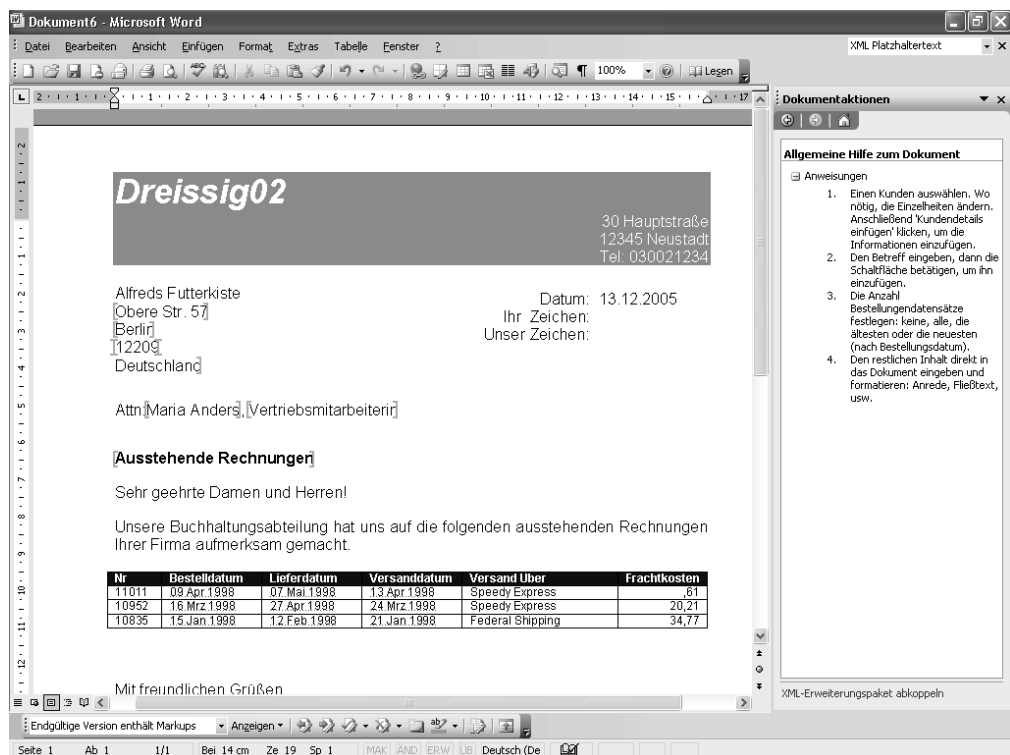
Das Beispiel dieses Abschnitts stellt einige, aber längst nicht alle Möglichkeiten einer DLL-Lösung vor. Es soll vor allem den Aufbau einer Smart Document-DLL verdeutlichen. Die Code-Beispiele sind in der klassischen Visual Basic 6-Sprache gehalten.

Angenommen, die Abteilung »Dreissig02« der Firma Nordwind muss häufig Zahlungserinnerungen an Kunden versenden. Als Vorlage für diese Schreiben dient ein Smart Document, das dem Anwender folgende Arbeitserleichterungen bietet:

- Ein Einführungstext beschreibt, wie die Lösung zu verwenden ist.
- Der anzuschreibende Kunde wird aus einer Dropdownliste gewählt.
- Textfelder dienen der Kontrolle und Eingabe von Angaben, wie die Betreffzeile.
- Über Schaltflächen werden Informationen in die (unsichtbaren) XML-Elemente eingefügt.
- Optionsschaltflächen helfen, die Auswahl der Kunden-Bestellungen festzulegen.

Das fertige Schreiben ist in Abbildung 30.16 dargestellt.

Abbildg. 30.16 Ergebnis der Smart Document-Lösung



## Das DLL-Beispiel installieren

Um die Diskussion mitzuverfolgen, installieren Sie am besten die Dateien von der CD-ROM.



Alle in Tabelle 30.3 benötigten Beispieldateien befinden sich auf der CD-ROM im Ordner `\Beispiele\Kap30`. Die Beispiel-Datenbank *Nordwind.mdb* finden Sie im Ordner `\Beispiele\Datenbank`.

**Tabelle 30.3** Beispieldateien für den Abschnitt über eine Smart Document-DLL

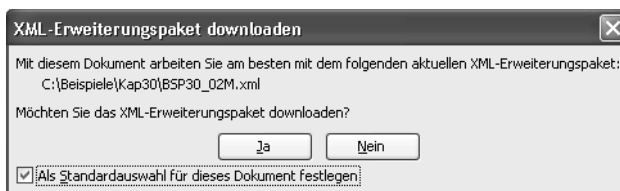
Beispieldatei	Beschreibung
<i>Bsp30_02.dot</i>	Eine Word-Vorlage mit Briefkopf und Standardtext sowie Elemente eines benutzerdefinierten XML-Vokabulars
<i>Bsp30_02.xsd</i>	Das mit der Vorlage <i>Bsp30_02.dot</i> verbundene Schema
<i>Bsp30_02.dll</i>	Eine in VB6 erstellte COM-DLL (»action handler«), die das Smart Document-Interface implementiert. Das Projekt enthält ein einziges Klassenmodul: <i>smartdoc.cls</i> .
<i>Bsp30_02.udl</i>	Eine Data-Link-Datei, die der DLL die Verbindungsangaben für die Datenbank bereitstellt.
<i>Bsp30_02.xsl</i>	Wird vom »action handler« zur Transformation der Bestelldaten bei deren Einfügung benutzt.
<i>Bsp30_02.xml</i>	Das XML-Erweiterungspaket-Manifest. Installiert die Komponenten der Lösung: DLL, Schema sowie Transformation.

Es befinden sich im Ordner zudem in der ZIP-Datei *Bsp30\_02.zip* alle Quelldateien, um das Projekt in Visual Basic 6 zu öffnen und bearbeiten.

1. Kopieren Sie alle Dateien im Ordner `\Beispiele\Kap30` auf der Buch-CD, deren Namen mit den Zeichen »Bsp30\_02« beginnen, in einen lokalen Ordner mit der Pfadangabe `C:\Beispiele\Kap30`.
2. Stellen Sie sicher, dass sich die Beispieldatenbank *Nordwind.mdb* im Ordner `C:\Beispiele\Datenbank` befindet. (Falls Sie Nordwind in einem anderen Ordner gespeichert haben, müssen Sie die Pfadangabe in *Bsp30\_02.udl* mit dem Windows-Editor entsprechend anpassen.)
3. Navigieren Sie im Windows-Explorer zum Ordner mit den Beispieldateien und klicken Sie doppelt auf *Bsp30\_02.dot*, um ein neues Dokument zu erzeugen.

Falls das damit verbundene Erweiterungspaket noch nicht installiert ist, erscheint die Meldung in Abbildung 30.17. Aktivieren Sie, wie beim MOSTL-Beispiel, das Kontrollkästchen und bestätigen Sie die Installation mit *Ja*.

**Abbildg. 30.17** Das Erweiterungspaket installieren



Falls die Sicherheit für Erweiterungspakete ausgeschaltet ist, wird auch diese Meldung (Abbildung 30.4) eingeblendet.

Im Gegensatz zu einer MOSTL-Lösung ist für die Installation des DLL-Erweiterungspakets die vollständige Pfadangabe vorgeschrieben. Diese wird in der benutzerdefinierten Dokumenteigenschaft *Solution URL* festgehalten. Beim Öffnen eines Dokuments kontrolliert Word, ob diese Eigenschaft vorhanden ist. Wenn ja, wird das Erweiterungspaket im angegebenen Ordner gesucht. Ist das Erweiterungspaket vorhanden, aber noch nicht lokal installiert, erfolgt die Aufforderung in Abbildung 30.17. Ansonsten wird das Dokument wie ein normales Dokument, ohne den Aufgabenbereich *Dokumentaktionen*, geöffnet. (Mehr darüber können Sie im SDK lesen.)

Um eine Lösung aus einem anderen Pfad installieren zu lassen, ändern Sie einfach in der Vorlage (*Bsp30\_02.dot*) die Pfadangabe in *Datei/Eigenschaften/Anpassen* für *Solution URL*.

Bitte beachten Sie jedoch, dass Word die Lösung erst wieder neu installiert, wenn sie nicht schon auf dem Rechner registriert ist. Sind Sie den Anweisungen bis hierher gefolgt, müssen Sie:

1. alle mit dem Schema verbundenen Dateien in Word schließen,
2. in *Extras/Vorlagen und Add-Ins/XML-Schema* die Schaltfläche *Schemabibliothek* anklicken,
3. den Schema-Eintrag in der Liste markieren und *Schema löschen* betätigen.
4. die darauf folgende Frage, ob Sie das Schema samt alle verknüpften Dateien löschen möchten, mit *Ja* beantworten.

Nun wird beim Erstellen eines neuen Dokuments die Meldung in Abbildung 30.17 eingeblendet und die Lösung installiert.

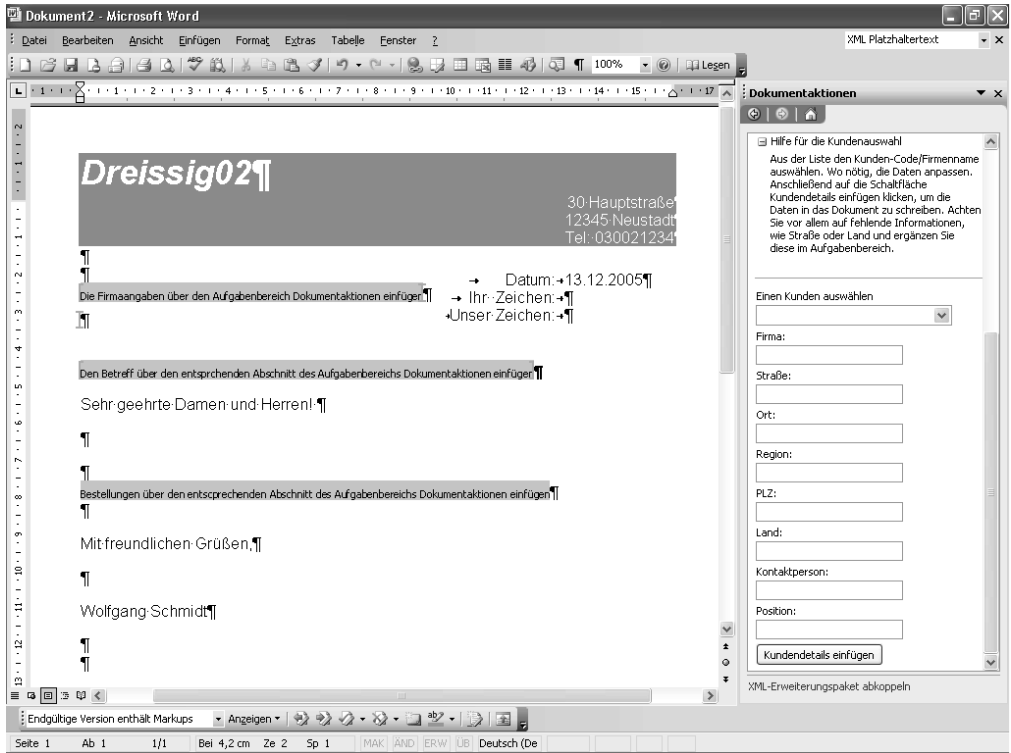
#### HINWEIS

Sie können das Erweiterungspaket auch direkt laden. Stellen Sie sicher, dass das Schema für die Lösung in der Registerkarte *XML-Schema* aufgelistet, markiert und aktiviert ist. Wechseln Sie zur Registerkarte *XML-Erweiterungspakete*, um das Erweiterungspaket hinzuzufügen. Wenn Sie dies in der Vorlage tun, wird der Speicherort des Manifests in der Dokumenteigenschaft festgehalten.

Auf dem Bildschirm erscheint das Gerüst des Briefes mit Platzhaltertexten an drei Stellen (Abbildung 30.18). Die Eingaben hierfür werden mit den Werkzeugen des Aufgabenbereichs *Dokumentaktionen* ausgewählt bzw. eingegeben und eingefügt. Dank dem Code, der beim Laden des Erweiterungspakets ausgeführt wurde (siehe den Abschnitt »Die Interaktion mit dem *SmartDocument-Interface*« in diesem Kapitel), befindet sich die Einfügemarke an der ersten Stelle, dem Adressbereich. Im Aufgabenbereich erscheinen die zugehörigen Eingabefelder für die Kundendaten.

Das Smart Document stellt eine Verbindung zur Datenbank her, um die Dropdownliste in Abbildung 30.19 zu füllen. Nach Auswahl eines Eintrags erscheinen die Informationen in den darunter liegenden Textfeldern, wo sie nach Bedarf bearbeitet werden können. Durch Betätigung der Schaltfläche *Kundendetails einfügen* werden die Daten in die (unsichtbaren) XML-Elemente des Dokuments übertragen. Die Erläuterung dazu finden Sie im Abschnitt »Steuerelementereignisse« in diesem Kapitel.

Abbildg. 30.18 Das Gerüst für einen Kundenbrief



Enthält das Element *Betreff* noch keinen Text, wird anschließend die Einfügemarke automatisch in den Bereich für den *Betreff* gesetzt und der Aufgabenbereich passt sich entsprechend an, wie in Abbildung 30.19 ersichtlich. Beachten Sie den Hilfetext im oberen Teil, der für das ganze Dokument gilt. Analog zu MOSTL-Lösungen wird der Inhalt des Aufgabenbereichs hierarchisch nach Verschachtelung der Elemente aufgebaut. Dieser Vorgang wird im Abschnitt »Die Interaktion mit dem *SmartDocument*-Interface« beschrieben.

**Abbildg. 30.19** Der Kunden-spezifische Teil des Aufgabenbereichs stellt aktuelle Daten aus der Datenbank zur Verfügung.

**Dokumentaktionen**

**Hilfe für das Kunde-Element**

☐ Hilfe für die Kundenauswahl

Aus der Liste den Kunden-Code/Firmenname auswählen. Wo nötig, die Daten anpassen. Anschließend auf die Schaltfläche Kundendetails einfügen klicken, um die Daten in das Dokument zu schreiben. Achten Sie vor allem auf fehlende Informationen, wie Straße oder Land und ergänzen Sie diese im Aufgabenbereich.

Einen Kunden auswählen

ALFKI: Alfreds Futterkiste

ALFKI: Alfreds Futterkiste

ANATR: Ana Trujillo Emparedados y h

ANTON: Antonio Moreno Taqueria

AROUT: Around the Horn

BERGS: Berglunds snabbköp

BLAUS: Blauer See Delikatessen

BLONP: Blondel père et fils

BOLID: Bólido Comidas preparadas

BONAP: Bon app'

BOTTM: Bottom-Dollar Markets

BSBEV: B's Beverages

CACTU: Cactus Comidas para llevar

12209

Land:

Deutschland

Kontaktperson:

Maria Anders

Position:

Vertriebsmitarbeiterin

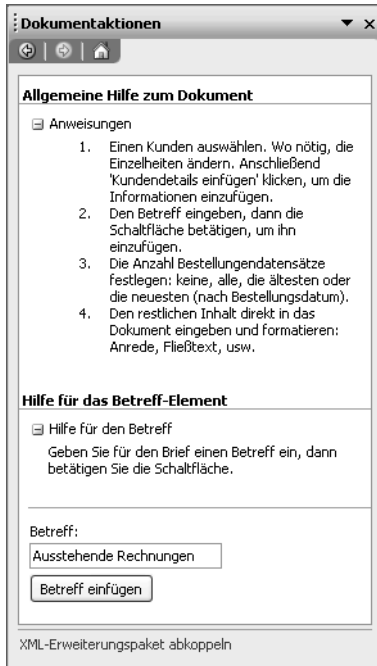
Kundendetails einfügen

XML-Erweiterungspaket abkoppeln

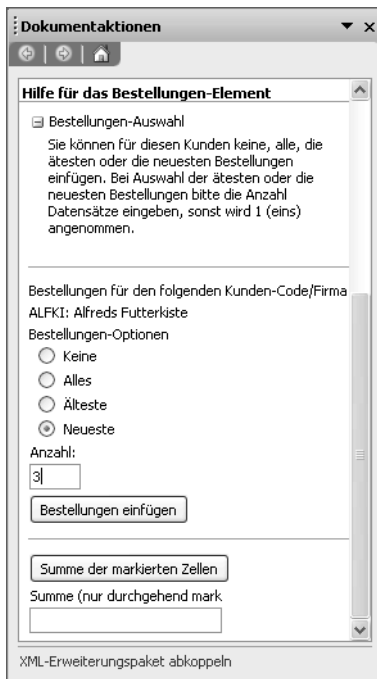
Nach dem Einfügen des Betreffs bietet die Lösung die Möglichkeit, Bestellinformationen für den Kunden auszuwählen und einzufügen. Der Anwender kann entweder keine, alle oder eine Auswahl der jüngsten bzw. ältesten Bestellungen einfügen, wie in Abbildung 30.21 dargestellt. Diese werden in Form einer Tabelle in das Element *Bestellungen* eingefügt (bzw. ein Leerzeichen, falls der Anwender *Keine* angibt). Mehr dazu lesen Sie im Abschnitt »Steuerelementereignisse« weiter hinten in diesem Kapitel.

Zudem bietet dieser Teil die Gelegenheit, den numerischen Inhalt der in der Tabelle markierten Zellen zu addieren. Nach einem Klick auf die Schaltfläche *Summe der markierten Zellen* erscheint das Ergebnis im darunter liegenden Textfeld.

Abbildg. 30.20 Der Aufgabenbereich mit der Hilfe zum ganzen Dokument sowie für das Element *Betreff*



Abbildg. 30.21 Über die Schaltfläche *Summe der markierten Zellen* können Werte addiert werden





Der Brief muss nicht in der beschriebenen Reihenfolge abgearbeitet werden. Zudem können durch Klicken in einen Elementbereich jederzeit Änderungen vorgenommen werden. Klickt der Anwender beispielsweise in einen Teil der Kundenadresse, blendet der Aufgabenbereich die entsprechenden Steuerelemente in Abbildung 30.19 wieder ein, und es kann ein anderer Kunde ausgewählt werden.

## Wie es funktioniert: das *SmartDocument*-Interface

Im Allgemeinen funktioniert eine auf einer DLL basierende Smart Document-Lösung ähnlich wie eine MOSTL-Lösung. Beim Öffnen des Dokuments wird das Erweiterungspaket aufgerufen, das nötigenfalls die Lösungskomponenten herunterlädt und installiert. Dann erstellt es eine Verbindung zur zuständigen DLL (im Fall von einer MOSTL *MOFL.dll*). Während bei einer MOSTL die Angaben für die im Aufgabenbereich definierten Aktionen durch eine XML-Datei (die Solution-Datei) bereitgestellt werden, übernimmt diese Aufgabe in einer DLL das Klassenmodul, worin das *SmartDocument*-Interface implementiert wird.



In der folgenden Diskussion werden nicht alle Codezeilen des Beispiel-Projekts wiedergegeben. Diese stehen in voller Länge in der reinen Textdatei *smartdoc.cls* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap30* zur Verfügung.

### Die Steuerelemente des Aufgabenbereichs

Word braucht einen Mechanismus, um die sich im Dokument befindlichen Elemente mit den Aktionen im Aufgabenbereich zu verbinden. In einer DLL werden diese Zusammenhänge in der Implementierung des *SmartDocument*-Interface definiert und über die folgenden Eigenschaften bereitgestellt.

SmartDoc  
XMLType  
Count

Die *SmartDocXMLTypeCount*-Eigenschaft teilt Word mit, für wie viele Elemente des Schemas im Aufgabenbereich *Dokumentaktionen* Steuerelemente definiert sind. Im Beispiel sind es deren vier: ein besonderes Pseudo-Element namens *actionPertainsToEntireSchema* (Aktionen, die sichtbar sind, egal wo die Einfügemarke im Dokument steht, wie allgemeiner Hilfetext) sowie die drei Elemente *Kunde*, *Betreff* und *Bestellungen*.

```
Private Property Get ISmartDocument_SmartDocXmlTypeCount() As Long
    ISmartDocument_SmartDocXmlTypeCount = NUM_TYPES
End Property
```

Für jedes Element werden weitere Eigenschaften festgelegt:

XMLType  
ID

Intern weist Word jedem Element einen Ganzzahl-ID-Wert – die *XMLTypeID* – zu, angefangen mit 1 (eins). Im Beispiel betragen die Werte also 1, 2, 3 und 4.

Um den Umgang damit einfacher zu gestalten, werden am Anfang des Klassenmoduls Konstanten mit diesen Werten deklariert:

```
Private Const SCHEMA_ID As Integer = 1
Private Const KUNDE_ID As Integer = 2
Private Const BETREFF_ID As Integer = 3
Private Const BESTELLUNGEN_ID As Integer = 4
```

SmartDoc Um die Steuerelemente eindeutig zu identifizieren, verwendet das *SmartDocument*-Interface wie die  
 XmlType MOSTL den SmartDocXMLTypeName – eine Kombination des Namensraums plus Elementname: *name-  
 Name space#element-name*. Diese wird der XMLTypeID zugewiesen. Damit weiß Word, welche Elemente im  
 Dokument mit welchem Satz Aktionen verbunden sind. Die Namen werden am Modulanfang als  
 Konstantwerte deklariert ...

```
Private Const SCHEMA As String = NAMESPACE & "#actionPertainsToEntireSchema"
Private Const KUNDE As String = NAMESPACE & "#Kunde"
Private Const BETREFF As String = NAMESPACE & "#Betreff"
Private Const BESTELLUNGEN As String = NAMESPACE & "#Bestellungen"
```

... und dann dem SmartDocXMLTypeName zugewiesen:

```
Private Property Get ISmartDocument_SmartDocXMLTypeName( _
  ByVal XMLTypeID As Long) As String
  Select Case XMLTypeID
    Case SCHEMA_ID
      ISmartDocument_SmartDocXMLTypeName = SCHEMA
    Case KUNDE_ID
      ISmartDocument_SmartDocXMLTypeName = KUNDE
    Case BETREFF_ID
      ISmartDocument_SmartDocXMLTypeName = BETREFF
    Case BESTELLUNGEN_ID
      ISmartDocument_SmartDocXMLTypeName = BESTELLUNGEN
    Case Else
  End Select
End Property
```

SmartDoc Die Abschnittsüberschrift im Aufgabenbereich für ein bestimmtes Element (»Hilfe für das Kunde-  
 Xml Element« beispielsweise, siehe Abbildung 30.19) bestimmt die Eigenschaft SmartDocXMLCaption.  
 Caption Diese kann sprachspezifisch nach LCID sein, um mehrsprachige Lösungen zu ermöglichen. Das fol-  
 gende Codefragment veranschaulicht nur die deutschen Beschriftungen:

```
Private Property Get ISmartDocument_SmartDocXMLTypeCaption( _
  ByVal XMLTypeID As Long, ByVal LocaleID As Long) As String
  Select Case LocaleID
    Case 1031 ' Deutsch
      Select Case XMLTypeID
        Case SCHEMA_ID
          ISmartDocument_SmartDocXMLTypeCaption = "Allgemeine Hilfe zum Dokument"
        Case KUNDE_ID
          ISmartDocument_SmartDocXMLTypeCaption = "Hilfe für das Kunde-Element"
        Case BETREFF_ID
          ISmartDocument_SmartDocXMLTypeCaption = "Hilfe für das Betreff-Element "
        Case BESTELLUNGEN_ID
          ISmartDocument_SmartDocXMLTypeCaption = "Hilfe für das Bestellungen-Element"
        Case Else
      End Select
    Case Else 'Für nicht angegebenen LCID
  End Select
End Property
```

Control  
Count

Word benötigt auch die Angabe, wie viele Aktionen für jedes Element im Aufgabenbereich angezeigt werden. Dies wird in der Eigenschaft `ControlCount` festgehalten. Im Beispiel sind es für das Element Kunde beispielsweise zwölf Steuerelemente:

```
Private Property Get ISmartDocument_ControlCount( _
    ByVal XMLTypeName As String) As Long
    Select Case XMLTypeName
        Case SCHEMA
            ISmartDocument_ControlCount = 1
        Case KUNDE
            ISmartDocument_ControlCount = 12
        Case BETREFF
            ISmartDocument_ControlCount = 4
        Case BESTELLUNGEN
            ISmartDocument_ControlCount = 10
        Case Else
            End Select
    End Property
```

Control  
Index  
sowie  
Control  
ID

Intern weist das Interface jedem Steuerelement eines XML-Elements (`XMLTypeName`) einen Indexwert zu, beginnend mit 1 (eins). Die ersten Steuerelemente des Elements Kunde sowie des Elements Betreff haben also den gleichen `ControlIndex`-Wert: 1.

Um dem Entwickler den Umgang damit etwas zu erleichtern, wird jeder Kombination von `XMLTypeName` plus `ControlIndex` eine `ControlID` zugewiesen. Diese *muss* eine Ganzzahl und eindeutig sein. Deshalb wird meistens jedem XML-Element ein gewisser Zahlenbereich, wie im folgenden Codefragment, zugewiesen und zu dem `ControlIndex`-Wert addiert.

Die Trennlinie im Kunde-Abschnitt der Abbildung 30.19 beispielsweise besteht aus 200 (Zahlenbereich für das Element Kunde) + 2 (`ControlIndex` des Steuerelements).

```
Private Property Get ISmartDocument_ControlID(
    ByVal XMLTypeName As String, ByVal ControlIndex As Long) As Long
    Select Case XMLTypeName
        Case SCHEMA
            ISmartDocument_ControlID = ControlIndex + 100
        Case KUNDE
            ISmartDocument_ControlID = ControlIndex + 200
        Case BETREFF
            ISmartDocument_ControlID = ControlIndex + 300
        Case BESTELLUNGEN
            ISmartDocument_ControlID = ControlIndex + 400
        Case Else
            End Select
    End Property
```

Da diese Zuteilung bekannt ist, kann der Umgang noch entwicklerfreundlicher gestaltet werden, indem die Werte Konstanten zugewiesen werden. In allen Prozeduren, die mit der `ControlID` arbeiten, wird dann der Konstantenwert anstelle der Ganzzahl verwendet. Hier befinden sich einige der Konstanten-Deklarationen am Modulanfang:

```
Private Const SCHEMA_HELP As Integer = 101
Private Const KUNDE_HELP As Integer = 201
Private Const KUNDE_SEPARATOR1 As Integer = 202
Private Const KUNDE_COMBO As Integer = 203
```

Control  
Name  
FromID

Um gewisse Aufgaben erledigen zu können, sollen die Steuerelemente im Aufgabenbereich durch Automatisierungscode (Word-VBA beispielsweise) angesprochen werden. Die `ControlNameFromID`-Eigenschaft definiert den dafür notwendigen, eindeutigen Indexwert. In diesem Beispiel wird einfach der `ControlID`-Wert in eine Zeichenkette umwandelt. Ein Beispiel für dessen Einsatz finden Sie in Listing 30.12.

```
Private Property Get ISmartDocument_ControlNameFromID(ByVal ControlID As Long) As String
    ISmartDocument_ControlNameFromID = CStr(ControlID)
End Property
```

**TIPP**

Noch eindeutiger wäre eine Kombination des `SmartDocXmlTypeName` plus `ControlID` (ergäbe beispielsweise »<http://www.KAP30beispiel.com/BSP30#Kunde202>«).

Control  
Caption  
FromID

Jedem Steuerelement soll eine Beschriftung zugewiesen werden, was über die Eigenschaft `ControlCaptionFromID` erfolgt. Beschriftungen dürfen sprachspezifisch zugewiesen werden, wie der folgende Codeschnipsel veranschaulicht. Beachten Sie, wie die Konstantwerte der `ControlID` hier und in der nächsten Eigenschaft eingesetzt werden:

```
Private Property Get ISmartDocument_ControlCaptionFromID( ByVal ControlID As Long, _
    ByVal ApplicationName As String, ByVal LocaleID As Long, ByVal Text As String, _
    ByVal Xml As String, ByVal Target As Object) As String
    Dim rngTarget As Word.Range
    Select Case LocaleID
        Case 1031 ' Deutsch
            Select Case ControlID
                Case SCHEMA_HELP
                    ISmartDocument_ControlCaptionFromID = "Anweisungen"
                Case KUNDE_HELP
                    ISmartDocument_ControlCaptionFromID = "Hilfe für die Kundenauswahl"
                Case KUNDE_SEPARATOR1
                    ISmartDocument_ControlCaptionFromID = "Trennlinie"
                Case KUNDE_COMBO
                    ISmartDocument_ControlCaptionFromID = "Einen Kunden auswählen"
                Case KUNDE_FIRMA_TEXTBOX
                    ISmartDocument_ControlCaptionFromID = "Firma:"
            ' Weitere Zuweisungen ...
        Case Else
            End Select
        Case 1033 'UK Englisch
        Case Else ' Standardmäßige Sprache
    End Select
End Property
```

**WICHTIG**

Ist die Beschriftung eines Steuerelements eine leere Zeichenkette, wird das Element im Aufgabenbereich unterdrückt. Es ist deshalb wichtig, allen Steuerelementen eine Beschriftung zuzuweisen, auch wenn sie nicht sichtbar ist.

**HINWEIS**

Diese Eigenschaft stellt einige Parameter zur Verfügung, die noch nicht vorgestellt wurden: `ApplicationName`, `Text`, `XML` sowie `Target`. Da eine Smart Document-Lösung in Excel wie auch Word geladen werden kann, wird mit `ApplicationName` der Name der aktuellen Anwendung geprüft.

`Text` und `XML` geben den Inhalt des mit dem Steuerelement verbundenen XML-Elements im Dokument zurück. Das Objekt `Target` kann in Word ein Bereich (Range-Objekt) oder Dokument (Document-Objekt) sein, je nach Kontext.

Control  
Type  
FromID

Die Eigenschaft `ControlTypeFromID` legt die Art des Steuerelements (Textfeld, Schaltfläche usw.) im Aufgabenbereich fest:

```
Private Property Get ISmartDocument_ControlTypeFromID(ByVal ControlID As Long,
ByVal ApplicationName As String, ByVal LocaleID As Long) As SmartTagLib.C_TYPE
Select Case ControlID
Case KUNDE_COMBO
ISmartDocument_ControlTypeFromID = C_TYPE_COMBO
Case KUNDE_INSERT_BUTTON, BETREFF_INSERT_BUTTON, _
BESTELLUNGEN_INSERT_BUTTON, BESTELLUNGEN_CALC_BUTTON
ISmartDocument_ControlTypeFromID = C_TYPE_BUTTON
Case SCHEMA_HELP, KUNDE_HELP, BETREFF_HELP, BESTELLUNGEN_HELP
ISmartDocument_ControlTypeFromID = C_TYPE_HELP
Case BESTELLUNGEN_KUNDE_CAPTION, BESTELLUNGEN_KUNDE_LABEL
ISmartDocument_ControlTypeFromID = C_TYPE_LABEL
Case BESTELLUNGEN_OPTIONS
ISmartDocument_ControlTypeFromID = C_TYPE_RADIOGROUP
Case KUNDE_SEPARATOR1, BETREFF_SEPARATOR1, BESTELLUNGEN_SEPARATOR1, _
BESTELLUNGEN_SEPARATOR2
ISmartDocument_ControlTypeFromID = C_TYPE_SEPARATOR
Case KUNDE_FIRMA_TEXTBOX, KUNDE_STRASSE_TEXTBOX, KUNDE_ORT_TEXTBOX, _
KUNDE_REGION_TEXTBOX, KUNDE_PLZ_TEXTBOX, KUNDE_LAND_TEXTBOX, _
KUNDE_KONTAKTPERSON_TEXTBOX, KUNDE_POSITION_TEXTBOX, BETREFF_TEXTBOX, _
BESTELLUNGEN_ZAHL_TEXTBOX, BESTELLUNGEN_CALC_TEXTBOX
ISmartDocument_ControlTypeFromID = C_TYPE_TEXTBOX
Case Else
End Select
End Property
```

Eine Übersicht der verschiedenen Eigenschaften, die die Steuerelemente dieses Beispiels bezeichnen, fasst die Tabelle 30.4 zusammen. Dem *Type-Namen* müsste der Namensraum vorangestellt werden, Beispiel: »<http://www.KAP30Beispiel.com/BSP02#Kunde>«.

**Tabelle 30.4**

In diesem Beispiel verwendete Elementnamen, Element-IDs, Controls, ControlIndex-Wertes, ControlIDs sowie Controlnamen

Type-Name	Type ID	Control	Control Index	Control ID	Control Name
#actionPertainsTo-EntireSchema	1	SCHEMA_HELP	1	101	"101"
#Kunde	2	KUNDE_HELP	1	201	"201"

**Tabelle 30.4** In diesem Beispiel verwendete Elementnamen, Element-IDs, Controls, ControllIndex-Wertes, ControlIDs sowie Controlnamen (Fortsetzung)

Type-Name	Type ID	Control	Control Index	Control ID	Control Name
		KUNDE_SEPARATOR1	2	202	"202"
		KUNDE_COMBO	3	203	"203"
		und so weiter, bis ...	bis ...	bis ...	
		KUNDE_INSERT_BUTTON	12	212	"212"
#Betreff	3	BETREFF_HELP	1	301	"301"
		BETREFF_SEPARATOR1	2	302	"302"
		BETREFF_TEXTBOX	3	303	"303"
		BETREFF_INSERT_BUTTON	3	304	"304"
#Bestellungen	4	BESTELLUNGEN_HELP	1	401	"401"
		BESTELLUNGEN_SEPARATOR1	2	402	"402"
		BESTELLUNGEN_KUNDE_CAPTION	3	403	"403"
		und so weiter, bis ...	bis ...	bis ...	
		BESTELLUNGEN_CALC_TEXTBOX	10	410	"410"

## Die Interaktion mit dem *SmartDocument*-Interface

In diesem Abschnitt wird die Wirkungsweise der Lösung etwas näher unter die Lupe genommen. Alles beginnt mit der Aktivierung eines Erweiterungspakets durch seine Zuweisung an das aktuelle Dokument bzw. durch Öffnen oder Erstellen eines damit verbundenen Dokuments.

Ereignis  
SmartDoc  
Initia-  
lize

Zunächst wird das SmartDocInitialize-Ereignis der Lösung ausgelöst. Darin werden alle Vorbereitungen durchgeführt. (Wird aus irgendeinem Grund das Erweiterungspaket nicht aufgerufen, finden diese Handlungen nicht statt und das Dokument öffnet sich ohne Aufgabenbereich.)

Im vorliegenden Beispiel werden den globalen Variablen `strSolutionPath` sowie `objDocument` ihre Anfangswerte zugewiesen. Im Dokument wird die Einfügemarke in den Knotenpunkt Kunde gesetzt und die Anfangswerte in Dokumentvariablen geschrieben:

```

Private Sub ISmartDocument_SmartDocInitialize(ByVal ApplicationName As String, _
    ByVal Document As Object, ByVal SolutionPath As String, _
    ByVal SolutionRegKeyRoot As String)
' Schaltet die Fehlermeldung aus, falls kein Kunde-Element vorhanden ist
On Error Resume Next
' Pfadangabe zu den UDL- sowie Temp-Dateien
strSolutionPath = SolutionPath
Set objDocument = Document
With objDocument
    .SelectSingleNode("//ns:Kunde", "xmlns:ns='" & NAMESPACE & "'").Range.Select
    .Variables.Add Name:="Kunden-Code", Value:=" "
    .Variables.Add Name:="Kunde", Value:=" "
End With
Set objDocument = Nothing
varKunden = var1
bPlannedKundeMaintenance = False
End Sub

```

Anschließend werden die Steuerelemente, wie im Abschnitt »Die Steuerelemente des Aufgabenbereichs« in diesem Kapitel erklärt, vorbereitet und die für die gegenwärtige Markierung (in diesem Fall das Element Kunde) passenden im Aufgabenbereich angezeigt. Die Lösung steht nun für den Anwender bereit.

**Element-Ereignisse** Bei jeder Verschiebung der Einfügemarke prüft Word erneut, in welchem Element sich diese nun befindet. Klickt der Anwender beispielsweise in das Element Ort, steht sie in einer verschachtelten Hierarchie bestehend aus Ort, Adresse, Kunde und Brief. Das letztere Element befindet sich schließlich im Dokument (im Pseudo-Element `actionPertainsToEntireSchema`). Die Smart Document-Lösung baut dann den Aufgabenbereich auf, angefangen mit der äußersten Ebene – dem Dokument – bis zum letzten Element der Hierarchie. Dabei wird die entsprechende Methode zur Bereitstellung jedes Steuerelements ausgeführt.

Bei der nächsten Verschiebung der Einfügemarke wiederholt sich das Ganze von vorne. Falls der Zielbereich ein Element ohne vordefinierte Steuerelemente ist, bleibt dieser Teil des Aufgabenbereichs leer.

Die Methoden für die Bereitstellung sind nach Art des Steuerelements gegliedert. Es handelt sich um `PopulateActiveXProps`, `PopulateCheckBox`, `PopulateDocumentFragment`, `PopulateHelpContent`, `PopulateImage`, `PopulateListBoxOrComboContent`, `PopulateRadioGroup` sowie `PopulateTextBox`. Schaltflächen (Buttons), Bezeichnungsfelder (Labels), Links und Trennlinien (Separator) teilen die Methode `PopulateOther`. Einige der Methoden dieses Beispiels, die die Wirkungsweise sowie einige Besonderheiten veranschaulichen, sind im Abschnitt »Methoden für die Bereitstellung von Steuerelementen« in diesem Kapitel vorgestellt.

**OnPaneUpdateComplete** Nachdem die Steuerelemente bereitstehen, wird die Ereignismethode `OnPaneUpdateComplete` ausgeführt. Im vorliegenden Beispiel soll sie sicherstellen, dass die XML-Tags ausgeschaltet sind. Um dem Anwender zu ermöglichen, diese doch eingeschaltet zu lassen (so dass beispielsweise die Struktur der Vorlage bearbeitet werden kann), wird die Einstellung in einer Dokumenteigenschaft niedergeschrieben. Um die XML-Tags eingeblendet zu lassen, muss der Wert der benutzerdefinierten Eigenschaft `ShowXMLMarkup` »aus« anstatt »ein« betragen.

```
Private Sub ISmartDocument_OnPaneUpdateComplete(ByVal Document As Object)
    Dim objDocument As Word.Document
    Set objDocument = Document
    With objDocument
        On Error Resume Next
        ' Vorhandene Werte werden nicht überschrieben, es soll
        ' jedoch sichergestellt werden, dass ein Wert vorhanden ist
        .CustomDocumentProperties.Add Name:="ShowXMLMarkup", Value:="aus"
        On Error GoTo 0
        Select Case .CustomDocumentProperties("ShowXMLMarkup").Value
            Case "aus"
                .ActiveWindow.View.ShowXMLMarkup = False
            Case "ein"
                .ActiveWindow.View.ShowXMLMarkup = True
            Case Else
            End Select
        End With
        Set objDocument = Nothing
    End Sub
```

Steuerelement-Ereignisse

Ab diesem Zeitpunkt stehen die Ereignisse der Steuerelemente – die Aktionen – zur Verfügung. Wird beispielsweise auf eine Schaltfläche geklickt oder ein Eintrag aus einer Liste gewählt, kann die Lösung darauf reagieren. Smart Documents umfassen folgende Ereignisse: `ImageClick`, `OnCheckBoxChange`, `OnListOrComboSelectChange`, `OnRadioGroupSelectChange` sowie `OnTextBoxContentChange`. Das Anklicken einer Schaltfläche (Button), eines Links oder Dokumentfragments führt die Methode `InvokeControl` aus. Einige dieser Ereignis-Prozeduren sind im Abschnitt »Steuerelementereignisse« in diesem Kapitel erläutert.

## Methoden für die Bereitstellung von Steuerelementen

PopulateHelpContent

Die Methode `PopulateHelpContent` sorgt für die Hilfetexte im Aufgabenbereich *Dokumentaktionen*. Sie können sprachspezifisch, nach der Word-Umgebungssprache (LCID), angezeigt werden. Der Hilfetext muss als HTML bereitgestellt werden. Dies bedeutet, dass Sonderzeichen wie Umlaute als Zeichen-Entitäten angegeben werden müssen (siehe Listing 30.7).

Beachten Sie, dass der Hilfetext über den Parameter `Content`, der durch `ByRef` an die Methode übergeben wurde, zurückgegeben wird.

Listing 30.7

Auszug der `PopulateHelpContent`-Methode der Beispiel-Lösung

```
Private Sub ISmartDocument_PopulateHelpContent( ByVal ControlID As Long, _
    ByVal ApplicationName As String, ByVal LocaleID As Long, _
    ByVal Text As String, ByVal Xml As String, ByVal Target As Object, _
    ByVal Props As SmartTagLib.ISmartDocProperties, Content As String) _

    Select Case LocaleID
        Case 1031 ' Deutsch
            Select Case ControlID
                Case SCHEMA_HELP
                    Content = "<html><body><ol><li>Einen Kunden ausw&uuml;hlen. Wo n&ouml;mtig," & _
                        " die Einzelheiten &uuml;ndern. Anschlie&szlig;end 'Kundendetails " & _
                        " einf&uuml;gen' klicken, um die Informationen einzuf&uuml;gen.</li>" & _
                        "<li>Den Betreff eingeben, dann die Schaltfl&uuml;che bet&uuml;tigen," & _
                        " um ihn einzuf&uuml;gen.</li>" & _
```



Listing 30.7 Auszug der *PopulateHelpContent*-Methode der Beispiel-Lösung (Fortsetzung)

```

        "<li>Die Anzahl Bestelldatens&uml;tze festlegen: keine, alle," & _
        " die &uml;testen oder die neuesten (nach Bestelldatum).</li>" & _
        "<li>Den restlichen Inhalt direkt ins Dokument eingeben und formatieren:" & _
        " Anrede, Flie&szlig;text, usw.</li></ol></body></html>"
    Case KUNDE_HELP
        'Definition folgt hier
    Case BETREFF_HELP
        'Definition folgt hier
    Case BESTELLUNGEN_HELP
        Content = "<html><body><p>Sie k&ouml;nnen f&uuml;r diesen Kunden keine," & _
            " alle, die &uml;testen oder die neuesten Bestellungen " & _
            " einf&uuml;gen. Bei Auswahl der &uml;testen oder die neuesten " & _
            " bitte die Anzahl Datens&uml;tze eingeben, sonst wird 1 (eins)" & _
            " angenommen.</p></body></html>"
    Case Else
    End Select
    Case Else ' Englisch
    End Select
End Sub

```

Populate  
ListOr  
Combo  
Content

Die Methode *PopulateListOrComboContent* in Listing 30.8 veranschaulicht, wie Steuerelement-Eigenschaften («Properties» eines »Property Bag») festgelegt werden. Es gibt etwa 20 Eigenschaften für Steuerelemente des Aufgabenbereichs *Dokumentaktionen*, die beispielsweise die Größe, Position, Ausrichtung und Schrift festlegen.

```

Props.Write Key:="IsEditable", Value:="false"      'kann geändert werden
Props.Write Key:="ControlOnSameLine", Value:="False" 'mit in der gleichen Zeile
Props.Write Key:="W", Value:="200"                 'Breite

```

Für mehr Informationen schlagen Sie bitte im SDK nach.

Listing 30.8 Füllt die Dropdownliste im Abschnitt für das Element *Kunde* mit Kundeninformationen aus einer Datenbank

```

Private Sub ISmartDocument_PopulateListOrComboContent(ByVal ControlID As Long, _
    ByVal ApplicationName As String, ByVal LocaleID As Long, ByVal Text As String, _
    ByVal Xml As String, ByVal Target As Object, _
    ByVal Props As SmartTagLib.ISmartDocProperties, List() As String, Count As Long, _
    InitialSelected As Long)

    Dim lngRowCount As Long
    Dim lngIndex As Long
    Dim objRecordset As ADODB.Recordset

    On Error GoTo ErrorHandler
    Select Case ControlID
        Case KUNDE_COMBO
            If Not bPlannedKundeMaintenance Then
                Set objRecordset = New ADODB.Recordset
                ' Die Kundendaten können getrennt gelesen werden. Wenn man versucht, die Daten in
                ' ein Datenfeld des Typs Variant zu lesen, soll das Vorkommen von Null-Werten
                ' geprüft werden. Sonst kann ADO unerwartete Ergebnisse liefern.

```

**Listing 30.8** Füllt die Dropdownliste im Abschnitt für das Element *Kunde* mit Kundeninformationen aus einer Datenbank (Fortsetzung)

```

objRecordset.Open
    Source=" SELECT [Kunden-Code]," & "[Kunden-Code] + ': ' + Firma as `X`," & _
        " Firma, iif(isnull([Straße]),',[Straße])," & _
        " iif(isnull(Ort),',[Ort]," & " iif(isnull(Region),',[Region]," & _
        " iif(isnull(PLZ),',[PLZ]," & " iif(isnull(Land),',[Land]," & _
        " iif(isnull(Kontaktperson),',[Kontaktperson]," & _
        " iif(isnull([Position]),',[Position])" & " FROM Kunden ORDER BY 1", _
    ActiveConnection:=strConnectionPrefix & strSolutionPath & _
    strConnectionUDL, CursorType:=adOpenForwardOnly
If objRecordset.EOF And objRecordset.BOF Then
    ReDim List(1 To 1)
    Select Case LocaleID
        Case 1031 'Deutsch
            List(1) = "Keine Kundendatensätze gefunden"
        Case Else 'Englisch
            End Select
    Else
        varKunden = var1
        varKunden = objRecordset.GetRows
        Count = UBound(varKunden, 2) + 1
        ReDim List(1 To Count) As String
        For lngIndex = 1 To Count
            List(lngIndex) = varKunden(1, lngIndex - 1)
        Next
    End If
    objRecordset.Close
    Set objRecordset = Nothing
    InitialSelected = -1
    Props.Write Key:="IsEditable", Value:="false"
    Props.Write Key:="ControlOnSameLine", Value:="False"
    Props.Write Key:="W", Value:="200"
End If
Case Else
End Select

Finally:
Exit Sub

ErrorHandler:
    MsgBox Err.Description
    Resume Finally
End Sub

```

Populate  
Radio  
Group

Optionsschaltflächen können auch sprachspezifisch bezeichnet werden, wie das Listing 30.9 veranschaulicht. Sie werden in der Reihenfolge der Definition angezeigt. Die Eigenschaft `InitialSelected` legt fest, welche standardmäßig aktiviert ist; der Wert `-1` bedeutet »keine«. Wie festgestellt wird, welche Option (wenn überhaupt) gewählt wurde, veranschaulicht das Listing 30.13.

**Listing 30.9** Optionsschaltflächen für den Abschnitt des Elements *Bestellungen* definieren

```

Private Sub ISmartDocument_PopulateRadioGroup(ByVal ControlID As Long, _
    ByVal ApplicationName As String, ByVal LocaleID As Long, ByVal Text As String, _
    ByVal Xml As String, ByVal Target As Object, _

```

**Listing 30.9** Optionsschaltflächen für den Abschnitt des Elements *Bestellungen* definieren (Fortsetzung)

```

ByVal Props As SmartTagLib.ISmartDocProperties, _
List() As String, Count As Long, InitialSelected As Long)

Dim intIndex As Integer
Select Case ControlID
    Case BESTELLUNGEN_OPTIONS
        Count = 4
        ReDim List(1 To Count) As String
        Select Case LocaleID
            Case 1031 ' Deutsch
                List(1) = "Keine"
                List(2) = "Alles"
                List(3) = "Älteste"
                List(4) = "Neueste"
                InitialSelected = -1
            Case Else ' Englisch
        End Select
    Case Else
End Select
End Sub

```

## Steuerelementereignisse

In diesem Abschnitt werden einige der Steuerelementereignisse der Beispiel-Lösung vorgestellt. Bitte beachten Sie, dass eine einzige Prozedur für alle Steuerelemente des gleichen Typs zuständig ist. Der Code für die einzelnen Steuerelemente wird anhand des ControlID-Parameters in einer Select Case-Anweisung festgelegt.

OnListOr  
Combo  
Select  
Change

Die Methode OnListOrComboSelectChange wird ausgeführt, wenn der Anwender einen Eintrag aus der Liste auswählt. Der Code in Listing 30.10 ruft die Hilfsprozedur SetTextboxText auf, die die Datenbankdaten des ausgewählten Kunden in die entsprechenden Textfelder schreibt.

**Listing 30.10** Der Code »hinter« der Dropdownliste des Abschnitts für das Element *Kunde*

```

Private Sub ISmartDocument_OnListOrComboSelectChange(ByVal ControlID As Long, _
    ByVal Target As Object, ByVal Selected As Long, ByVal Value As String)

    Select Case ControlID
        Case KUNDE_COMBO
            ' Alle Textfelder des Kunde-Abschnitts im Aufgabenbereich mit Daten füllen
            SetTextboxText Target, KUNDE_FIRMA_TEXTBOX, CStr(varKunden(2, Selected - 1))
            SetTextboxText Target, KUNDE_STRASSE_TEXTBOX, CStr(varKunden(3, Selected - 1))
            SetTextboxText Target, KUNDE_ORT_TEXTBOX, CStr(varKunden(4, Selected - 1))
            SetTextboxText Target, KUNDE_REGION_TEXTBOX, CStr(varKunden(5, Selected - 1))
            SetTextboxText Target, KUNDE_PLZ_TEXTBOX, CStr(varKunden(6, Selected - 1))
            SetTextboxText Target, KUNDE_LAND_TEXTBOX, CStr(varKunden(7, Selected - 1))
            SetTextboxText Target, KUNDE_KONTAKTPERSON_TEXTBOX, CStr(varKunden(8, _
                Selected - 1))
            SetTextboxText Target, KUNDE_POSITION_TEXTBOX, CStr(varKunden(9, Selected - 1))
            lngKundenSelectedRow = Selected
        Case Else
    End Select
End Sub

```

Methode Da alle Schaltflächen über die Methode `InvokeControl` gesteuert werden, wurden die eigentlichen Handlungen der Methode in Listing 30.11 in Hilfsprozeduren ausgelagert. Zwei davon, die wichtige Aspekte veranschaulichen, werden weiter unten vorgestellt.

**Listing 30.11** Die Methode `InvokeControl` wird durch Anklicken einer Schaltfläche im Aufgabenbereich *Dokumentaktionen* ausgelöst

```
Private Sub ISmartDocument_InvokeControl(ByVal ControlID As Long, _
    ByVal ApplicationName As String, ByVal Target As Object, ByVal Text As String, _
    ByVal Xml As String, ByVal LocaleID As Long)

    On Error Resume Next

    ' Der Anwender darf die Vorlage nicht als Dokument verwenden
    If Target.Parent.Type = wdTypeTemplate Then
        Select Case LocaleID
            Case 1031 ' Deutsch
                MsgBox "Sie dürfen in der Lösung-Vorlage" & _
                    " (.dot) keinen Text eingeben. Bitte davon ein neues" & _
                    " Word-Dokument (.doc) erstellen.", vbOKOnly, "Word-Vorlage"
            Case Else ' Englisch
                ' ...
        End Select
    End If

    Select Case ControlID
        Case KUNDE_INSERT_BUTTON
            Invoke_KUNDE_INSERT_BUTTON Target, LocaleID
        Case BETREFF_INSERT_BUTTON
            Invoke_BETREFF_INSERT_BUTTON Target, LocaleID
        Case BESTELLUNGEN_INSERT_BUTTON
            Invoke_BESTELLUNGEN_INSERT_BUTTON Target, LocaleID
        Case BESTELLUNGEN_CALC_BUTTON
            Invoke_BESTELLUNGEN_CALC_BUTTON Target, LocaleID
        Case Else
            ' ...
    End Select
End Sub
```

Invoke\_  
KUNDE\_  
INSERT\_  
BUTTON

Die Prozedur in Listing 30.12 veranschaulicht, wie Werte aus anderen Steuerelementen des Aufgabenbereichs gelesen werden. Bitte beachten Sie, dass unsichtbare Steuerelemente nicht zugänglich sind. Es ist beispielsweise nicht möglich, den Inhalt eines Steuerelements im Abschnitt für das Element *Bestellungen* zu lesen, wenn sich die Einfügemarke im Element *Kunde* befindet.

Der Parameter `ControlID` in der rufenden Prozedur `InvokeControl` stellt das Steuerelement dar, das die Methode ausgelöst hat (in diesem Fall die Schaltfläche `KUNDE_INSERT_BUTTON` (»Kundendetails einfügen«)). Es ist jedoch notwendig, die Werte der Combobox sowie der Textfelder zu lesen. Um dies zu tun, muss der Dokumentbereich des *Kunde*-Elements angesprochen werden, was über den Parameter `Target` (stellt den Bereich des gegenwärtigen Elements dar) möglich ist.

Dieser Bereich stellt ein `SmartTag`-Objekt zur Verfügung, das seinerseits eine Auflistung `SmartTagActions` bereitstellt. Diese stellen die Steuerelemente des Aufgabenbereichs *Dokumentaktionen* dar, die mit dem Element verbunden sind. Eine `SmartTagAction` wird über die `ControlNameFromID`-Eigenschaft angesprochen. Da in diesem Beispiel diese Zeichenketten gleich den Werten der `ControlID` sind (`ISmartDocument_ControlNameFromID = CStr(ControlID)`), können die

Steuerelemente mit den Konstantwerten für die ControlID angesprochen werden. Beispiel: Die folgenden Zeilen ermitteln, welcher Eintrag der Combobox ausgewählt ist:

```
Set rngTarget = Target
Set objXMLNode = rngTarget.XMLNodes(1)
objXMLNode.SmartTag.SmartTagActions(CStr(KUNDE_COMBO)).ListSelection
```

Zudem muss Invoke\_KUNDE\_INSERT\_BUTTON den Textinhalt der Unterknotenpunkte (wie Ort oder Land) des Kunde-Elements schreiben oder lesen. Die Parameter Text und XML der rufenden Prozedur InvokeControl beinhalten den gesamten Text oder XML des Kunde-Elements. Deshalb wird mit der SelectSingleNode-Methode des Word-Objektmodells gearbeitet. Durch eine XPath-Anweisung wird der gewünschte Knotenpunkt (Element) festgelegt und der Text aus seinem Bereich gelesen. Die folgende Codezeile liest den Inhalt des Firma-Elements aus dem Dokument:

```
If Trim$(rngTarget.Document.SelectSingleNode("//ns:Firma", "xmlns:ns='" & _
    NAMESPACE & "'").Text)
```

**Listing 30.12** Beim Anklicken der Schaltfläche *Kundendetails einfügen* werden Daten aus dem Aufgabenbereich in das Dokument geschrieben.

```
Private Sub Invoke_KUNDE_INSERT_BUTTON(ByVal Target As Object, ByVal LocaleID As Long)
    Dim objRecordset As ADODB.Recordset
    Dim objXMLNode As Word.XMLNode
    Dim rngTarget As Word.Range

    ' Target (als Objekt übergeben) ist ein Range-Objekt
    ' Diese Zeile ermöglicht den Einsatz von IntelliSense während der Code-Eingabe
    Set rngTarget = Target
    Set objXMLNode = rngTarget.XMLNodes(1)

    If objXMLNode.SmartTag.SmartTagActions(CStr(KUNDE_COMBO)).ListSelection _
        = 1 Then
        Select Case LocaleID
            Case 1031 'Deutsch
                MsgBox "Bitte wählen Sie in der Liste einen Kundennamen aus.", _
                    vbOKOnly, "Kunde auswählen"
            Case Else 'Englisch
            End Select
        Exit Sub
    End If

    If Trim$(rngTarget.Document.SelectSingleNode(
        "//ns:Firma", "xmlns:ns='" & NAMESPACE & "'").Text) <> "" Then
        Select Case LocaleID
            Case 1031 'Deutsch
                If MsgBox("Im Dokument sind bereits Informationen zum Kunden " & _
                    " sowie seinen Bestellungen vorhanden. Möchten Sie alle " _
                    " Kundeninformationen aus dem Dokument entfernen?", _
                    vbYesNo, "Kunde & Bestellungen überschreiben") = vbNo Then
                    Exit Sub
                End If
            Case Else 'Englisch
            End Select
        End If
    End If
```

**Listing 30.12** Beim Anklicken der Schaltfläche *Kundendetails einfügen* werden Daten aus dem Aufgabenbereich in das Dokument geschrieben. (Fortsetzung)

```
' Bei der Aktualisierung der Kundendaten im Dokument wird der Fokus des
' Aufgabenbereichs geändert. Um dies zu verhindern, halten wir fest,
' ob die Daten im Dokument aktualisiert werden.
bPlannedKundeMaintenance = True

' Kundeninformationen direkt aus den Steuerelementen einfügen,
' falls diese Daten enthalten
SetNodeValueFromTextbox "//ns:Kunde/ns:Firma", objXMLNode, _
    KUNDE_FIRMA_TEXTBOX
SetNodeValueFromTextbox "//ns:Kunde/ns:Adresse/ns:Straße", objXMLNode, _
    KUNDE_STRASSE_TEXTBOX
SetNodeValueFromTextbox "//ns:Kunde/ns:Adresse/ns:Ort", objXMLNode, _
    KUNDE_ORT_TEXTBOX
'''Weitere Zuweisungen folgen

' Felder in den Kundenelementen aktualisieren
rngTarget.Document.SelectSingleNode("//ns:Kunde", "xmlns:ns='" & _
    NAMESPACE & "'").Range.Fields.Update

' Den gegenwärtigen Kunden festhalten, um später seine Bestellungen nachzuschlagen
rngTarget.Document.Variables("Kunden-Code").Delete
rngTarget.Document.Variables("Kunde").Delete
rngTarget.Document.Variables.Add Name:="Kunden-Code", _
    Value:=CStr(varKunden(0, lngKundenSelectedRow - 1)) _
rngTarget.Document.Variables.Add Name:="Kunde", _
    Value:=CStr(varKunden(1, lngKundenSelectedRow - 1))

' Die gegenwärtig aufgelisteten Bestellungen löschen
ClearBestellungen rngTarget.Document, False

' Falls der Betreff leer ist, diesen Knotenpunkt anspringen;
' sonst zum Knotenpunkt Bestellungen gehen
If Trim(rngTarget.Document.SelectSingleNode("//ns:Betreff", "xmlns:ns='" & _
    NAMESPACE & "'").Range.Text) = "" Then
    rngTarget.Document.SelectSingleNode("//ns:Betreff", "xmlns:ns='" & _
        NAMESPACE & "'").Range.Select
Else
    rngTarget.Document.SelectSingleNode("//ns:Bestellungen", "xmlns:ns='" & _
        NAMESPACE & "'").Range.Select
End If

bPlannedKundeMaintenance = False
rngTarget.Document.SmartDocument.RefreshPane
Set rngTarget = Nothing
Set objXMLNode = Nothing
End Sub
```

Invoke\_  
BESTEL-  
LUNGEN\_  
INSERT\_  
BUTTON

Der Großteil dieser Prozedur beschäftigt sich damit, Werte zu verifizieren; diese Codezeilen haben wir in Listing 30.13 weggelassen. Die darin enthaltenen Codezeilen veranschaulichen, wie man feststellt, welche Optionsschaltfläche aktiviert ist, sowie das Lesen und die Transformation der Datenbankdaten für die Bestellungen. (Einige der fehlenden Codezeilen legen Werte fest, die für den Aufbau der SQL-Anweisung (strSQL) benötigt werden.)

Die Angaben zu den ausgewählten Bestellungen werden in einem ADO-Recordset festgehalten, das durch die Save-Methode mit der Option adPersistXML als XML-Datei *Bestellungen.xml* gespeichert wird. Speicherort ist der in strSolutionPath (globale Variable, die in SmartDocInitialize initialisiert wurde) angegebene Ordner.

Über eine *IncludeText*-Feldfunktion wird diese Datei in das Dokument eingefügt. Eine Transformation sorgt dafür, dass die Daten in einer in WordProcessingML definierten Tabelle erscheinen. Anschließend wird die Feldfunktion aufgelöst.

**Listing 30.13** Ein Klick auf die Schaltfläche *Bestellungen einfügen* führt diese Methode aus.

```
Private Sub Invoke_BESTELLUNGEN_INSERT_BUTTON(ByVal Target As Object, _
    ByVal LocaleID As Long)

    Dim intBestellungenZahl As Integer, objRecordset As ADODB.Recordset
    Dim objXMLNode As Word.XMLNode, rngTarget As Word.Range
    Dim strBestellungenZahl As String, strPad1 As String, intOption As Integer
    Dim strPad2 As String, strSequence As String, strSQL As String, strTOP As String

    Set rngTarget = Target
    Set objXMLNode = rngTarget.XMLNodes(1)

    intOption = _
        objXMLNode.SmartTag.SmartTagActions(CStr(BESTELLUNGEN_OPTIONS)).RadioGroupSelection
    If intOption = -1 Then 'Keine Optionsschaltfläche wurde aktiviert
        Select Case LocaleID
            '''Eine sprachspezifische Meldung folgt hier
            End Select
        Exit Sub
    End If

    ''' Hier folgt eine Prüfung, ob sich bereits Bestellungen im Dokument befinden.
    ''' Nötigenfalls wird eine Warnung einblendet.

    'Auswahl der Optionsschaltfläche
    Select Case intOption
        Case 1 ' Keine - die Liste entfernen und ein Leerzeichen eingeben, fertig
            ClearBestellungen rngTarget.Document, True
            Exit Sub
        Case Else ' Sicherstellen, dass ein Kunde vorhanden ist
            ''' Codezeilen für die Prüfung folgen hier
            End Select
            Exit Sub
        End If
    End Select
    Select Case intOption
        Case 3, 4 ' Älteste/Neueste wurde gewählt.
            ''' In den folgenden Codezeilen werden die Anzahl der erwünschten Datensätze geprüft.
            Case Else "Alle" wird angenommen
                strTOP = ""
                strSequence = "ASC"
            End Select

            ' Die Datensätze aus der Datenbank holen
            Set objRecordset = New ADODB.Recordset
            strSQL = " SELECT " & strTOP & "be.[Bestell-Nr]," & _
                " format(be.Bestelldatum,'YYYYMMDD') As `sortable`,`" & _
```

**Listing 30.13** Ein Klick auf die Schaltfläche *Bestellungen einfügen* führt diese Methode aus. (Fortsetzung)

```

" format(be.Bestelldatum,'DD MMM YYYY') As `Bestelldatum`,`" & _
" format(be.Lieferdatum,'DD MMM YYYY') As `Lieferdatum`,`" & _
" format(be.Versanddatum,'DD MMM YYYY') As `Versanddatum`,`" & _
" ve.Firma As `VersandÜber`, iif(isnull(be.Frachtkosten),'0.00`,`" & _
" format(be.Frachtkosten,'#.00')) As `Frachtkosten`,`" & _
" FROM Bestellungen be INNER JOIN Versandfirmen ve" & _
" ON be.VersandÜber = ve.[Firmen-Nr] WHERE be.[Kunden-Code] = '" & _
rngTarget.Document.Variables("Kunden-Code").Value & "'" & _
" ORDER BY 2 " & strSequence
objRecordset.Open Source:=strSQL, _
ActiveConnection:=strConnectionPrefix & strSolutionPath & strConnectionUDL,
CursorType:=adOpenForwardOnly

' Die vorhandene Bestellungen.xml-Datei löschen
' und via ADO das Recordset in eine neue serialisieren
'On Error Resume Next
Kill strSolutionPath & strBestellungenDatei
objRecordset.Save strSolutionPath & strBestellungenDatei, adPersistXML
objRecordset.Close
Set objRecordset = Nothing

' Eine INCLUDETEXT-Feldfunktion in das Element "Bestellungen" einfügen,
' um die XML-Datei in Bestellungen.xml zu transformieren. Die Feldfunktion
' anschließend auflösen.
strPad1 = Replace(strSolutionPath & strBestellungenDatei, "\", "\\")
strPad2 = Replace(strSolutionPath & strBestellungenXSLT, "\", "\\")
ClearBestellungen rngTarget.Document, False
rngTarget.Fields.Add Range:=rngTarget, Type:=wdFieldEmpty,
Text:="INCLUDETEXT """" & strPad1 & """" \t """" & strPad2 & """"", _
preserveformatting:=False
rngTarget.Fields.Unlink
Set rngTarget = Nothing
Set objXMLNode = Nothing
End Sub

```

## XML und die *IncludeText*-Feldfunktion

Die *IncludeText*-Feldfunktion wurde kurz in Kapitel 6 vorgestellt. Damit wird Inhalt anderer, auf Text basierenden Dateien in ein Word-Dokument verknüpft. Es kann sich um ein Text-, Word-, WordPerfect- oder sonstiges Dateiformat handeln, solange Word ein passender Konvertierfilter zur Verfügung steht.

Für die Feldfunktion gab es bis Word 2003 nur einen optionalen Parameter (die Bezeichnung einer Textmarke, um nur einen bestimmten Teil des Dokuments einzubinden) sowie den Schalter \c, womit der Name eines Konvertierfilters festgelegt werden kann.

Seit Word 2003 unterstützt *IncludeText* zusätzlich XML-Dateien. Dafür wurden drei neue Schalter eingeführt:

- \t Legt eine XSL-Transformation fest.
- \x Legt einen XPath-Ausdruck fest, womit nur die Daten aus den angegebenen Knotenpunkten ins Dokument geholt werden. ►



- \nStellt die für den XPath-Ausdruck benötigten Namensräume bereit.

Im Fall einer XML-Datei beträgt der Wert des Schalters \c »XML«.

Beim Einfügen einer XML-Datei verhält sich Word, als würde sie über *Datei/Öffnen* geöffnet. Besteht sie aus einem eigenen XML-Vokabular, erscheinen die XML-Tags. Ist sie mit einer Transformation verbunden, wird diese ausgeführt. Falls die Datei gültiges WordProcessingML enthält, erscheint der Inhalt wie ein Word-Dokument.

Die Feldfunktion, die die DLL dieses Beispiels generiert, sieht folglich etwa so aus:

```
{ IncludeText "c:\\Beispiele\\Kap30\\BSP30_02.xml"
  \t "c:\\Beispiele\\Kap30\\BSP30_02.xml" \c XML }
```

## Die Transformation

Wie in der Diskussion zu Listing 30.13 erwähnt, werden die Informationen zu den Bestellungen über eine *IncludeText*-Feldfunktion ins Dokument geholt. Eine Transformation wandelt diese in eine Tabelle um.

Diese Transformation (*Bsp30\_2.xsl*) basiert auf der des MOSTL-Beispiels. Sie musste jedoch einige Unterschiede berücksichtigen:

- Die XML-Datei mit den Daten hat eine andere Struktur (von ADO herkommend) als andere XML-Dateien, die in diesem Buch bislang vorgestellt wurden.
- Die Formatierung wurde verfeinert, so dass sich die Tabelle besser in die Gestaltung des Briefs einpasst.
- Überflüssige WordProcessingML wurde entfernt. Beim Speichern eines Word-Dokuments als WordProcessingML werden beispielsweise viele Formatvorlagendefinitionen mitgespeichert, die für das transformierte Ergebnis unerheblich sind. Es wurden auch alle von der Transformation nicht benötigten Namensräume entfernt.

Es folgen einige Auszüge aus der Transformationsdatei *Bsp30\_02.xsl*, die diese Unterschiede und andere Besonderheiten veranschaulichen.

Nur die tatsächlich gebrauchten Namensräume werden deklariert. Diese sind, neben dem standardmäßigen für eine Transformation, der für WordProcessingML sowie zwei, die ADO in die XML-Datei einbaut (mehr dazu weiter unten):

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:w="http://schemas.microsoft.com/office/word/2003/wordml"
  xmlns:ns2="http://www.KAP30Beispiel.com/BSP02"
  xmlns:rs="urn:schemas-microsoft-com:rowset"
  xmlns:z="#RowsetSchema">
```

Wie erfahrene Word-Benutzer wissen, kann das Einfügen von Text aus einem Word-Dokument in ein anderes zu Unstimmigkeiten in Formatierungen führen. Die Formatvorlagen des Zieldokuments überschreiben die Definition der Formatvorlagen gleichen Namens im Quelldokument. Deshalb wird für die Tabelle eine Formatvorlage deklariert, die im Brief nicht vorkommt (*Bsp3002\_th*).

```
<w:styles>
  <w:style w:type="paragraph" w:styleId="bsp3002_th">
    <w:name w:val="bsp3002_th" />
  </w:style>
```

Dafür werden keine Formatierungen festgelegt; diese werden den Tabellenzellen direkt zugewiesen. Das folgende Codefragment veranschaulicht die Rahmen- und Schattierungsformatierungen der Tabelle:

```
<w:tbl>
  <w:tblPr>
    <w:tblBorders>
      <w:top w:val="single" w:sz="4" w:space="0" w:color="auto"/>
      <w:left w:val="single" w:sz="4" w:space="0" w:color="auto"/>
      <w:bottom w:val="single" w:sz="4" w:space="0" w:color="auto"/>
      <w:right w:val="single" w:sz="4" w:space="0" w:color="auto"/>
      <w:insideH w:val="single" w:sz="4" w:space="0" w:color="auto"/>
      <w:insideV w:val="single" w:sz="4" w:space="0" w:color="auto"/>
    </w:tblBorders>
    <w:tblCellMar>
      <w:top w:w="0" w:type="dxa"/>
      <w:left w:w="108" w:type="dxa"/>
      <w:bottom w:w="0" w:type="dxa"/>
      <w:right w:w="108" w:type="dxa"/>
    </w:tblCellMar>
  </w:tblPr>
```

Den Aufbau der XML-Datei bestimmen die Richtlinien, denen ADO folgt, wenn bei der Speicherung die Option `adPersistXML` aktiv ist. Diese sind u.a.:

- Ein Wurzelement namens `<xml>`
- Ein Schema-Element, das alle Felder deklariert
- Feldnamen mit Zeichen wie Umlaute oder Striche werden ersetzt, `Bestell-Nr` wird also zu `c0`.
- Ein Data-Element, worin ein Row-Element für jeden Datensatz verschachtelt ist. Die Felder, samt ihrem Inhalt, sind dann Attribute des Row-Elements, wie folgender Auszug veranschaulicht. (Die Codezeilen, die das Schema definiert, wurden weggelassen, da sie auf die Transformation keinen Einfluss haben).

```
<xml xmlns:s='uuid:BDC6E3F0-6DA3-11d1-A2A3-00AA00C14882'
  xmlns:dt='uuid:C2F41010-65B3-11d1-A29F-00AA00C14482'
  xmlns:rs='urn:schemas-microsoft-com:rowset'
  xmlns:z='#RowsetSchema'>
  <rs:data>
    <z:row c0='10643' sortable='19970825' Bestelldatum='25-Aug-1997'
      Lieferdatum='22-Sep-1997' Versanddatum='02-Sep-1997' c5='Speedy Express'
      Frachtkosten='29.46' />
  </rs:data>
</xml>
```

Das bedeutet, die Werte für die Tabellenzellen kommen aus Attributen statt aus Elementen. Der Wert eines Attributs wird wie folgt (mit einem »@« vor dem Elementnamen) gelesen:

```
<xsl:value-of select="@Bestelldatum" />
```

## Das Schema

Ein Teil des Schemas (*Bsp30\_02.xsd*) befindet sich in Listing 30.14. Es legt für die Lösung <Brief> als Wurzelement fest. Dieses beinhaltet ein Kunde-Element, worin eine Sequenz Bestellungen-Elemente verschachtelt ist. Ihrerseits umfassen diese je eine Sequenz Bestellung-Elemente.

Das Bestellung-Element wurde mit Kindelementen wie Bestell-Nr und Bestelldatum ausgestattet, obwohl diese für die vorliegende Lösung nicht notwendig wären. Sie ermöglichen jedoch eine Erweiterung mit Aktionen für diese Elemente, falls dies jemals wünschenswert ist.

Für diese sowie für die Auflistung der Kindelemente des Kunde-Elements (wie Firma oder Adresse) wurde keine Sequenz festgelegt, sondern von der Anweisung <all> Gebrauch gemacht. Damit kann deren Reihenfolge frei erfolgen, was die Gestaltungsmöglichkeiten des Dokuments erhöht.

Obwohl viele der Datenbankfelder numerische oder Datums-Datentypen enthalten, wurden die Elemente hauptsächlich als Datentyp *string* (Zeichenkette) definiert, um Formatierungs- und lokalen Problemen auszuweichen. Da Word sowieso alles als Text betrachtet, stellt dies keinen Nachteil dar. (Beispiel: In einigen Ländern wird ein Komma als Dezimalzeichen eingesetzt, während andere einen Punkt benötigen. Wenn im Schema eine Zahl als Datentyp *double* definiert wird, könnte das Dokument je lokaler Einstellung als ungültig betrachtet werden, weil das Dezimalzeichen nicht erkannt wurde.)

Listing 30.14 Das Schema für die Smart Document-Vorlage

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.KAP30Beispiel.com/BSP02"
  xmlns="http://www.KAP30Beispiel.com/BSP02"
  elementFormDefault="qualified">
  <xsd:element name="Brief">
    <xsd:complexType mixed="true">
      <xsd:sequence>
        <xsd:element name="Kunde" minOccurs="1">
          <xsd:complexType mixed="true">
            <xsd:all>
              <xsd:element name="Firma" type="xsd:string" minOccurs="1" />
              <xsd:element name="Adresse" type="typAdresse" minOccurs="1" />
              <xsd:element name="Kontaktperson" type="xsd:string" minOccurs="0" />
              <xsd:element name="Position" type="xsd:string" minOccurs="0" />
            </xsd:all>
          </xsd:complexType>
        </xsd:element>
        <xsd:element name="Betreff" type="xsd:string" minOccurs="1" />
        <xsd:element name="Bestellungen" minOccurs="0" maxOccurs="1" >
          <xsd:complexType mixed="true">
            <xsd:sequence>
              <xsd:element name="Bestellung" minOccurs="0" maxOccurs="unbounded" >
                <xsd:complexType mixed="true">
```

**Listing 30.14** Das Schema für die Smart Document-Vorlage (Fortsetzung)

```

        <xsd:all>
          <xsd:element name="Bestell-Nr" type="xsd:string" minOccurs="1" />
          <xsd:element name="Bestelldatum" type="xsd:string" minOccurs="1" />
          <xsd:element name="Lieferdatum" type="xsd:string" minOccurs="1" />
          <xsd:element name="Versanddatum" type="xsd:string" minOccurs="1" />
          <xsd:element name="Versandüber" type="xsd:string" minOccurs="1" />
          <xsd:element name="Frachtkosten" type="xsd:string" minOccurs="1" />
        </xsd:all>
      </xsd:complexType>
    </xsd:element>
  </xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
<!-- Die Definitionen der typAdresse-Elemente folgen hier. Es umfasst fünf Kindelemente des
Datentyps "string" -->

</xsd:schema>

```

## Das Erweiterungspaket-Manifest

Die vollständige Manifest-Datei (*BSP30\_02M.xml*) des Erweiterungspakets befindet sich in Listing 30.15. Der Großteil der Elemente ist im Abschnitt »Wie es funktioniert« unter »Ein MOSTL-Smart Document« in diesem Kapitel beschrieben. Das Manifest enthält zwei *solution*-Elemente, beide mit der gleichen *solutionID*; eines für das Schema und eines für die Smart Document-Lösung, welche drei *file*-Elemente auflistet:

- die DLL, die den Dateityp (*fileType*) »*solutionActionHandler*« hat
- die UDL, die die Pfadangabe zur Nordwind-Datenbank enthält
- die XSL-Transformation. Da in diesem Fall die Transformation nicht mit der Lösung geladen wird, wie im MOSTL-Beispiel, muss sie sich nicht in einem eigenen *solution*-Element befinden.

Für die COM-DLL müssen die *CLSID*- sowie *regsvr32*-Elemente angegeben werden. Diese stellen sicher, dass die DLL bei der Installation korrekt in der Registry eingetragen wird. Um die *CLSID* zu ermitteln, suchen Sie nach »[Projektname].[Klassname]« (in diesem Fall *Bsp02.SmartDoc*) im Abschnitt *HKEY\_CLASSES\_ROOT* und kopieren Sie den Wert seines *Clsid*-Schlüssels.

**Listing 30.15** Das Manifest

```

<SD:manifest xmlns:SD="http://schemas.microsoft.com/office/xmlexpansionpacks/2003">
  <SD:version>1.0</SD:version>
  <SD:updateFrequency>20160</SD:updateFrequency>
  <SD:uri>http://www.KAP30Beispiel.com/BSP02</SD:uri>
  <SD:solution>
    <SD:solutionID>{A6000394-B5D8-4502-B9EA-2672AD55076F}</SD:solutionID>
    <SD:type>schema</SD:type>
    <SD:alias lcid="0">BSP02</SD:alias>
    <SD:file>
      <SD:type>schema</SD:type>
      <SD:version>1.0</SD:version>
    </SD:file>
  </SD:solution>

```

Listing 30.15 Das Manifest (Fortsetzung)

```

    <SD:filePath>BSP30_02.xsd</SD:filePath>
  </SD:file>
</SD:solution>
<SD:solution>
  <SD:solutionID>{A6000394-B5D8-4502-B9EA-2672AD55076F}</SD:solutionID>
  <SD:type>smartDocument</SD:type>
  <SD:alias lcid="1031">Nordwind Bestellungen Brief</SD:alias>
  <SD:alias lcid="1033">Northwind Order Letter</SD:alias>
  <SD:alias lcid="0">Northwind Order Letter</SD:alias>
  <SD:targetApplication>Word.Application.11</SD:targetApplication>
  <SD:file>
    <SD:type>solutionActionHandler</SD:type>
    <SD:version>1.0</SD:version>
    <SD:filePath>BSP30_02.DLL</SD:filePath>
    <SD:CLSID>{4A66510E-4960-45D0-B8D8-F6D830B4C39E}</SD:CLSID>
    <SD:regsvr32/>
  </SD:file>
  <SD:file>
    <SD:type>other</SD:type>
    <SD:version>1.0</SD:version>
    <SD:filePath>BSP30_02.UDL</SD:filePath>
  </SD:file>
  <SD:file>
    <SD:type>other</SD:type>
    <SD:version>1.0</SD:version>
    <SD:filePath>BSP30_02.XSL</SD:filePath>
  </SD:file>
</SD:solution>
</SD:manifest>

```

## Die DLL in VB6 erstellen

Folgen Sie den in Kapitel 29 beschriebenen Angaben für die Erstellung eines Smarttag-DLLs, um das Projekt zu erstellen. Das Beispiel-Projekt enthält Verweise zu den folgenden Objektbibliotheken:

- Microsoft Smart Tags 2.0 (für eine Smart Document-Lösung erforderlich)
- Microsoft Word 11.0 (erforderlich, um mit dem Word-Objektmodell zu arbeiten)
- Microsoft ActiveX Data Objects 2.8 (für die Verbindung zur Datenbank)

Das SmartDocument-Interface muss implementiert werden. Gehen Sie wie folgt vor:

1. Geben Sie am Anfang des von VB erstellten Klassenmoduls, nach der Deklaration Option Explicit, im Teil (*Allgemein*) (*Deklarationen*) folgende Zeile ein:

```
Implements SmartTagLib.ISmartDocument
```

2. Wählen Sie aus der *Objekt*-Liste (dort, wo »Allgemein« steht) den Eintrag *ISmartDocument*.
3. Jede Methode und Eigenschaft eines Interfaces muss implementiert werden, auch wenn Sie nicht beabsichtigen, diese in Ihrer Lösung einzusetzen. Wählen Sie also aus der *Prozedur*-Liste (dort, wo »Deklarationen« stand) jeden Eintrag, einen nach dem anderen, aus. Visual Basic fügt das Prozedurskelett in das Klassenmodul ein.

4. Es bleibt nur noch, die Konstantwerte und globalen Variablen zu deklarieren sowie den Code für die Eigenschaften und Methoden einzugeben.

Wenn Sie über VB6 verfügen, können Sie das Projekt für die Beispiellösung öffnen, betrachten und damit experimentieren.



Die Visual Basic-Projektdateien für die in diesem Kapitel beschriebenen Beispiel-Lösung befinden sich in der ZIP-Datei *Bsp30\_02.zip* auf der CD-ROM im Ordner *\Beispiele\Kap30*.

## Das Erweiterungspaket-Manifest digital signieren

In diesem Kapitel wurde gezeigt, wie auf dem Entwicklungsrechner die Erweiterungspaket-Sicherheit ausgeschaltet wird. Sie soll jedoch auf Produktionssystemen aktiv gelassen werden, um die ungewollte Installation von Viren und weiterer Malware zu verhindern.

Um eine Smart Document-Lösung bei den Anwendern zu installieren, muss zumindest die Manifest-Datei des Erweiterungspakets digital signiert werden. Ist dies nicht der Fall, wird die Lösung gar nicht erst installiert.

In diesem Buch werden allgemeine Sicherheitsthemen nicht diskutiert. Wir zeigen lediglich auf, wo Sie ein digitales Zertifikat bekommen können und wie es angewendet wird.

Digitale Zertifikate gibt es für verschiedene Zwecke; um eine Manifest-Datei zu signieren, muss es für »code-signing« zugelassen sein. Wird die Smart Document-Lösung zum Verkauf angeboten, soll das Zertifikat von einem »Certification Authority« ausgestellt werden (=kostenpflichtig). Eine Liste von Microsoft vertrauten Zertifizierungsstellen finden Sie im Internet unter <http://msdn.microsoft.com/library/?url=/library/en-us/dnsecure/html/rootcertprog.asp>.

Für Smart Document-Lösungen innerhalb der eigenen Firma oder Organisation genügt ein Zertifikat einer internen »Certification Authority«. Windows Server 2003, beispielsweise, stellt ein »Certification Authority« zur Verfügung, das unter anderem Zertifikate für das Signieren von Code generieren kann, die in der eigenen Domäne gültig sind.

Um eine Lösung auf dem eigenen Rechner zu betreiben, kann sie mit dem *SelfCert.exe*-Werkzeug, das im Lieferumfang von Office enthalten ist, signiert werden (*SelfCert.exe* wurde in Kapitel 1 vorgestellt).

Nach der Installation des Zertifikats muss die Manifest-Datei damit signiert werden. Das Microsoft Office 2003 Smart Document SDK stellt das Werkzeug *XMLSign.exe* zur Verfügung. Es muss im Eingabeaufforderungsfenster ausgeführt werden. Um eine Manifest-Datei namens *Bsp30\_02M.xml* mit einem Zertifikat mit der Bezeichnung *Handbuch* zu signieren, gehen Sie wie folgt vor:

- Öffnen Sie ein Eingabeaufforderungsfenster über *Start/Alle Programme/Zubehör*.
- Navigieren Sie zum Ordner, worin sich das Werkzeug befindet.
- Geben Sie die folgende Befehlsfolge ein und drücken Sie danach die -Taste:

```
xmlsign -cn Handbuch c:\Beispiele\Kap30\Bsp30_02M.xml
```

Das Werkzeug fügt die digitale Signatur (einen Block XML-Text) am Ende der Manifest-Datei ein. Jetzt muss es möglich sein, das Erweiterungspaket zu installieren, ohne dass Sicherheitsmeldungen eingeblendet werden.

**HINWEIS** Mehr darüber, wie Sie mit Windows Server 2003 ein »Certification Authority« errichten, finden Sie unter

[http://www.microsoft.com/technet/security/prodtech/windowsserver2003/build\\_ent\\_root\\_ca.msp](http://www.microsoft.com/technet/security/prodtech/windowsserver2003/build_ent_root_ca.msp)  
sowie <http://www.microsoft.com/technet/prodtechnol/windowsserver2003/de/library/ServerHelp/2746cc74-5401-443b-898f-5dc53b1cbcb0.msp>.

## Zusammenfassung

Dieses Kapitel stellt die Smart Document-Technologie vor und beschreibt Folgendes:

- Wie der Aufgabenbereich *Dokumentaktionen* dem Anwender kontextbezogene Hilfe und Steuerelemente zur Verfügung stellt (Seite 944 ff.)
- Die Entwicklung einer Smart Document-Lösung mit MOSTL (Seite 945 ff.)
- Das Erstellen einer GUID (Seite 965 ff.)
- Die Realisierung einer Smart Document-Lösung mit einer in VB6 erstellten COM-DLL (Seite 966 ff.)



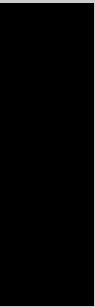


# Teil G

## Anhang

### In diesem Teil:

Anhang A	Namenskonventionen für Word-VBA	999
Anhang B	Allgemeines zum Thema Feldfunktionen	1003
Anhang C	Bezeichnungen von Symbolleisten	1011
Anhang D	Startoptionen von Word	1021
Anhang E	Bei Problemen – Word zurücksetzen	1025
Anhang F	Nützliche Links ins Internet	1029
Anhang G	Inhalt der CD-ROM	1033
Anhang H	Über die Autoren	1039



## Anhang A

# Namenskonventionen für Word-VBA

### In diesem Kapitel:

Variablen	1000
Benutzerdefiniertes Dialogfeld	1000
Entwicklungsumgebung	1001
Word-Objekte	1001

Müssen Programmzeilen von anderen Entwicklern mühsam analysiert oder eigene Funktionen nach einigen Monaten überarbeitet werden, dann ist es angenehm, wenn die entsprechenden Programmsequenzen dokumentiert sind.

Bei der Dokumentation von Programmcode sind zwei Aspekte zu beachten. Zum einen sollte die eigentliche Funktionalität innerhalb der einzelnen Zeilen kurz beschrieben werden. Zum anderen ist ein »guter« Code teilweise selbsterklärend. Dazu gehört unter anderem eine konsequente Benennung von Variablen.

Die nachstehende Zusammenstellung ist eine Empfehlung und beinhaltet die wichtigsten Datentypen, Objekte und andere Elemente. Sie baut auf den allgemeinen Empfehlungen von Microsoft auf.

## Variablen

Datentyp	Präfix	Deklaration	Beispiel
Boolean	b	Dim bFlag As Boolean	bFlag = True
Byte	byt	Dim bytWert As Byte	bytWert = Chr\$(0)
Currency	cur	Dim curEuroKurs As Currency	curEuroKurs = 1.5125
Date	date	Dim dateGeburtstag As Date	dateGeburtstag = 31.12.2005
Double	dbl	Dim dblBasis As Double	dblBasis = 100.123
Integer	int	Dim intZähler As Integer	intZähler = 5
Long	lng	Dim lngFarbwert As Long	lngFarbwert = RGB(255,0,0)
Object	obj	Dim objXls As Object	Set objXls = CreateObject("Excel.Application")
Single	sng	Dim sngRabatt As Single	sngRabatt = 0.85
String	str	Dim strOrt As String	strOrt = "Zürich"
String (fix)	stf	Dim stfUmlaute As String * 3	stfUmlaute = "äöü"
Variant	var	Dim varArgument As Variant	

## Benutzerdefiniertes Dialogfeld

Steuerelement	Präfix	Deklaration	Beispiel
Auswahlfeld	cbo		cboAnrede
Beschriftung	lbl		lblVorname
Bild	img		imgPassfoto
Dialogfeld	frm		frmInfo

Steuerelement	Präfix	Deklaration	Beispiel
Drehfeld	spn		spnAnzahl
Kontrollkästchen	chk		chkKopieErstellen
Listefeld	lst		lstVerteiler
Option	opt		optDruckenAlleSeiten
Rahmen	fra		fraOptionenAnsicht
Schaltfläche	cmd		cmdAbbrechen
Textfeld	txt		txtVorname

## Entwicklungsumgebung

Element	Präfix	Deklaration	Beispiel
Modul	vbm		vbmBrief
Klasse	cls		clsPrinter
Prozedur	sub	Private Sub subKopieDrucken( _ ByVal intAnzahl As Integer) _	subKopieDrucken
Funktion	fkt	Private Function fktSumme( _ ByVal dblSummand1 As Double, _ ByVal dblSummand2 As Double) _ As Double	fktSumme

## Word-Objekte

Objekt	Präfix	Deklaration	Beispiel
Absatz	para	Dim para As Word.Paragraph	Set para = doc.Paragraphs(1)
Add-In	addin	Dim addin As Word.addin	Set addin = app.AddIns(1)
Applikation	app	Dim app As Word.Application	
Bereich	rng	Dim rnd As Word.Range	Set rng = doc.Range
Dialogfeld	dlg	Dim dlg As Word.Dialog	Set dlg = app.Dialogs(750)
Dokument	doc	Dim doc As Word.Document	Set doc = Documents.Add
Dokumenteigen-schaft	prop	Dim prop As Office.DocumentProperty	Set prop = doc.BuiltInDocument-Properties(1)

Objekt	Präfix	Deklaration	Beispiel
Dokumentvariable	vrbl	Dim vrbl As Word.Variable	set vrbl = doc.Variables(1)
Dokumentvorlage	tmpl	Dim tmpl As Word.Template	Set tmpl = doc.AttachedTemplate
Feld	fld	Dim fld As Word.Field	Set fld = doc.Fields(1)
Formularfeld	ffld	Dim ffld As Word.FormField	set ffld = doc.FormFields(1)
Fußzeile	ftr	Dim ftr As Word.HeaderFooter	Set ftr = doc.Sections(1).Headers(1)
InlineShape	ils	Dim ils As Word.InlineShape	Set ils = doc.InlineShapes(1)
Kopfzeile	hdr	Dim hdr As Word.HeaderFooter	Set hdr = doc.Sections(1).Headers(1)
Shape	shp	Dim shp As Word.Shape	Set shp = doc.Shapes(1)
Spalte	col	Dim col As Word.Column	Set col = tbl.Columns(1)
Symboleleisten-listenfeld	ccbx	Dim ccbx As Office.CommandBarComboBox	Set ccbx = cbr.Controls(25)
Symbolleiste	cbr	Dim cbr As Office.CommandBar	Set cbr = app.CommandBars("File")
Symboleisten-schaltfläche	cbtn	Dim cbtn As Office.CommandBarButton	Set cbtn = cbr.Controls(1)
Tabelle	tbl	Dim tbl As Word.Table	Set tbl = doc.Tables(1)
Textmarke	bkm	Dim bkm As Word.Bookmark	Set bkm = doc.Bookmarks(1)
Zeile	row	Dim row As Word.Row	Set row = tbl.Rows(1)
Zelle	cel	Dim cel As Word.Cell	set cel = tbl.Cell(1,1)

## Anhang B

# Allgemeines zum Thema Feldfunktionen

Detaillierte Informationen zu den einzelnen Feldfunktionen finden Sie in den Hilfedateien von Word sowie in Kapitel 25 des Buches »Microsoft Word Version 2002 – Das Handbuch« von Microsoft Press. Hier werden nur einige Grundlagen für Word-Neulinge erläutert.



Feldfunktionen können über die Menüfolge *Einfügen/Feld* oder direkt ins Dokument eingefügt werden. Bei eingblendetem Feldcode ist erkennbar, dass sich die Feldfunktion innerhalb eines geschweiften Klammerspaares befindet. Hierbei handelt es sich um Sonderzeichen, die Sie über die Tastatur nur mit der Tastenkombination **[Strg] + [F9]** (Tabelle B.1) einfügen können.

Um verschachtelte Feldfunktionen zu erstellen, muss der Feldcode eingblendet werden.

**Tabelle B.1** Die wichtigsten Tastaturkombinationen für die Arbeit mit Feldfunktionen

Kürzel	Wirkung	VBA-Äquivalent
<b>[F9]</b>	Markierte Feldcodes aktualisieren	<code>.Fields.Update</code>
<b>[Strg] + [F9]</b>	Feldklammern einfügen { }	<code>.Fields.Add</code>
<b>[Alt] + [F9]</b>	Alle Feldcodes ein- oder ausblenden	<code>.View.ShowFieldCodes</code>
<b>[⇧] + [F9]</b>	Feldcode der markierten Feldfunktion einblenden	<code>.Field.ShowCodes</code>
<b>[⇧] + [Strg] + [F9]</b>	Markierte Feldfunktionen in Text umwandeln	<code>.Fields.Unlink</code>
<b>[Alt] + [⇧] + [F9]</b>	<i>GOTOBUTTON</i> oder <i>MACROBUTTON</i> ausführen	
<b>[F11]</b>	Springt zur nächsten Feldfunktion im Dokument	<code>.NextField</code>
<b>[⇧] + [F11]</b>	Springt zur vorherigen Feldfunktion im Dokument	<code>.PreviousField</code>
<b>[Strg] + [F11]</b>	Markierte Feldfunktionen für die Aktualisierung sperren	<code>.Fields.Locked = True</code>
<b>[⇧] + [Strg] + [F11]</b>	Sperrung der markierten Feldfunktionen aufheben	<code>.Fields.Locked = False</code>

Oft werden in einer Feldfunktion Trennzeichen gebraucht, beispielsweise für die Formatierung von Zahlen und Datum oder für eine Liste von Parametern. Bitte beachten Sie, dass Sie genau die gleichen Trennzeichen verwenden müssen, wie in den Windows-Ländereinstellungen festgelegt. Ist dort als Listentrennzeichen ein Strichpunkt eingetragen, verwenden Sie ihn, auch wenn das Beispiel in der Word-Hilfe ein Komma enthält.

Wie aus Tabelle B.2 ersichtlich, kennzeichnet Word Formatschalter mit einem umgekehrten Schrägstrich. Das bedeutet, dass Sie für Dateipfadnamen die umgekehrten Schrägstriche verdoppeln müssen:

```
{ IncludeText "NetzwerkServer\\\\"Daten\\Word.doc" }
```

**Tabelle B.2** Allgemeine Formatschalter

Schalter	Wirkung	Beispiel
<b>\#</b>	Legt die numerische Abbildung eines Feldfunktionsergebnisses fest.	<code>{ = SUM(A1;A2) \# "0,00" }</code> 10,00



Tabelle B.2 Allgemeine Formatschalter (Fortsetzung)

Schalter	Wirkung	Beispiel
\@	Formatiert das Ergebnis einer Datums-Feldfunktion. d = Tag M = Monat y = Jahr	{ DATE \@ "d. MMMM yyyy" } 3. Februar 2002 { CREATEDATE \@ "dd-MMM-yyyy" } 03-Feb-2002
\!	Verhindert die Aktualisierung einer verschachtelten Feldfunktion.	siehe Kapitel *** (Ref Seitennummer mit IncludeText)
\*	Legt das allgemeine Textformat fest.	{ REF Textmarke \* Upper } DER INHALT

Seit der Version 2000 verwendet Word die englischen Namen für Feldfunktionen und deren Schalter. In Tabelle B.3 finden Sie alle Feldfunktionen aufgelistet: links nach dem älteren deutschen Ausdruck sortiert; rechts nach dem englischen.

Tabelle B.3 Gegenüberstellung der deutschen und englischen Feldfunktionsnamen

WdFieldType	Englischer Ausdruck	Alter deutscher Ausdruck
wdFieldExpression	=(Formula)	=(Ausdruck)
wdFieldAddressBlock	AddressBlock	(neu in Word 2002)
wdFieldAdvance	Advance	Versetzen
wdFieldAsk	Ask	Frage
wdFieldAuthor	Author	Autor
wdFieldAutoNum	AutoNum	AutoNr
wdFieldAutoNumLegal	AutoNumLgl	AutoNrDez
wdFieldAutoNumOutline	AutoNumOut	AutoNrGli
wdFieldAutoText	AutoText	AutoText
wdFieldAutoTextList	AutoTextList	AutoTextListe
wdFieldComments	Comments	Kommentar
wdFieldCompare	Compare	Vergleich
wdFieldCreateDate	CreateDate	ErstellDat
wdFieldDatabase	Database	Datenbank
wdFieldDate	Date	AktualDat
wdFieldDocProperty	DocProperty	DokEigenschaft
wdFieldDocVariable	DocVariable	DokVariable
wdFieldEditTime	EditTime	(neu in Word 2002)
wdFieldFormula	EQ	Formel
wdFieldFileName	FileName	Dateiname

**Tabelle B.3** Gegenüberstellung der deutschen und englischen Feldfunktionsnamen *(Fortsetzung)*

<b>WdFieldType</b>	<b>Englischer Ausdruck</b>	<b>Alter deutscher Ausdruck</b>
wdFieldFileSize	FileSize	Dateigrösse
wdFieldFillin	Fillin	Eingeben
wdFieldGoToButton	GoToButton	Gehezu
wdFieldGreetingLine	GreetingLine	(neu in Word 2002)
wdFieldHyperlink	Hyperlink	Hyperlink
wdFieldIf	If	Wenn
wdFieldIncludePicture	IncludePicture	EinfügenGrafik
wdFieldIncludeText	IncludeText	EinfügenText
wdFieldIndex	Index	Index
wdFieldInfo	Info	Info
wdFieldKeyWord	Keywords	Stichwörter
wdFieldLastSavedBy	LastSavedBy	GespeichertVon
wdFieldLink	Link	Verknüpfung
wdFieldListNum	ListNum	ListenNr
wdFieldMacroButton	Macrobutton	MakroSchaltfläche
wdFieldMergeField	Mergefield	Seriendruckfeld
wdFieldMergeRec	MergeRec	Datensatz
wdFieldMergeSeq	MergeSeq	SeriendruckSeq
wdFieldNext	Next	Nächster
wdFieldNextIf	NextIf	Nwenn
wdFieldFootnoteRef	NoteRef	FussEndnoteRef
wdFieldNumChars	NumChars	AnzZeichen
wdFieldNumPages	NumPages	AnzSeiten
wdFieldNumWords	NumWords	AnzWörter
wdFieldPage	Page	Seite
wdFieldPageRef	PageRef	SeitenRef
wdFieldPrint	Print	Druck
wdFieldPrintDate	PrintDate	DruckDat
wdFieldQuote	Quote	Angeben
wdFieldRefDoc	RD	RD
wdFieldRef	Ref	Ref
wdFieldRevisionNum	RevNum	Überarbeitungsnummer

Tabelle B.3 Gegenüberstellung der deutschen und englischen Feldfunktionsnamen (Fortsetzung)

WdFieldType	Englischer Ausdruck	Alter deutscher Ausdruck
wdFieldSaveDate	SaveDate	SpeicherDat
wdFieldSection	Section	Abschnitt
wdFieldSequence	Seq	Seq
wdFieldSet	Set	Bestimmen
wdFieldSkipIf	SkipIf	Überspringen
wdFieldStyleRef	StyleRef	FVRef
wdFieldSubject	Subject	Thema
wdFieldSymbol	Symbol	SondZeichen
wdFieldTOCEntry	TC	Inhalt
wdFieldTemplate	Template	DokVorlage
wdFieldTime	Time	Zeit
wdFieldTitle	Title	Titel
wdFieldTOC	TOC	Verzeichnis
wdFieldUserAddress	UserAddress	BenutzerAdr
wdFieldUserInitials	UserInitials	Benutzerinitialen
wdFieldUserName	UserName	Benutzername
wdFieldIndexEntry	XE	XE
<b>Veraltete Feldtypen, die aus Gründen der Rückwärtskompatibilität noch aufgeführt sind</b>		
wdFieldData	Data	
wdFieldDDE	DDE	Diese zwei Funktionen sorgten für die Anzeige von Daten aus einer anderen Anwendung oder Dokument über eine DDE-Verknüpfung. OLE hat diese Technologie ersetzt; solche Verbindungen sollen über <i>IncludeText</i> und <i>Link</i> -Feldfunktionen vorgenommen werden.
wdFieldDDEAuto	DDEAuto	
wdFieldImport	Import	Wurde durch die Feldfunktion <i>IncludePicture</i> ersetzt.
wdFieldInclude	Include	Trägt jetzt den Namen <i>IncludeText</i> .
wdFieldGlossary	Glossary	Wurde durch die Feldfunktion <i>AutoText</i> ersetzt, als die Funktionalität in der Benutzerumgebung umbenannt wurde (Word 6.0).

**Tabelle B.3** Gegenüberstellung der deutschen und englischen Feldfunktionsnamen (Fortsetzung)

WdFieldType	Englischer Ausdruck	Alter deutscher Ausdruck
wdFieldSubscriber		Diese Feldfunktion kommt nur in Word für Macintosh vor. Sie ist ähnlich der <i>IncludePicture</i> -Feldfunktion aus Word für Windows.
<b>Feldfunktionen für die interne Verwaltung (Typ nur lesbar)</b>		
wdFieldBidiOutline	Legt die Gliederungsrichtung auf rechts nach links	wdFieldBidiOutline
wdFieldEmbed	Verwaltet ein eingebettetes Objekt wie ein Excel-Tabellenblatt	wdFieldEmbed
wdFieldHTMLActiveX	Ein Steuerelement aus der Webtools-Symbolleiste	wdFieldHTMLActiveX
wdFieldOCX	Ein ActiveX-Steuerelement, das über die Steuerelement-Toolbox eingefügt wurde.	wdFieldOCX
wdFieldFormCheckbox	Ein Formular-Kontrollkästchen	wdFieldFormCheckbox
wdFieldFormDropdown	Ein Formular-Dropdownfeld	wdFieldFormDropdown
wdFieldFormTextInput	Ein Formular-Textfeld	wdFieldFormTextInput
wdFieldPrivate	Wo Word Informationen speichert für Dokumente, die aus einem anderen Format konvertiert wurden, um das Dokument wieder in das ursprüngliche Format zurückspeichern zu können, möglichst ohne Informationsverluste	wdFieldPrivate
wdFieldShape	Eine AutoForm, die »mit Text in Zeile« formatiert ist	wdFieldShape

Noch kritischer ist die Liste der Formatschalter in Tabelle B.4, weil Sie nur die deutschen Ausdrücke in den Word-Hilfdateien finden werden. Wenn Sie versuchen, diese einzusetzen, liefern die Feldfunktionen Fehlermeldungen anstatt Ergebnisse.

**Tabelle B.4** Deutsche und englische Feldschalternamen

Deutscher Name	Englischer Name
alphabetisch	alphabetic
arabisch	Arabic
Formatverbinden	MergeFormat
Großbuchstaben	Upper
Grundtext	CardText
hex	Hex

**Tabelle B.4** Deutsche und englische Feldschalternamen (Fortsetzung)

Deutscher Name	Englischer Name
Initial	Caps
Kleinbuchstaben	Lower
OrdnungsZahl	Ordinal
OrdText	OrdText
römisch	roman
SatzanfangGroß	FirstCap
Währungstext	DollarText
Zeichenformat	CharFormat

**Tabelle B.5** Englische und deutsche Feldschalternamen

Englischer Name	Deutscher Name
alphabetic	alphabetisch
Arabic	arabisch
Caps	Initial
CharFormat	Zeichenformat
DollarText	Währungstext
FirstCap	SatzanfangGroß
CardText	Grundtext
hex	Hex
Lower	Kleinbuchstaben
MergeFormat	Formatverbinden
Ordinal	OrdnungsZahl
OrdText	OrdText
roman	römisch
Upper	Großbuchstaben



## Anhang C

# Bezeichnungen von Symbolleisten

Wie in Kapitel 15 beschrieben, dienen englische Bezeichnungen als Indexwerte bei den Symbolleisten (CommandBars). In Tabelle C.1 sowie in Tabelle C.2 finden Sie diejenigen der standardmäßigen Vorlage *Normal.dot* alphabetisch zusammen mit dem numerischen Indexwert aufgelistet.

**Tabelle C.1**    Sortiert nach den deutschen Begriffen

Deutsche Bezeichnung	Englische Bezeichnung	Indexwert
3D-Einstellungen	3-D Settings	24
Absatzschriftart	Font Paragraph	80
ActiveX-Steuerelement	ActiveX Control	101
Address Block Popup	Address Block Popup	110
Aktualisieren	Refresh	42
Änderungen nachverfolgen	Track Changes	88
Anmerkungsstifte	Annotation Pens	128
Anmerkungsstifte	Annotation Pens	130
Aufgabenbereich	Task Pane	51
Aufzeichnung beenden	Stop Recording	10
Ausrichten oder verteilen	Align or Distribute	131
AutoFormen	AutoShapes	116
AutoSignaturmenü	AutoSignature Popup	91
AutoText	AutoText	21
AutoText-Feld	Field AutoText	92
AutoZusammenfassen	AutoSummarize	31
Blockpfeile	Block Arrows	119
Canvas Popup	Canvas Popup	99
Chinesische Schriftumwandlung	Chinese Translation	109
Datenbank	Database	4
Diagramm	Diagram	29
Diagramm	Diagram	106
Dokumentlayout	Document Layout	46
Dokumentstruktur	Document Map	93
Drehen oder kippen	Rotate or Flip	132
Drehungsmodus	Rotate Mode	103
E-Mail	E-mail	44
E-Mail senden	Send Mail	19
Endnoten	Endnotes	55
Entwurfsmodus beenden	Exit Design Mode	32
Erweiterte Formatierung	Extended Formatting	20



**Tabelle C.1** Sortiert nach den deutschen Begriffen (Fortsetzung)

Deutsche Bezeichnung	Englische Bezeichnung	Indexwert
Feldanzeige für Listenzahlen	Field Display List Numbers	58
Felder	Fields	56
Felder anzeigen	Display Fields	57
Flussdiagramm	Flowchart	118
Form einfügen	Insert Shape	125
Format	Formatting	2
Format Inspector Popup in Compare Mode	Format Inspector Popup in Compare Mode	82
Format Inspector Popup in Normal Mode	Format Inspector Popup in Normal Mode	81
Formen	Shapes	94
Formular	Forms	6
Formularfelder	Form Fields	59
Frameeigenschaften	Frame Properties	89
Frames	Frames	43
Freihandanmerkungen	Ink Annotations	49
Freihandkommentar	Ink Comment	50
Freihandkommentar	Ink Comment	115
Freihandzeichnung und -schrift	Ink Drawing and Writing	48
Füllfarbe	Fill Color	124
Funktionstastenanzeige	Function Key Display	37
Fußnoten	Footnotes	60
Ganze Tabelle	Whole Table	75
Ganzer Bildschirm	Full Screen	7
Gliederung	Outlining	15
Grafik	Picture	26
Grafik bearbeiten	Edit Picture	8
Grammatik	Grammar	84
Grammatik (2)	Grammar (2)	85
Greeting Line Popup	Greeting Line Popup	111
Hyperlink-Kontextmenü	Hyperlink Context Menu	90
Initial	Drop Caps	54
Inline Zeichnungsbereich	Inline Canvas	67
Inline-ActiveX-Steuerelement	Inline ActiveX Control	112
Inlinebild	Inline Picture	66
Japanische Grußformeln	Japanese Greetings	40

**Tabelle C.1**    Sortiert nach den deutschen Begriffen *(Fortsetzung)*

Deutsche Bezeichnung	Englische Bezeichnung	Indexwert
Knoten krümmen	Curve Node	96
Kommentar	Comment	104
Konsistenz formatieren	Format consistency	86
Kopf- und Fußzeile	Header and Footer	14
Kopfzeilen	Headings	62
Krümmen	Curve	95
Kurvenabschnitt	Curve Segment	97
Legenden	Callouts	117
Lesemoduslayout	Reading Layout	45
Linien	Lines	121
Linienfarbe	Line Color	126
Listen	Lists	65
Macros book	Macros book	53
Menüleiste	Menu Bar	41
Microsoft	Microsoft	13
Nachricht lesen	Read Mail	18
Nebeneinander vergleichen	Compare Side by Side	47
OLE-Objekt	OLE Object	100
Onlinebesprechung	Online Meeting	143
Organigramm	Organization Chart	28
Organization Chart Popup	Organization Chart Popup	105
Outlook-E-Mail lesen	Outlook Read Mail	35
Outlook-E-Mail senden	Outlook Send Mail	36
Popupmenü: Horizontale Linie	Horizontal Line Popup	68
Positionsrahmen	Frames	61
Präzisionsausrichtung	Nudge	134
Rahmen	Borders	135
Rechtschreibung	Spelling	83
Reihenfolge	Order	133
Schatteneinstellungen	Shadow Settings	25
Schattierungsfarbe	Shading Color	137
Schriftfarbe	Font Color	136
Seitenansicht	Print Preview	16
Seriendruck	Mail Merge	11

**Tabelle C.1** Sortiert nach den deutschen Begriffen (Fortsetzung)

Deutsche Bezeichnung	Englische Bezeichnung	Indexwert
Skriptanchor-Popupmenü	Script Anchor Popup	64
Standard	Standard	1
Standardformen	Basic Shapes	122
Statusleiste für Rechtschreibprüfung im Hintergrund	Background Proofing Status Bar	87
Sterne & Banner	Stars & Banners	120
Steuerelement-Toolbox	Control Toolbox	33
System	System	142
Tabellen	Tables	69
Tabellen und Rahmen	Tables and Borders	3
Tabellengrafiken	Table Pictures	73
Tabellenlisten	Table Lists	72
Tabellentext	Table Text	74
Tabellenüberschriften	Table Headings	71
Tabellenzellen	Table Cells	70
Text	Text	77
Textfeld	Text Box	34
Textfluss	Text Wrapping	139
Überarbeiten	Reviewing	30
Überarbeitungsanzeige	Track Changes Indicator	108
Umschlag	Envelope	141
Untermenü Schrift	Font Popup	79
Unverankerte Grafik	Floating Picture	98
Verbindung	Connector	107
Verbindungen	Connectors	123
Verknüpfte Tabelle	Linked Table	76
Verknüpfte Überschriften	Linked Headings	63
Verknüpfter Text	Linked Text	78
Visual Basic	Visual Basic	9
Web	Web	22
Webtools	Web Tools	38
Word für Windows 2.0	Word for Windows 2.0	17
WordArt	WordArt	23
WordArt-Kontextmenü	WordArt Context Menu	102
Wörter zählen	Word Count	39

**Tabelle C.1** Sortiert nach den deutschen Begriffen (Fortsetzung)

Deutsche Bezeichnung	Englische Bezeichnung	Indexwert
XML Strukturknoten-Popup	XML Structure Node Popup	113
XML-Fehleroptionen	XML Error Options	114
Zeichnen	Drawing	5
Zeichnungs- und Schreibstifte	Drawing and Writing Pens	127
Zeichnungs- und Schreibstifte	Drawing and Writing Pens	129
Zeichnungsbereich	Drawing Canvas	27
Zellausrichtung	Cell Alignment	138
Zentraldokument	Master Document	12
Zwischenablage	Clipboard	140

**Tabelle C.2** Sortiert nach den englischen Begriffen

Englische Bezeichnung	Deutsche Bezeichnung	Indexwert
3-D Settings	3D-Einstellungen	24
ActiveX Control	ActiveX-Steuerelement	101
Address Block Popup	Address Block Popup	110
Align or Distribute	Ausrichten oder verteilen	131
Annotation Pens	Anmerkungsstifte	128
Annotation Pens	Anmerkungsstifte	130
AutoShapes	AutoFormen	116
AutoSignature Popup	AutoSignaturmenü	91
AutoSummarize	AutoZusammenfassen	31
AutoText	AutoText	21
Background Proofing Status Bar	Statusleiste für Rechtschreibprüfung im Hintergrund	87
Basic Shapes	Standardformen	122
Block Arrows	Blockpfeile	119
Borders	Rahmen	135
Callouts	Legenden	117
Canvas Popup	Canvas Popup	99
Cell Alignment	Zellausrichtung	138
Chinese Translation	Chinesische Schriftumwandlung	109
Clipboard	Zwischenablage	140
Comment	Kommentar	104

**Tabelle C.2** Sortiert nach den englischen Begriffen (Fortsetzung)

Englische Bezeichnung	Deutsche Bezeichnung	Indexwert
Compare Side by Side	Nebeneinander vergleichen	47
Connector	Verbindung	107
Connectors	Verbindungen	123
Control Toolbox	Steuerelement-Toolbox	33
Curve	Krümmen	95
Curve Node	Knoten krümmen	96
Curve Segment	Kurvenabschnitt	97
Database	Datenbank	4
Diagram	Diagramm	29
Diagram	Diagramm	106
Display Fields	Felder anzeigen	57
Document Layout	Dokumentlayout	46
Document Map	Dokumentstruktur	93
Drawing	Zeichnen	5
Drawing and Writing Pens	Zeichnungs- und Schreibstifte	127
Drawing and Writing Pens	Zeichnungs- und Schreibstifte	129
Drawing Canvas	Zeichnungsbereich	27
Drop Caps	Initial	54
Edit Picture	Grafik bearbeiten	8
E-mail	E-Mail	44
Endnotes	Endnoten	55
Envelope	Umschlag	141
Exit Design Mode	Entwurfsmodus beenden	32
Extended Formatting	Erweiterte Formatierung	20
Field AutoText	AutoText-Feld	92
Field Display List Numbers	Feldanzeige für Listenzahlen	58
Fields	Felder	56
Fill Color	Füllfarbe	124
Floating Picture	Unverankerte Grafik	98
Flowchart	Flussdiagramm	118
Font Color	Schriftfarbe	136
Font Paragraph	Absatzschriftart	80

**Tabelle C.2** Sortiert nach den englischen Begriffen (Fortsetzung)

Englische Bezeichnung	Deutsche Bezeichnung	Indexwert
Font Popup	Untermenü Schrift	79
Footnotes	Fußnoten	60
Form Fields	Formularfelder	59
Format consistency	Konsistenz formatieren	86
Format Inspector Popup in Compare Mode	Format Inspector Popup in Compare Mode	82
Format Inspector Popup in Normal Mode	Format Inspector Popup in Normal Mode	81
Formatting	Format	2
Forms	Formular	6
Frame Properties	Frameeigenschaften	89
Frames	Frames	43
Frames	Positionsrahmen	61
Full Screen	Ganzer Bildschirm	7
Function Key Display	Funktionstastenanzeige	37
Grammar	Grammatik	84
Grammar (2)	Grammatik (2)	85
Greeting Line Popup	Greeting Line Popup	111
Header and Footer	Kopf- und Fußzeile	14
Headings	Kopfzeilen	62
Horizontal Line Popup	Popupmenü: Horizontale Linie	68
Hyperlink Context Menu	Hyperlink-Kontextmenü	90
Ink Annotations	Freihandanmerkungen	49
Ink Comment	Freihandkommentar	50
Ink Comment	Freihandkommentar	115
Ink Drawing and Writing	Freihandzeichnung und -schrift	48
Inline ActiveX Control	Inline-ActiveX-Steuerelement	112
Inline Canvas	Inline Zeichnungsbereich	67
Inline Picture	Inlinebild	66
Insert Shape	Form einfügen	125
Japanese Greetings	Japanische Grußformeln	40
Line Color	Linienfarbe	126
Lines	Linien	121
Linked Headings	Verknüpfte Überschriften	63

**Tabelle C.2** Sortiert nach den englischen Begriffen (Fortsetzung)

Englische Bezeichnung	Deutsche Bezeichnung	Indexwert
Linked Table	Verknüpfte Tabelle	76
Linked Text	Verknüpfter Text	78
Lists	Listen	65
Macros book	Macros book	53
Mail Merge	Seriendruck	11
Master Document	Zentraldokument	12
Menu Bar	Menüleiste	41
Microsoft	Microsoft	13
Nudge	Präzisionsausrichtung	134
OLE Object	OLE-Objekt	100
Online Meeting	Onlinebesprechung	143
Order	Reihenfolge	133
Organization Chart	Organigramm	28
Organization Chart Popup	Organization Chart Popup	105
Outlining	Gliederung	15
Outlook Read Mail	Outlook-E-Mail lesen	35
Outlook Send Mail	Outlook-E-Mail senden	36
Picture	Grafik	26
Print Preview	Seitenansicht	16
Read Mail	Nachricht lesen	18
Reading Layout	Lesemoduslayout	45
Refresh	Aktualisieren	42
Reviewing	Überarbeiten	30
Rotate Mode	Drehungsmodus	103
Rotate or Flip	Drehen oder kippen	132
Script Anchor Popup	Skriptanchor-Popupmenü	64
Send Mail	E-Mail senden	19
Shading Color	Schattierungsfarbe	137
Shadow Settings	Schatteneinstellungen	25
Shapes	Formen	94
Spelling	Rechtschreibung	83
Standard	Standard	1

**Tabelle C.2**    Sortiert nach den englischen Begriffen *(Fortsetzung)*

Englische Bezeichnung	Deutsche Bezeichnung	Indexwert
Stars & Banners	Sterne & Banner	120
Stop Recording	Aufzeichnung beenden	10
System	System	142
Table Cells	Tabellenzellen	70
Table Headings	Tabellenüberschriften	71
Table Lists	Tabellenlisten	72
Table Pictures	Tabellengrafiken	73
Table Text	Tabellentext	74
Tables	Tabellen	69
Tables and Borders	Tabellen und Rahmen	3
Task Pane	Aufgabenbereich	51
Text	Text	77
Text Box	Textfeld	34
Text Wrapping	Textfluss	139
Track Changes	Änderungen nachverfolgen	88
Track Changes Indicator	Überarbeitungsanzeige	108
Visual Basic	Visual Basic	9
Web	Web	22
Web Tools	Webtools	38
Whole Table	Ganze Tabelle	75
Word Count	Wörter zählen	39
Word for Windows 2.0	Word für Windows 2.0	17
WordArt	WordArt	23
WordArt Context Menu	WordArt-Kontextmenü	102
XML Error Options	XML-Fehleroptionen	114
XML Structure Node Popup	XML Strukturknoten-Popup	113



## Anhang D

# Startoptionen von Word

**In diesem Kapitel:**

Die Befehlszeilenargumente von Word

1022

Zusammenfassung

1023

Word kann auf verschiedene Arten gestartet werden. Diese zusätzlichen Befehlszeilenargumente steuern das Verhalten von Word zu einem ganz bestimmten Zweck.

## Die Befehlszeilenargumente von Word

In der nachstehenden Tabelle sind alle bekannten Startoptionen von Word zusammengefasst. Gemäß den Informationen aus der Microsoft Knowledge Base sind alle möglichen Argumente in der Online-Hilfe zu Word offen gelegt.

Die Startoptionen von Word werden in erster Line für den manuellen Programmstart auf der Kommandozeile oder vom Entwickler beim Einsatz eines Shell-Aufrufs verwendet.

**Tabelle D.1** Zusammenfassung der Startoptionen von Word

Syntax	Bedeutung
winword.exe	Ein neues Word-Fenster mit einem neuen leeren Dokument wird erzeugt, wobei die bestehende Instanz von Word genutzt wird.
winword.exe /q	Startet eine neue Instanz von Word, ohne den so genannten Splash-screen anzuzeigen.
winword.exe "Dateiname1" "Dateiname2" "DateinameX"	Ein neues Word-Fenster wird für jede Datei erzeugt. Nach erfolgreichem Start werden die entsprechenden Dokumente geöffnet.
winword.exe /n	Startet eine neue Instanz von Word, ohne ein leeres Dokument zu generieren.
winword.exe /w	Startet eine neue Instanz von Word und generiert ein leeres Dokument.
winword.exe /t"Vorlage"	Startet eine neue Instanz von Word. Anschließend wird ein neues Dokument basierend auf der angegebenen Dokumentvorlage generiert, wobei weder das Makro <i>AutoNew</i> abgearbeitet noch das Ereignis <b>DocumentNew</b> eintritt.
winword.exe /m	Startet eine neue Instanz von Word. Es werden keine <i>AutoExec</i> -Makros ausgeführt.
winword.exe /m"Makroname"	Startet eine neue Instanz von Word. Anschließend wird das entsprechende Makro ausgeführt. Dieses muss sich in der <i>Normal.dot</i> befinden. Es werden keine <i>AutoExec</i> -Makros ausgeführt. Das definierte Makro kann sich jedoch auch auf einen internen Word-Befehl beziehen (beispielsweise <i>/Datei1</i> bzw. <i>/mFile1</i> ).
winword.exe /a	Startet eine neue Instanz von Word, ohne die hinterlegten Add-In zu aktivieren.
winword.exe /safe	Startet eine neue Instanz von Word im abgesicherten Modus. Dies entspricht dem Argument <i>/a</i> , wobei noch weitere Optionen beim Start nicht aktiviert werden (ab Word 2002).
winword.exe /r	Word wird nicht gestartet, sondern führt eine Neuregistrierung in der Windows-Registrierung durch.
Winword.exe /l"Add-In-Name"	Startet eine neue Instanz von Word; zusätzlich wird das entsprechende Add-In geladen und aktiviert.
Winword.exe /	Startet eine neue Instanz von Word, wenn nur ein Schrägstrich (/) oder eine unbekannte Befehlszeilenooption angegeben wird.

Grundsätzlich gelten für alle Argumente die nachstehenden Punkte:

- Lange Dateinamen mit oder ohne Pfadangaben, welche Leerzeichen enthalten, müssen in Anführungszeichen eingeschlossen werden.
- Die einzelnen Argumente werden mit einem Leerzeichen voneinander getrennt. Zwischen dem eigentlichen Argument und dessen Ergänzung wird kein Leerzeichen eingefügt.
- Die meisten Argumente können miteinander kombiniert werden. Dies gilt nicht für die beiden Schalter */m* und */t*, welche nicht zusammen verwendet werden können.

#### HINWEIS

Detaillierte Informationen zu den einzelnen Startoptionen von Word und zum Thema allgemein können der Microsoft Knowledge Base entnommen werden. Die nachstehenden Artikel beschäftigen sich explizit mit dem Thema:

- WD2002: Problembehandlung bei Word: Vergleich der Befehlszeilenoptionen *"/a"* und *"/safe"* (<http://support.microsoft.com/?kbid=813589>)
- WD: Word-Startoptionen (Befehlszeilenoptionen) und ihre Verwendung (<http://support.microsoft.com/?kbid=210565>)
- Beschreibung von abgesichertem Office-Modus für Word 2003 und Word 2002 (<http://support.microsoft.com/default.aspx?scid=kb;de;827706>)
- WD: Word für Windows-Startoptionen (<http://support.microsoft.com/?kbid=70014>)

## Zusammenfassung

In diesem Kapitel wurde aufgezeigt, welche Startoptionen von Word unterstützt werden:

- Es wurde die Syntax der einzelnen Argumente erläutert und deren eigentliche Bedeutung erklärt.
- Ebenso wurden allgemeingültige Punkte und Einschränkungen, welche für alle Argumente zutreffen, aufgezeigt.



## Anhang E

# Bei Problemen – Word zurücksetzen

Mit den Hinweisen in diesem Anhang möchten wir Word nichts Negatives anhängen. Trotzdem muss sich der Anwender dieses Programms damit abfinden, dass dieses in einen instabilen Zustand kommen kann. Dieser Umstand tritt zwar gemäß unseren Erfahrungen nur selten auf, doch dies ist ein schwacher Trost, wenn es doch einmal passiert.

Mit der aktuellen Version von Word ist ein einzelner Programmabsturz meistens unproblematisch, da ungespeicherte Dateien durch die Funktion *AutoWiederherstellen-Info* beim nächsten Start des Programms, oft ohne Datenverlust, automatisch wieder hergestellt werden. Um diese Funktion zu aktivieren, wählen Sie den Menübefehl *Extras/Optionen* an. Wechseln Sie zur Registerkarte *Speichern* und aktivieren Sie das Kontrollkästchen *AutoWiederherstellen-Info*.

Die Problematik liegt jedoch ganz anders, wenn sich Word bei jedem Programmstart gleich wieder verabschiedet oder dies regelmäßig während der Arbeit mit dem Programm geschieht. Zwei Aspekte sind meist an diesem Übel beteiligt.

### *Normal.dot* neu erzeugen



Normal.dot

Bei einer unstabilen Word-Umgebung oder wenn einzelne Dokumente regelmäßig beschädigt werden, sollte die Standardvorlage *Normal.dot* neu aufgebaut werden.

Der Grund dazu liegt im Konzept von Word. Die Standardvorlage ist während der ganzen Word-Sitzung im Schreibmodus geöffnet. Tritt bei der Bearbeitung eines Textes ein Fehler auf und Word hat sich verabschiedet, kann dies zu einer defekten *Normal.dot* führen. Dies wiederum kann Word regelmäßig zum »Abschmieren« bringen.

Wie ein neues Original der Standardvorlage *Normal.dot* erzeugt wird, wurde bereits in Kapitel 12 ausführlich beschrieben.

#### **HINWEIS**

Bevor eine neue *Normal.dot* erzeugt werden kann, muss die bestehende Datei dem Zugriff von Word entzogen werden. Dazu muss Word beendet und die bestehende *Normal.dot* umbenannt werden.

Wir Autoren empfehlen Ihnen unbedingt, die bestehende Datei nicht zu löschen, sondern mit einem neuen Namen zu versehen (beispielsweise *AlteNormal.dot*). Somit können zu einem späteren Zeitpunkt die in der *Normal.dot* abgespeicherten Daten (Symbolleisten, Formatvorlagen, Auto-Texte und Makros) von eben dieser Datei restauriert werden. Die Funktion »Organisieren« wurde bereits in Kapitel 1 vorgestellt.

### *Data*-Schlüssel in der Windows-Registrierung



regedit.exe

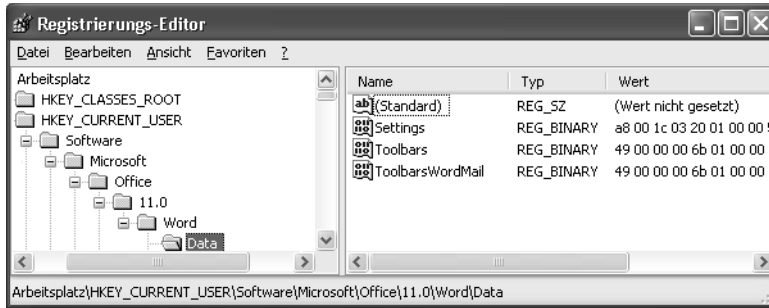
Word hinterlegt viele Einstellungen zu den gewählten Optionen und zur aktuellen Programmumgebung in der Windows-Registrierung. Diese Daten werden im Schlüssel *Data* in ein paar wenigen Einträgen mit entsprechend großen binären Datenfeldern abgespeichert.

Können diese Daten nicht sauber in der Windows-Registrierung gespeichert werden, was zum Beispiel bei einem Programmabsturz der Fall sein kann, so besteht die Möglichkeit, dass diese nicht mehr konsistent sind. Diese Inkonsistenzen können wiederum dazu führen, dass Word regelmäßig abstürzt.

Um den besagten Schlüssel innerhalb der Windows-Registrierung zu entfernen, muss Word beendet werden. Starten Sie anschließend den Registrierungseditor (*Regedit.exe*) und wechseln Sie zum Schlüssel *HKEY\_CURRENT\_USER\Software\Microsoft\Office\11.0\Word\Data*. Markieren Sie diesen Schlüssel und rufen Sie den Menübefehl *Datei/Exportieren* auf, um eine Sicherheitskopie zu

erstellen. In einem zweiten Schritt wird der Schlüssel entfernt, dazu muss der Menübefehl *Bearbeiten/Löschen* angewählt werden.

Abbildg. E.1 Data-Schlüssel in der Windows-Registrierung umbenennen



Weitere Informationen zum Zurücksetzen der Optionen und Einstellungen können dem Artikel 822005 aus der Microsoft Knowledge Base entnommen werden (<http://support.microsoft.com/default.aspx?scid=kb;en-us;822005>).

---

**HINWEIS** Wir Autoren empfehlen Ihnen, eine Sicherheitskopie dieses Schlüssels zu erstellen. Der Vorteil dabei: Konnte durch diese Aktion die bestehende Problematik nicht behoben werden, lassen sich die alten Werte jederzeit wieder aktivieren.

---





## Anhang F

# Nützliche Links ins Internet

Auf dieser Seite sind die Adressen einiger interessanter Internet-Seiten zusammengefasst, die wir Autoren weiterempfehlen können. Diese Liste hat keinen Anspruch auf Vollständigkeit.

<http://www.microsoft.com/germany/msdn/library/office/default.msp>

Die deutsche MSDN-Seite für Microsoft Office. Bis heute wurden zwar noch nicht allzu viele Artikel und Informationen, die Word direkt betreffen, übersetzt. Trotzdem sind auf dieser Seite viele interessante Artikel vorhanden. Dazu zählen auch Artikel, die sich mit .NET und Office auseinandersetzen.

[http://msdn.microsoft.com/library/en-us/odc\\_2003\\_ta/html/odc\\_ancffsol.asp](http://msdn.microsoft.com/library/en-us/odc_2003_ta/html/odc_ancffsol.asp)

Die englischen MSDN-Seiten für Microsoft Office. Auf dieser Seite werden reichlich Informationen zur Verfügung gestellt, die zur Entwicklung von Word-Lösungen, auch basierend auf .NET, dienen.

<http://homepage.swissonline.ch/cindymeister>

Die englischsprachige Homepage von Cindy Meister, Autorin dieses Buches und MVP. Eine umfassende Anzahl von Informationen und Beispielen zu Word und VBA. Einen besonderen Schwerpunkt bilden die Informationen rund um das Thema »Serienbrief« (Mailmerge).

<http://www.chf-online.de>

Die Internet-Seite von Christian Freßdorf, Co-Autor dieses Buches und MVP. Die Seite enthält viele attraktive Beispiele zu VBA. Besonders erwähnenswert sind die Beispiele mit eingebundenen API-Funktionen sowie sein VBA-HTML Konverter.

<http://mypage.bluewin.ch/reprobst/WordFAQ>

Eine wirklich umfangreiche deutschsprachige FAQ-Seite. Auf dieser Seite kann man Stunden lang verweilen und findet immer wieder neue Informationen.

<http://word.mvps.org/FAQs/index.htm>

Eine englischsprachige FAQ-Seite, die von den MVPs weltweit als gemeinsame und zentrale Plattform betrieben wird.

[http://www.wordsite.com/word\\_faqs.html](http://www.wordsite.com/word_faqs.html)

Die englischsprachige FAQ-Seite des MVPs Bill Coan. Sie enthält Beiträge mit vertieftem Einblick zu verschiedenen Themen rund um Word.

<http://www.ti5.tu-harburg.de/manual/vba5/htoc.htm>

Ein Online-Lehrgang zu VBA. Der Lehrgang basiert auf der veralteten Version VBA5 (Office 97). Die Grundlagen zum Thema haben jedoch weiterhin ihre Gültigkeit. Besonders empfehlenswert für Leser, die ihre Kenntnisse des Inhalts aus Kapitel 2 und 3 erweitern möchten.

<http://www.aboutvb.de/vba/vba.htm>

Eine deutschsprachige Internet-Seite mit qualitativ hochwertigen Beiträgen, die sich an den professionellen Entwickler richten. Neben Beiträgen zu VBA stehen hier die Artikel zum klassischen Visual Basic sowie .NET und anderen Programmierthemen im Vordergrund.

<http://architag.com/xray/>

Architag stellt Werkzeuge zum Erstellen von *xml*-, *xsl*- sowie *xsd*-Dateien zur Verfügung. Diese Produkte wurden in Kapitel 28 kurz vorgestellt. Vom Hersteller wird ein kostenloses sowie ein eher teureres, dafür gutes, Produkt angeboten.

<http://www.iol.ie/~pxe/>

»Peters XML Editor« ist ein recht nützliches Programm zum Erstellen von *xml*-Dateien. Im Gegensatz zu X-Ray von Architag können die *xml*-Dateien in drei verschiedenen Ansichten bearbeitet werden (»nur Text«, im Browser und in einer Baumstruktur). Das Programm enthält jedoch keine Möglichkeit, die Wirkung von Schemas und Transforms dynamisch zu prüfen.



---

Sie finden zu jeder aufgeführten Internet-Adresse eine entsprechende *url*-Datei auf der CD-ROM zum Buch im Ordner *\Beilagen\AnhangF*.

---



## Anhang G

# Inhalt der CD-ROM

**In diesem Kapitel:**

Die Beispieldateien der CD-ROM zum Buch	1034
Zusatz-Software: Die Dateien im Anhang	1036

# Die Beispieldateien der CD-ROM zum Buch

Alle im Buch dargestellten Beispiele finden Sie im Ordner *\Beispiele* auf der beigelegten CD-ROM. In der Tabelle G.1 sind die benötigten Informationen zusammengefasst. Sie können dort die jeweiligen Kapitel, den Speicherort und die Namen der Beispieldateien entnehmen.

**HINWEIS** Beachten Sie jeweils die speziellen Hinweise zur Handhabung der Beispiel- und Übungsdateien im jeweiligen Kapitel.

Für die meisten Beispiele ist es von Vorteil, den jeweiligen Ordner von der CD-ROM direkt auf die Festplatte Ihres PCs zu kopieren. Entfernen Sie anschließend – falls erforderlich – das Schreibschutz-Attribut aller kopierten Dateien. Markieren Sie dazu im Windows-Explorer die betroffenen Dateien, klicken Sie mit der rechten Maustaste in die Markierung, wählen Sie aus dem Kontextmenü den Befehl *Eigenschaften*, deaktivieren Sie das Kontrollkästchen *Schreibgeschützt* und klicken Sie dann auf die Schaltfläche *OK*.

**Tabelle G.1** Übersicht über die Beispieldateien auf der CD-ROM zum Buch

Kapitel	Speicherort	Name der Dateien
1	\Beispiele\Kap01	Bsp01_01.doc
2	\Beispiele\Kap02	Bsp02_01.doc
3	\Beispiele\Kap03	Bsp03_01.doc, Bsp03_02.doc, Bsp03_03.doc, Bsp03_04.doc, Bsp03_05.doc, Bsp03_06.doc, Bsp03_07.doc, DebugTest.dot
4	\Beispiele\Kap04	Bsp04_1.doc, Bsp04-1.ini
5	\Beispiele\Kap05	Bsp05_01.doc
6	\Beispiele\Kap06	Blaue Spitzen 16.bmp, Bsp01_02_Field.xls, Bsp06_01_Bookmark.doc, Bsp06_01_Documen1_Aug02.doc, Bsp06_01_Document.doc, Bsp06_01_Field.doc, Bsp06_01_Find.doc, Bsp06_01_Graf.doc, Bsp06_01_Style.doc, Bsp06_01_System.doc, Bsp06_01_Table.doc, Bsp06_01_Template.doc, Bsp06_01_Zentraldokument.doc, Bsp06_01a.doc, Bsp06_01App.doc, Bsp06_01Range.doc, Bsp06_01Range_Test.doc, Bsp06_02_Field.doc, Bsp06_02_Field.xls, Bsp06_02_Find.txt, Bsp06_02_Graf.doc, Bsp06_02_Num.doc, Bsp06_02_Style.doc, Bsp06_02_Table.doc, Bsp06_02_Template.doc, Bsp06_02_Zentraldokument.doc, Bsp06_02a.doc, Bsp06_02Range.doc, Bsp06_03_Graf.doc, Bsp06_03_Num.doc, Bsp06_03_Style.doc, Bsp06_03_Table.doc, Bsp06_03Range.doc, Bsp06_04_Find.doc, Bsp06_04_Graf.doc, Bsp06_04_Table.doc, Bsp06_04_Zentraldokument.doc, Bsp06_05_Find.doc, Bsp06_05_Graf.doc, Bsp06_05_Table.doc, Bsp06_05_Zentraldokument.doc, Bsp06_06_Find.doc, Bsp06_2_Find.doc, Bsp06_2_Find.txt, Bsp06_3_Find.doc, Bsp06_Test.doc
7	\Beispiele\Kap07	Bsp_07.dot, Bsp_07-1.dot, Bsp_07-2.dot Kap07_Beiispiel.doc
8	\Beispiele\Kap08	Bsp08_01.doc, Bsp08_02.doc, Bsp08_03.doc, MSComctl.ocx

Tabelle 6.1 Übersicht über die Beispieldateien auf der CD-ROM zum Buch (Fortsetzung)

Kapitel	Speicherort	Name der Dateien
9	\Beispiele\Kap09	Bsp_Demo.doc, Bsp09_01.doc, Bsp09_02.xls, Bsp09_03.dot, Bsp09_04.dot, Bsp09_05.dot, Bsp09_06.doc, Bsp09_07.doc, Bsp09_07.xls, Bsp09_08.dot, Bsp09_Brief.dot, VSTOBrief_Beispiel.doc
10	\Beispiele\Kap10	Bsp_Demo.ppt, Bsp_Demo.vsd, Bsp_Demo.xls, Bsp10_01.doc, Bsp10_01a.doc, Bsp10_02.dot, Bsp10_02a.dot, Bsp10_02b.doc, Bsp10_03.dot, Bsp10_04.dot, Bsp10_1_Access.doc
11	\Beispiele\Kap11	Bsp11_01.xls, Bsp11_03.doc, Bsp11_04.doc, Bsp11_05.doc, Bsp11_06.doc, Bsp11_06.dot
12	\Beispiele\Kap12	Bsp12_01.doc
14	\Beispiele\Kap14	Bsp14_01.doc, Bsp14_02.doc, Bsp14_03.doc, Bsp14_03.ini, Bsp14_03a.txt, Bsp14_03b.txt, Bsp14_04.doc, wdDialogsKonstan- ten-Word2003.doc, wdDialogsübersicht_Word2003.doc
15	\Beispiele\Kap15	Bsp15_01.doc, Bsp15_02.doc, Bsp15_03.doc, Bsp15_04.doc, Bsp15_05.doc
16	\Beispiele\Kap16	AddinBeispiel.dll, Bsp16_01.dot, Bsp16_02.dot, Bsp16_03.dot, COMAddin_Test.dot, TestOeffnen.doc
17	\Beispiele\Kap17	Bsp17_01.doc, Bsp17_02.doc, Bsp17_03.doc, Bsp17_04.doc
18	\Beispiele\Kap18	Bsp18_01.dot
19	\Beispiele\Kap19	Bsp19_01.doc, Bsp19_02.doc, Bsp19-Referenz.dot, modImport- Modul.bas, modSearchProz.bas
20	\Beispiele\Kap20	Bsp20_01.doc
21	\Beispiele\Kap21	Bsp21_Symbolleiste.doc
22	\Beispiele\Kap22	Bsp22_01.doc, Bsp22_02.doc
23	\Beispiele\Kap23	Bsp23_01.dot, Bsp23_02.dot, Bsp23_03.dot, Bsp23_04.dot, Bsp23_05.dot, Bsp23_06.dot, Bsp23_Logo.bmp, Bsp23_Wasserzeichen.bmp
24	\Beispiele\Kap24	Bsp_24_01.doc
25	\Beispiele\Kap25	Bsp25_01.doc, Bsp25_01.dot, Kap25_Beispiel.doc
26	\Beispiele\Kap26	Bsp26_01.doc
27	\Beispiele\Kap27	Bsp27_01.doc, WordArt.doc
28	\Beispiele\Kap28	Bsp28_01.htm, Bsp28_02.xml, Bsp28_02a.doc, Bsp28_02a.doc.xml, Bsp28_02a.xml, Bsp28_02b.xml, Bsp28_03.xml, Bsp28_03.xml, Bsp28_04_Beispiel.xml, Bsp28_04_Beispiel.xml, Bsp28_05.xsd, Bsp28_06.xsd, Bsp28_07.xsd, Bsp28_08.xsd, Bsp28_09.xsd, Bsp28_10.xml, Bsp28_11.xml, Bsp28_12.xml, Bsp28_13.xml, Bsp28_14.xml, Bsp28_15.xml, Bsp28_16.xml, Bsp28_17.xml, Bsp28_18.xsd, Bsp28_19.doc, Bsp28_20.xsd, Bsp28_21.xml, Bsp28_22.xml, Bsp28_23.xml, Bsp28_25.doc, Bsp28_25.txt

Tabelle G.1 Übersicht über die Beispieldateien auf der CD-ROM zum Buch (Fortsetzung)

Kapitel	Speicherort	Name der Dateien
29	\Beispiele\Kap29	Bsp29_02.doc, Bsp29_02.xml, Bsp29_02_äp.htm, Bsp29_02_dl.htm, Bsp29_02_ip.htm, Bsp29_02_mi.htm, Bsp29_02_up_v2.xml, Bsp29_02_up_v3.xml, Bsp29_02_v2.xml, Bsp29_02_v3.xml, Bsp29_03_VB.zip, Bsp29_04.doc, Bsp29_04.xsl, planet.xml
30	\Beispiele\Kap30	Bsp30_01.reg, Bsp30_01.xsd, Bsp30_01.xsl, Bsp30_01A.xml, Bsp30_01B.xml, Bsp30_01B_Test.xml, Bsp30_01M.xml, Bsp30_01N.xml, Bsp30_01S.xml, Bsp30_02.dll, Bsp30_02.dot, Bsp30_02.udl, Bsp30_02.xsd, Bsp30_02.xsl, Bsp30_02.zip, Bsp30_02M.xml, smartdoc.cls
Diverse	\Datenbank	ArtikelListe.txt, Nordwind.mdb, Personalstamm.xls

## Zusatz-Software: Die Dateien im Anhang

Sie finden auf der CD-ROM zum Buch einen weiteren Ordner mit der Bezeichnung *\Beilagen*. Darin sind zusätzliche Dateien abgelegt, die weitere Informationen zum Buchinhalt enthalten oder deren Inhalt an irgendeiner Stelle innerhalb des Buches besprochen wurde.

In der Tabelle G.2 sind die benötigten Informationen zusammengefasst. Sie können dort das jeweilige Thema, den Speicherort und die Namen der Beispieldateien entnehmen.

Zu folgenden Themen sind noch weitere Informationen auf der CD-ROM vorhanden:

- Die Datei *Dsofile.dll* bietet eine Schnittstelle, wie die Dateieigenschaften aller Office-Dateien von außerhalb bearbeitet werden können. Die entsprechenden Informationen und die zugehörige *.dll*-Datei sind im Ordner *\Beilagen\Dsofile* abgelegt.
- Im Ordner *\Beilagen\Interne Word-Befehle* ist eine Zusammenstellung aller internen Word-Befehle (deutsch und englisch) abgespeichert.
- Im Anhang A ist ein Vorschlag für Namenskonventionen aufgeführt. Dieser Vorschlag lehnt sich an eine allgemeingültige Namenskonvention an. Informationen zu diesem Thema sind im Ordner *\Beilagen\Namenskonventionen für VBA 5.0-6.0* zu finden.
- Um die Eigenschaften von TrueType-Schriften im Windows-Explorer darzustellen, wird von Microsoft ein Add-In angeboten. Dieses Add-In steht im Ordner *\Beilagen\TrueTypeFont Eigenschaften* zur Verfügung.
- Die Hilfe zum WordBasic-Objekt sowie zu den internen Dialogfeldern von Word ist in der aktuellen Version der Online-Hilfe sehr sparsam ausgefallen. Aus diesem Grunde ist es sinnvoll, dass bei Bedarf auf die frühere Hilfedatei aus Word 95 zugegriffen werden kann. Die Datei befindet sich im Ordner *\Beilagen\Word95 Wordbasic Hilfe*.
- Um erfolgreich mit WordProcessingML zu arbeiten, sind Kenntnisse zu diesen Schemas erforderlich. Die *XML Reference Schemas* für Office 2003 sind im Ordner *\Beilagen\XML Reference Schemas* bereitgestellt.
- Microsoft stellt verschiedene Software Development Kits (SDK) auf MSDN zum Herunterladen bereit. In diesem Buch wurden die *Smart Tag*- sowie *SmartDocument*-SDK als weitere Informationsquellen empfohlen. Die entsprechenden Dateien befinden sich im Ordner *\Beilagen\Smart Tag SDK* bzw. im Ordner *\Beilagen\Smart Document SDK*.



- Ein Dokument mit vertiefenden Informationen zum Thema »Suchen und Ersetzen« ist im Ordner *\Beilagen\SuchenErsetzen* gespeichert. Hier wird unter anderem die Funktionsweise von »Suchen mit Mustervergleich« genauer beschrieben.
- Die Anbindung von Word an Datenbanken kann auf verschiedene Arten erfolgen. Alle diese Möglichkeiten zu beschreiben, würde ein ganzes Buch füllen. Zusätzliche Informationen sind im Ordner *\Beilagen\Zusatzmaterial Seriendruck* abgelegt.

Tabelle G.2 Übersicht zu den weiteren Beilagen auf der CD-ROM

Thema	Speicherort	Name der Dateien
Dsofile.dll	<i>\Beilagen\Dsofile</i>	dsofile.exe
Interne Wordbefehle	<i>\Beilagen\Interne Word-Befehle</i>	Interne Word-Befehle.pdf
Namenskonventionen	<i>\Beilagen\Namenskonventionen für VBA 5.0-6.0</i>	./ (Internet-Link)
True Type-Schriften	<i>\Beilagen\TrueTypeFont Eigenschaften</i>	ttfont.exe
Wordbasic Hilfe	<i>\Beilagen\Word95 Wordbasic Hilfe</i>	Readme.txt, Wrdbasic.hlp
Smart Document	<i>\Beilagen\Smart Document SDK</i>	sdcsdk.msi
Smart Tag	<i>\Beilagen\Smart Tag SDK</i>	mstagsdk.msi
XML Reference Schemas	<i>\Beilagen\XML Reference Schemas</i>	xsdref.msi
Suchen und Ersetzen	<i>\Beilagen\SuchenErsetzen</i>	SuchenErsetzen.pdf
Seriendruck	<i>\Beilagen\Zusatzmaterial Seriendruck</i>	Odbc.pdf, Oledb.pdf, MSQuery.pdf, Sql.pdf



Anhang H

# Über die Autoren

Anhang



Cindy Meister verwendet seit 1988 Textverarbeitungsprogramme. Ihr Werdegang fing mit WordPerfect 5.0 für DOS an, wofür sie Tastaturmakros zusammenstellte. Als sie wider Willen auf Word 2.0 umstieg, entdeckte sie die Möglichkeiten von WordBasic und schaut seither nur dann wehmütig zurück, wenn komplexe Seriendruckaufgaben vorliegen. Die Einführung von VBA in Word 97 erweiterte ihre Horizonte (und ihre Bibliothek) in den objekt-orientierten Bereichen rund um Office und Visual Basic, inklusiv Datenintegration mittels DAO und später ADO. Während der Beta-Phase von Office 2003 stellte sie sich den neuen Herausforderungen von XML, SmartDocuments und VSTO (was zugleich die ersten Schritte in der Welt des .NET Framework

bedeutete). Ihre Lieblingsprojekte sind Gesamtlösungen, wo sie als Mitglied eines Entwicklerteams für die Word-Integration sorgt.

Das vorliegende Werk ist nicht die erste Veröffentlichung Cindy Meisters. Neben publizierten Artikeln in »redmonds Inside Word«, »Smart Access«, »Microsoft Office & Visual Basic for Applications Developer« und auf der MSDN-Webseite wirkte sie als Mitautorin von zwei weiteren Microsoft Press-Büchern mit: *Microsoft Word – Das Profibuch* und *Excel-Tipps*. Sie unterhält (etwas sporadisch) die Website <http://homepage.swissonline.ch/cindymeister>.

Cindy Meister wurde im Jahr 1996 erstmals in das MVP-Programm aufgenommen und erhielt seither jährlich diese Auszeichnung. Sie kann über die E-Mail-Adresse [cindymeister@mvp.org](mailto:cindymeister@mvp.org) erreicht werden.



Christian Freßdorf arbeitet seit zehn Jahren als Technischer Redakteur hauptsächlich mit Word und erstellt (auch firmenweit) Dokumentvorlagen und programmiert Funktionen und Abläufe zur Dokumentationserstellung und -automatisierung. Standen am Anfang WordBasic-Funktionen zur Steuerung und Generierung von Onlinehilfe-Projekten im Vordergrund, konnten mit Einführung von VBA auch komplexere Abläufe realisiert werden.

Seit fünf Jahren ist Christian Freßdorf aktiv in den deutschen Newsgroups und schwerpunktmäßig in [microsoft.public.de.word.vba](http://microsoft.public.de.word.vba) anzutreffen. Er unterstützt die Anwender bei Fragen zur VBA-Programmierung; seit 2004 ist er Microsoft-MVP. Darüber hinaus stellt Christian Freßdorf auf seiner Homepage [www.chf-online.de](http://www.chf-online.de) schwerpunktmäßig Add-Ins und Prozeduren

rund um die VBA-Programmierung zur Verfügung. Sie können ihn über die Kontaktdaten auf seiner Homepage oder per E-Mail unter [wordbuch@chf-online.de](mailto:wordbuch@chf-online.de) erreichen.



Thomas Gahler arbeitet seit 1991 in der Informatik und kennt Word seit diesen Tagen. Die Beziehung zu diesem Programm kann schon fast als »Liebe auf den ersten Blick« eingestuft werden. Er konnte die Entwicklung dieses Programms seit Word 5 für DOS mitverfolgen. Die ersten Makros wurden in WordBasic in der Version Word 2.0 für Windows entwickelt. Seit damals steht bei der Entwicklung seiner Erweiterungen der Anwender im Vordergrund. Dokument-Assistenten und zusätzliche Funktionen sollen diesen bei seinen täglichen Routinearbeiten unterstützen.

Thomas Gahler gibt seit über fünf Jahren sein Wissen auf [microsoft.public.de.word.vba](http://microsoft.public.de.word.vba) weiter und ist seit 2002 aktiver Microsoft-MVP. Fragen zum vorliegenden Buch können ihm unter der E-Mail-Adresse [wurzel2@bluemail.ch](mailto:wurzel2@bluemail.ch) gestellt werden.



Peter Jamieson hat seit 1979 in vielen Bereichen der Informatik, meistens in Großbritannien, aber auch in Benelux und anderen Ländern gearbeitet. Er ist spezialisiert auf technischen Support, Troubleshooting, unabhängige Verifikation und Validierung für große Projekte und Unterstützung bei großen Projekten im Allgemeinen. Word für Windows benutzt er seit der Version 1.0 und war lange Zeit aktiver Microsoft-MVP. Er wohnt in Manchester, England, und ist Besitzer eines »riese und müller Birdy« Fahrrads.

Peter Jamieson ist Mitautor eines weiteren Microsoft Press-Buches: *Microsoft Word – Das Profibuch*. Sie können ihn unter [pjj@pjjnet.demon.co.uk](mailto:pjj@pjjnet.demon.co.uk) kontaktieren.



# Verzeichnis zum Objektmodell

## A

ActiveDocument 202, 907  
Addin-Objekt  
    Add-Methode 486  
    Delete-Methode 216, 486  
Application-Objekt  
    Run-Methode 482  
Application-Objekt 191  
    ActivePrinter 198, 213  
    AutomationSecurity 193, 529  
    BackgroundPrintingStatus 209  
    ButtonClicks 327  
    CentimetersToPoints 348, 766  
    CustomizationContext 601  
    DisplayAlerts 193  
    Exists 548  
    KeysBoundTo 674  
    MacroContainer 735  
    MailingLabel 399  
    Options siehe Options-Objekt  
    PathSeparator 127, 194, 205  
    Quit 547  
    Run-Methode 216  
    ScreenUpdating 192, 231  
    Visible 192  
    WordBasic siehe WordBasic  
AttachedTemplate siehe Document-Objekt  
AutomationSecurity siehe Application-Objekt  
AutoTextEntry-Objekt  
    Add-Methode 117  
    Insert-Methode 216, 240  
    Name 117  
    StyleName 117

## B

BackgroundPrintingStatus siehe Application-Objekt  
Bookmark-Objekt 271  
    Add 272  
    Exists 276  
    Range.Text 275

## C

Collapse siehe Range-Objekt  
CommandBar-Objekt 650  
    ActionControl 658  
    AdaptiveMenu 655  
    AdaptiveMenus 655  
    Add-Methode 661  
    Control 655  
    DisableAskAQuestionDropdown 655  
    DisableCustomize 654  
    Enabled 654  
    Height 651  
    Index 650  
    Left 651  
    Name 650  
    NameLocal 650  
    Protection 654  
    RowIndex 651  
    Top 651  
    Type 651  
    Visible 654  
    Width 651  
ComputeStatistics siehe Document-Objekt  
Control-Objekt 655  
    Add-Methode 657  
    Caption 655  
    CommandBar 656  
    CommandBarComboBox 660  
    CommandBarComboBox, AddItem-Methode 660  
    Enabled 656  
    Execute-Methode 656  
    FindControl 655  
    ID 655  
    OnAction 658  
    Priority 655  
    Visible 656

## D

DefaultFliePath siehe Options-Objekt  
Designer-Objekt  
    Controls 743  
    Controls.Add-Methode 743  
Dialogfelder

    DefaultTab-Eigenschaft 636  
    Dialogs-Auflistung 631  
Dialogs-Objekt  
    Display-Methode 632  
    Execute-Methode 633  
    Show-Methode 632  
    Timeout-Parameter 632  
    Update-Methode 633  
    wdWordDialog-Konstanten 636  
DisplayAlerts siehe Application-Objekt  
Document-Objekt 202  
    Add-Methode 203, 205, 214  
    AttachedTemplate 205  
    AutoFormatOverride 259  
    CheckNewSmartTag 935  
    ComputeStatistics 204  
    DocumentProperty 595  
    EmbedSmartTags 935  
    FullName 205  
    IsMasterDocument 379  
    IsSubDocument 379  
    MailMerge 389  
    MailMerge.MailMergeType 389  
    Name 205  
    Open-Methode 203  
    PageSetup.Zoom 766  
    Path 205  
    PrintOut-Methode 208  
    RecheckSmartTags 935  
    RemoveSmartTag 935  
    SaveAs-Methode 207  
    Saved 207  
    Save-Methode 207  
    SmartTagsAsXMLProps 935  
    Type 205  
    Unprotect-Methode 384  
    Variable 264, 597

## E

Events 432  
    Document\_Close 434  
    Document\_New 434  
    Document\_Open 434  
    DocumentBeforeClose 450  
    DocumentBeforePrint 452

Events (*Fortsetzung*)

- DocumentBeforeSave 451
- DocumentBeforeSync 454
- DocumentChange 449, 773
- DocumentOpen 449
- MailMergeWizardSendToCustom 395
- NewDocument 448
- WindowActivate 441
- WindowBeforeDoubleClick 443
- WindowBeforeRightClick 444
- WindowDeactivate 442
- WindowSelectionChange 446, 773
- WindowSize 442
- WithEvents 439

**F**

Field-Objekt 321

- Add-Methode 323
- Code 324
- DoClick 327
- Kind 321, 326
- Lock 322
- Result 324
- Update-Methode 321
- UpdateSource 381

Fields-Auflistung

- Count-Eigenschaft 792

Fields-Eigenschaft 787

- Update-Methode 787

FileDialog-Objekt 638

- DialogType-Eigenschaft 638

- FileDialogFilters.Add-Methode 641

- FileDialogFilters.Clear-Methode 641

- FileDialogFilters-Auflistung 640

- Filters-Eigenschaft 640

- msoFileDialogFilePicker-Dialogfeld 645

- msoFileDialogFolderPicker-Dialogfeld 646

- msoFileDialogOpen-Dialogfeld 642

- msoFileDialogOpen-Konstanten 638

- msoFileDialogSaveAs-Dialogfeld 644

- Show-Methode 640

Find-Objekt

- ClearFormatting 234

- Execute 240

- Format 234, 260

- Forward 234–235

- Found 238

- MatchCase 234

- MatchSoundsLike 234

- MatchWholeword 234

- MatchWildcards 234

- Replacement 234

- Style 260, 279

- Text 234

- Wrap 234–235

Find-Oobjekt

- MatchAllWordForms 234

Footers-Auflistung 789

Formfield-Objekt 382

- Add 766

- EntryMacro 386

- ExitMacro 386

- Name 766

- Result 383

- Valid 383

Frames-Objekt

- Add 344

**H**

HeaderFooter-Objekt 371, 788

- Exists 374

- IsHeader, IsFooter 376

- LinkToPrevious 373

- PageNumbers 374

Headers-Auflistung 789

**I**

IIF-Funktion 644

InlineShape-Objekt 334

- AddOLEObject 540

- AddPicture-Method 334

- ConvertToInlineShape-Methode 341

- ConvertToShape-Methode 341

- LinkFormat 340

- OLEFormat 541

- OLEFormat.DoVerb-Methode 541

InRange-Methode 228

InStory-Methode 227

Italic-Eigenschaft 221

**K**

KeyBinding-Objekt 667

- Add-Methode 676

- BuildKeyCode 669

- ClearAll-Methode 675

- Clear-Methode 675

- Count 667

- Disable-Methode 675

- FindKey 669

- Key 669

- KeyString 670

- Rebind-Methode 677

**M**

MailMerge-Objekt

- DataSource 389

- DataSource.ActiveRecord 389

- DataSource.Included 391

- DataSource.RecordCount 390

- DataSource.SetAllIncludedFlags-Methode 392

- Destination 396

- Execute-Methode 396

- FindRecord 72, 393

- OpenDataSource-Methode 400

- ShowSendToCustom-Methode 395

- ShowWizard-Methode 394

Mso-Konstantwert

- MsoBarProtection 654

- MsoLanguageID 194

- MsoPresetTextEffect 560

- MsoPresetTextEffectShape 560

**N**

NewDocument-Objekt 698, 702

- Add-Methode 698

- Remove-Methode 700

NextStoryRange-Eigenschaft 784

NormalTemplate-Objekt 205, 216, 576

**O**

OperatingSystem siehe System-Objekt

Options-Objekt 194

- DefaultFilePath 196

- PrintBackground 208

- WPDNavKeys 195

- WPHelp 195



**P**

PageSetup-Objekt 363  
 DifferentFirstPageHeaderFooter 368  
 FirstPageTray 369  
 Gutter 366  
 Header-, FooterDistance 366  
 MarginBottom 365  
 MarginLeft 365  
 MarginRight 365  
 MarginTop 365  
 OddAndEvenPagesHeaderFooter 368  
 Orientation 365  
 OtherPagesTray 369  
 PaperSize 364  
 TextColumns 367  
 PathSeparator siehe Application-Objekt  
 PrintBackground siehe Options-Objekt  
 PrintOut siehe Document-Objekt

**R**

Range-Objekt 216, 219  
 Calculate-Methode 230  
 Collapse-Methode 221  
 ConvertToTable-Methode 282  
 Duplicate 229, 238  
 Font.Color 287  
 Font.Name 287  
 FormattedText 230, 279  
 GoTo-Methode 226  
 Information 227  
 InRange 326  
 InRange-Methode 228  
 InsertAfter-Methode 223  
 InsertBefore-Methode 223  
 InsertBreak-Methode 223  
 InsertParagraphAfter-Methode 223  
 InsertParagraphBefore-Methode 223  
 IsEqual 228  
 MoveEnd-Methode 224  
 MoveEndUntil-Methode 224  
 MoveEndWhile-Methode 224  
 Move-Methode 224  
 MoveStart-Methode 224  
 MoveStartUntil-Methode 224  
 MoveStartWhile-Methode 224  
 MoveUntil-Methode 224  
 MoveWhile-Methode 224

Paragraph.Border 288  
 ParagraphFormat.FirstlineIndent 288  
 Select-Methode 219  
 Shading 288  
 SpellingErrors 832  
 Text 221  
 TextRetrievalMode 229, 244  
 Range-Objekt siehe InStory-Methode  
 Reference-Objekt  
 AddFromFile-Methode 751  
 AddFromGUID-Methode 751  
 Count 749  
 IsBroken 754  
 Remove-Methode 754

**S**

Save siehe Document-Objekt  
 SaveAs siehe Document-Objekt  
 Saved siehe Document-Objekt  
 ScreenUpdating siehe Application-Objekt  
 Section-Objekt 357  
 ProtectedForForms 385  
 SelectCurrent siehe Selection-Objekt  
 Selection-Objekt 183, 200  
 Move 202  
 SelectCurrent 202  
 Type 201  
 WdSelectionType 201  
 Shape-Objekt 334, 788  
 AddOLEObject 541  
 AddPicture-Method 334, 808  
 AddTextbox-Method 817  
 AddTextEffect-Methode 559  
 ConvertToInlineShape-Methode 341  
 ConvertToShape-Methode 341  
 Left 346  
 LinkFormat 340  
 LockAnchor 346, 776  
 Nodes.Points 354  
 OLEFormat 541  
 OLEFormat.DoVerb-Methode 541  
 RelativeHorizontalPosition 346  
 RelativeVerticalPosition 346  
 TextEffect 563  
 TextFrame 356  
 Top 346  
 WrapFormat 348  
 ZOrder-Methode 350  
 ZOrderPosition 350

Shapes-Eigenschaft 788  
 SmartTag-Objekt 932, 934  
 CustomProperty 933  
 SmartTagAction 933  
 SmartTagRecognizer 933  
 SmartTagType 932-933  
 SpellingErrors siehe Range-Objekt  
 StoryRange-Objekt 781, 787  
 wdCommentsStory 781  
 wdEndnoteContinuationNoticeStory 783  
 wdEndnoteContinuationSeparatorStory 783  
 wdEndnoteSeparatorStory 782  
 wdEndnotesStory 781  
 wdEvenPagesFooterStory 782  
 wdEvenPagesHeaderStory 782  
 wdFirstPageFooterStory 782  
 wdFirstPageHeaderStory 782  
 wdFootnoteContinuationNoticeStory 782  
 wdFootnoteContinuationSeparatorStory 782  
 wdFootnoteSeparatorStory 782  
 wdFootnotesStory 781  
 wdMainTextStory 781  
 wdPrimaryFooterStory 782  
 wdPrimaryHeaderStory 782  
 wdTextFrameStory 782  
 StoryRanges-Auflistung 244, 359, 781, 784  
 NextStoryRange 360, 807  
 NextStoryRange-Eigenschaft 784  
 StoryRanges-Eigenschaft 781  
 Style-Objekt 255  
 Add-Methode 265  
 BuiltIn 262  
 Delete-Methode 260  
 InUse 259  
 LinkStyle 270  
 Locked 259  
 NameLocal 262  
 Type 265  
 Visibility 705  
 Subdocument-Objekt  
 AddFromFile-Methode 379  
 AddFromRange-Methode 380  
 Count 380  
 Delete-Methode 380  
 Expanded 380  
 HasFile 380  
 Level 380  
 Locked 380  
 Merge-Methode 380  
 Open-Methode 380

Subdocument-Objekt (*Fortsetzung*)  
 Path 380  
 Range 380  
 Split-Methode 380  
 System-Objekt 189  
     CountryRegion 189  
     Cursor 189  
     LanguageDesignation 189  
     OperatingSystem 190  
     PrivateProfileString 190, 589  
     Version 190

## T

Table-Objekt 280  
     Add 766  
     Borders 766  
     BottomPadding 291  
     Cell 282  
     Cell.Border 288  
     Cell.Merge 766  
     Cell.Split-Methode 300  
     Cell.VerticalAlignment 291, 766  
     Cells.Merge-Methode 300  
     Column.Width 297  
     Condition 765  
     ConvertToText-Methode 302  
     DefaultTableBehavior 764  
     FitText 291  
     ID 300  
     LeftPadding 291  
     NestingLevel 306  
     RightPadding 291  
     Row.AllowBreakAcrossPages 290  
     Row.Cells 282  
     Row.HeadingFormat 290  
     Row.Height 298, 764  
     Row.HeightRule 764  
     Row.Index 282  
     Row.LeftIndent 290  
     Rows.Add-Methode 282  
     Rows.Alignment 295  
     Rows.HorizontalPosition 296  
     Rows.RelativeHorizontalPosition 296  
     Rows.RelativeVerticalPosition 296  
     Rows.VerticalPosition 296  
     Rows.WrapAroundText 296  
     Shading 288, 766  
     Style 766  
     TopPadding 291  
     WordWrap 291  
 Template-Objekt 214  
     AutoTextEntry 216

OpenAsDocument-Methode 218  
 Type 218  
 Templates-Auflistung 216  
 ThisDocument-Objekt 58, 571

## U

UserForm-Objekt 608  
     Activate 614  
     Auflistung 616  
     Caption 611, 624  
     Deactivate 614  
     Font 624  
     Height 611, 624  
     Hide 612  
     Initialize 614  
     layout 614  
     Left 611, 624  
     Load 611  
     Move 613  
     Name 610  
     QueryClose 615  
     Repaint 613  
     Show 612  
     StartupPosition 611  
     Tag 611  
     Terminate 614  
     Top 611, 624  
     Unload 611  
     Width 611, 624

## V

VB-Anweisung  
     #Const 474  
     #If...Then 477  
     Debug.Print 112  
     Do-Schleife 108, 126, 238  
     Enum 96, 135, 491  
     For Each...Next 180  
     For...Next 110, 180  
     GetObjekt 122  
     If...Then...Else 106  
     Is 228  
     Kill 131  
     On Error Goto 119  
     On Error Resume Next 122  
     Open 131  
     Option Base 89  
     Redim 89  
     Redim Preserve 89  
     Resume 120  
     Select Case 98, 107, 121  
     Set 176

Type 93  
 VBComponent-Objekt  
     Add-Methode 737  
     CodeModule 723, 738  
         AddFromFile-Methode 738  
         AddFromString-Methode 739  
         InsertLines-Methode 741  
     CodeModule.CountOfDeclaration  
         nLiens 725  
     CodeModule.CountOfLines 724, 740  
     CodeModule.CreateEventProc-Methode 740  
     CodeModule.DeleteLines 733  
     CodeModule.InsertLines-Methode 739  
     CodeModule.Lines 724  
     CodeModule.ProcBodyLine 727  
     CodeModule.ProcCountLines 727  
     CodeModule.ProcOfLine 726  
     CodeModule.ProcStartLine 727  
     CodeModule.ReplaceLine 731  
     Count 722  
     Designer siehe Designer  
     Name 735  
     Remove-Methode 745  
     Type 722  
 VB-Datentyp  
     Boolean 79  
     Integer 80  
     Long 80  
     Object 81, 175  
     Single 80  
     String 79  
     Variant 80  
 VBE-Objekt 716  
     ActiveVBProject 719  
     FileName 719  
     Name 718  
     Protection 718  
     VBProject.Import-Methode 735  
     VBProject.References siehe Reference-Objekt  
     VBProject.VBComponents siehe VBComponent-Objekt  
     VBProjects 718  
 VB-Ereignis  
     Auto-Makros 184  
 VB-Funktion  
     CallByName 747  
     CBool 84  
     CInt 84  
     CLng 84  
     CreateObject 471

VB-Funktion (*Fortsetzung*)

CSng 84  
 CStr 84  
 CVar 84  
 CDate 766  
 Date 105  
 Dir 125, 139  
 Environ 157  
 FileDateTime 133  
 FileLen 134  
 Format 105–106, 766  
 GetAttr 128, 130  
 GetPoint 227  
 InputBox 100  
 Instr 104  
 InstrRev 104  
 IsDate 107  
 IsNumeric 106  
 LBound 90  
 Left 103  
 Mid 103  
 MsgBox 101  
 Replace 104  
 Right 103, 127  
 Timer 160  
 UBound 90  
 UCase 106  
 VB-Konstantwert  
   vbAlias 129  
   vbArchive 129  
   vbCR 104  
   vbDirectory 129–130  
   vbHidden 129

  vbNormal 129  
   vbObjectError 123  
   vbReadOnly 129  
   vbSystem 129  
   vbVolume 129  
 VB-Objekt  
   Err 123  
 VB-Schlüsselwort  
   ByRef 85, 87  
   ByVal 85–86  
   Nothing 83, 178  
   Optional 133  
   Private 81, 93, 96  
   Public 82  
   Step 182  
   WithEvents 439  
 Version siehe System-Objekt  
 View-Objekt  
   FieldShading 841  
   ShowTextBoundaries 365  
 Visible siehe Application-Objekt

**W**

Wd-Konstantwert  
   WdBreakType 224  
   WdBuiltinStyle 262  
   WdCollapseDirection 221  
   WdCursor 189  
   WdDefaultFilePath 196  
   WdDocumentType 205  
   WdFieldKind 321  
   WdHeaderFooterIndex 372

  WdKey 667  
   WdKeyCategory 667  
   WdMailMergeActiveRecord 390  
   WdMailMergeDestination 396  
   WdMailMergeMainDocType 389  
   WdMergeSubType 402  
   WdPaperTray 370  
   WdPrintOutItem 212  
   WdPrintOutPages 213  
   WdPrintOutRange 210  
   WdRowHeightRule 298  
   WdSelectionType 201  
   WdStatistic 204  
   WdStoryType 360  
   WdTablePosition 296  
   WdTaskPane 688  
   WdTemplateType 218  
   WdUnits 224  
 wdLine siehe Selection-Objekt, Move  
 WordBasic-Objekt 197  
   DisableAutoMacros 185, 194,  
     198, 434  
   FileNameInfo 198  
   FilePrintSetup 198  
   MailMergePropagateLabel 399  
   SelectSimilarFormatting 198  
   SortArray 198

**X**

XMLNamespace-Objekt 907



# Stichwortverzeichnis

## .NET

- aufgezeichneten Code bearbeiten 34
  - Konvertierung von Hilfe-Beispielen 78
  - Konvertierung von VBA ByVal / ByRef 86
  - Konvertierung von VBA User Defined Types 94
  - Konvertierung von VBA Variant 81
  - Konvertierung von VBA-Ganzzahlen 80
  - Konvertierung von VBA-Zeichenketten 79
  - optionale Argumente 203
  - Umwandlung von VBA-Datentypen 84
  - Verweis zur COM-Anwendung hinzufügen 69
- .NET Framework 477

## A

- Absatz
- Einzug der ersten Zeile 288
  - mit Rahmen versehen 288
- Abschnitt 357, 788
- Abschnittswechsel 357
  - Eigenschaften 357
- Abschnittsumbruch einfügen 223
- Absenderadresse eintragen 813
- Access siehe Fernsteuern
- ActiveX-Steuerelement 564
- Datengültigkeit prüfen 568
  - Entwurfs-Modus 565
  - Größe regeln 565
  - im Code ansprechen 571
  - Nachteile 564
  - navigieren 566
- Add-In
- Funktion aufrufen 491
  - Prozedur aufrufen 491
  - UserForm aufrufen 492
  - Vorlage (\*.dot) 490
- ADO 531, 536, 551
- Adressbuch 406
- Aktualisierung
- von Feldfunktionen 321
  - von Inhaltsverzeichnissen 322
- Alias siehe XML, Namensraumpräfix
- Anforderungen an ein Dialogfeld 624

## Anpassung

- Speicherort 600

## API 138

- Deklaration und Aufbau 138
- GetKeyState 163
- GetPrivateProfileSection 153
- GetPrivateProfileSectionNames 155
- GetUserNameEx 157
- PathAddBackslash 140
- PathAddExtension 142
- PathAppend 142
- PathCombine 141
- RegCloseKey 147
- RegEnumKeyEx 147
- RegOpenKey 147
- ShellExecute 143
- Sleep 159
- sndPlaySound 160

## Application-Objekt

- Language 194, 263
- Run-Methode 733

## Application-Objekt siehe VBE

## Array siehe Datenfeld

## Aufgabenbereich

- Clipart 689
- Dokument schützen 689, 694
- Dokumentaktionen 689, 693
- Dokumentaktualisierung 689, 693
- Einfache Suchoptionen 691, 693
- Erste Schritte 687, 689, 693, 702
- Faxdienst 689, 694
- Formatierungen anzeigen 690
- Formatvorlagen und Formatierungen 689, 695, 703
- Freigegebener Arbeitsbereich 691, 694
- Hilfe 690
- Neues Dokument 686, 689, 693, 698
- Recherchieren 690–691, 694
- selbstdefiniert 944
- Seriendruck 690, 694
- Startaufgabenbereich 686
- XML-Dokument 691
- XML-Struktur 691, 695, 895, 900
- Zwischenablage 689, 693

## Aufgabenbereich siehe Task Pane

## Auflistung

- bearbeiten 180
- Element ansprechen 179
- Element entfernen 181
- Index eines Objekts ermitteln 179

## AutoKorrektur

- definieren 581

## Auto-Makros 194, 432

- AutoClose 185
- AutoExec 185, 773
- AutoExit 185, 775
- AutoNew 185, 522, 532
- AutoOpen 185
- nicht starten 185
- und Makro-Sicherheit 434
- unterbinden 434
- Vergleich mit Ereignissen 435

## AutoText

- einer Kategorie zuweisen 117
- entfernen 579

## B

## Befehlszeilenargumente 1022

## Benutzerdefinierte Dialogfelder

- alle geladenen Objekte bearbeiten 616
- als Argument an Prozedur übergeben 617
- Anforderungen des Anwenders 624
- anzeigen 612
- benennen 610
- effizient erstellen 626
- Eigenschaften in Dateien auslagern 629
- Eigenschaften in Ini-Datei 629
- Eigenschaften in Txt-Datei 630
- Eigenschaften zwischenspeichern 626
- entladen 611
- Ereigniseintritt 615
- Ereignisse 614
- erstellen 609
- Fenstertitel eintragen 611
- flackern am Bildschirm verhindern 613
- laden 611
- neu zeichnen 613
- positionieren 611
- schließen verhindern 615
- Startposition festlegen 611
- Steuerelement Anzeige 619
- Steuerelement Befehlsschaltfläche 619
- Steuerelement Bezeichnungsfeld 618
- Steuerelement Bildlaufleiste 619
- Steuerelement Drehfeld 619

Benutzerdefinierte Dialogfelder (*Fortsetzung*)

- Steuerelement einbinden 618
- Steuerelement
  - Kombinationsfeld 618
- Steuerelement
  - Kontrollkästchen 618
- Steuerelement Listenfeld 618, 622
- Steuerelement Multiseiten 619
- Steuerelement Optionsfeld 618, 621
- Steuerelement Rahmen 619
- Steuerelement Register 619
- Steuerelement Textfeld 618
- Steuerelement Umschaltfeld 619
- verbergen 612
- verschieben 613
- zur Laufzeit beeinflussen 626
- Benutzernamen ermitteln (API) 157
- Benutzerschnittstelle anpassen
  - Möglichkeiten 600
  - Speicherort 600
  - Speicherort (COM-Add-In) 603
- Bereich
  - auf einen Punkt verkleinern 221
  - erweitern 224
  - festlegen 219
  - formatieren 221
  - formatierter Text 230
  - Formel berechnen 230
  - in Tabelle umwandeln 282
  - lokalisieren 226
  - schattieren 288
  - Text hinzufügen 223
  - Text zuweisen 221
  - und Feldcodes 229
  - und verborgener Text 229
  - vergleichen 227
  - verkleinern 224
  - verschieben 224
- Beschriftungen
  - mit Grafik zusammenhalten 822
  - wissenschaftliche 822
- Bildschirm
  - unsichtbar machen 192
- Bitweiser Vergleich 166
- BOM 864
- Browserobjekt 252, 254
- Byte Ordering Mark siehe BOM

**C**

- Codezeilen umbrechen, lange 104
- Codierung 864
- COM-Add-In
  - entfernen 680
  - installieren 679
  - mit Tastenkombination verbinden 677
- Compiler-Anweisung 477
- CustomDocumentProperties-Auflistung
  - siehe Document-Objekt

**D**

- Datei über Dateieindung ausführen (API) 143
- Dateiname auf Dokument eintragen 815
- Dateisystem
  - alle Dateien auflisten 125
  - Datei ist gesperrt 131
  - Datei ist vorhanden 128
  - Datei löschen 131
  - Dateidatum ermitteln 133
  - Dateigröße ermitteln 134
  - Ordnername mit Backslash ergänzen 127
  - Platzhalter 126
  - Verzeichnis ist vorhanden 129
- Datenbanken siehe Fernsteuern
- Datenfeld
  - dimensionieren 89
  - Größe ermitteln 90
  - sortieren 90
- Datentyp
  - Aufzählung 96
  - Benutzerdefinierter Typ 93
  - umwandeln 84
- Datum
  - Anzahl Tage im Monat berechnen 762
  - Wochentag berechnen 762
- Debuggen 111
  - Debug.Print 112
- Deklarationsbereich 725
- Delete Block-Aufforderung 195
- Dialogfelder
  - anzeigbare Dialogfelder 637
  - anzeigen 632
  - Argumente der integrierten Dialogfelder 634
  - Argumente der internen Dialogfelder 634
  - ausführen 632
  - benannte Konstanten 631
  - direkt ausführen 633
  - Eigenschaften 655
  - Einstellungen aktualisieren 633
  - Grafik einfügen 827
  - Index 636
  - integrierte 631
  - interne 631
  - Konstanten 636
  - nur ausführbare Dialogfelder 637
  - Rückgabewerte 632
  - Übersicht der WordBasic-Anweisungen 634
- Digitales Zertifikat 994
- Document Object Model siehe XML, DOM
- Dokument
  - als Textdatei speichern 207
  - Ansicht festlegen 578
  - ausdrucken 208
  - AutoWiederherstellen-Info 1026
  - Eigenschaft bearbeiten 595–596
  - Eigenschaft eintragen 578
  - einrichten, Seitenlayout 783
  - erstellen 203
  - in Vorlage umwandeln 205
  - mit einer Vorlage verbinden 205
  - mit Kennwort speichern 207
  - öffnen 203
  - schützen 384
  - speichern 207
  - Variable bearbeiten 597
- Dokumentbeschädigung 358
  - VBA-Code 75
- Dokumentdatum auf Dokument eintragen 815
- Dokumenteigenschaft siehe Dokument
- Dokumenteinstellungen
  - abspeichern 593
- Dokumentkomponenten
  - Bereiche 789
  - Übersicht 785
- Dokumentkomponentenbereiche 781, 789
- Dokumentvariable siehe Dokument
- Dokumentvorlage 214
- Druckerschacht 369
- Dsofile.dll 596

**E**

- Early Binding 470, 544, 571
- Eigenschaftenprozeduren 728
- Eingabeaufforderung
  - InputBox 100
- Elemente einer Auflistung wahlweise löschen 111
- Ereignisse 432
  - auf Applikationsebene 436
  - auf Applikationsebene anlegen 439
  - auf Applikationsebene deklarieren 439
  - auf Dokumentebene 434
  - Document\_Close (VSTO) 504
  - DocumentBeforeClose 450
  - DocumentBeforePrint 452
  - DocumentBeforeSave 451
  - DocumentBeforeSync 454
  - DocumentChange 449, 773
  - DocumentOpen 449
  - Gültigkeitsbereich 437
  - Klassenmodul einrichten 438, 773
  - Klassenmodul initialisieren 440, 773
  - NewDocument 448
  - Speicherort 437
  - WindowActivate 441
  - WindowBeforeDoubleClick 443
  - WindowBeforeRightClick 444
  - WindowDeactivate 442

Ereignisse (*Fortsetzung*)  
 WindowSelectionChange 446, 773  
 WindowSelectionChange (VSTO) 502  
 WindowSize 442  
 Excel  
 Diagramm im Word-Dokument 550  
 Tabellenobjekt im Word-Dokument 543  
 Tabellenobjekt, Größe festlegen 544  
 Tabellenobjekt, schließen 547  
 Excel siehe Fernsteuern  
 Excel-Tabelle in Word verknüpfen 329  
 Extensible Markup Language siehe XML  
 Extensible Stylesheet Language Transformation siehe XSLT  
 Externe Daten  
 einfügen 330  
 verknüpfen 328

## F

Facets siehe XML, Facetten  
 Falzmarke setzen 810  
 Farben in Combobox auflisten 834  
 Fehlerbehandlung 118  
 Fehlermeldungen  
 für Task Panes 687  
 Felder  
 aktualisieren 780, 787  
 aktualisieren über die Dokumentansicht 781  
 aktualisieren über Optionseinstellung 781  
 aktualisieren über Tastenkombination 781  
 automatisch aktualisieren 780  
 in Dokumentkomponenten aktualisieren 788  
 Locked-Eigenschaft 790  
 manuell aktualisieren 780  
 sperren 790  
 Feldfunktion  
 aktualisieren 321  
 Ausdruck 330  
 bestimmten Typ aktualisieren 824  
 CreateDate 371, 532  
 Database 530  
 Date 816  
 DocProperty 371  
 DocVariable 371  
 Feldcode bearbeiten 324  
 Feldcode in Text umwandeln 333  
 FileName 371, 817  
 If 371  
 in einem Textfeld 324  
 IncludePicture 341

IncludeText 988  
 Link 329  
 MacroButton 327, 537  
 MergeFormat-Schalter 323, 341  
 NumPages 371, 802  
 Page 371, 802  
 PrintDate 371, 816  
 Quote 816  
 SaveDate 371, 816  
 Section 371  
 SectionPages 371  
 Seq 824  
 StyleRef 371  
 verschachteln 330  
 Fernsteuern  
 Access 528  
 Access, ADO-Verknüpfung 531  
 Access, Bericht ausdrucken 529  
 Access, Datenbank lesen 530  
 Access, Formular anzeigen 528  
 Access, Tabelle in Word verknüpfen 530  
 Datenbanken 530  
 Excel 510, 536  
 Excel für komplexe Berechnungen verwenden 512  
 Excel, ADO-Verbindung 537  
 Excel, Arbeitsmappe drucken 511  
 Excel-Daten an Word übermitteln 486  
 Excel-Tabelle als Datenbank ansprechen 536  
 Outlook 520  
 Outlook-Dokument versenden 526  
 Outlook-Kalendereintrag drucken 520  
 Outlook-Kontakte als Empfängeradresse verwenden 521  
 Outlook-MailItem-Objekt 526  
 PowerPoint 513  
 PowerPoint-Präsentation drucken 513  
 PowerPoint-Struktur der Präsentation übernehmen 515  
 Visio 518  
 Visio-Zeichnung drucken 518  
 Word 488  
 Word-Dokument drucken 488  
 Fettformatierung 221  
 FileDialog  
 Filter löschen und erstellen 641  
 FileDialog-Dialogfeld 638  
 anzeigen und ausführen 640  
 Datei speichern 644  
 Dateien öffnen 642  
 Dateien wählen 645  
 Filter definieren 640  
 Mehrfachauswahl 642  
 Ordner wählen 646  
 Rückgabewerte 640

FileDialog-Objekt  
 Execute-Methode 640  
 Firmenlogo einfügen 803  
 Formatierung 255  
 Formatvorlage  
 anwenden 266  
 Auswahl in der Benutzerschnittstelle 256  
 Automatische Aktualisierung 270  
 definieren 266, 580  
 Einschränkungen 257  
 Kategorie ändern 112  
 neu erstellen 265  
 Vorteile 255  
 Word-interne 262  
 Formel im Bereich berechnen 230  
 Formel-Editor  
 automatisieren 823  
 Formular 381  
 ActiveX-Steuerelement 564  
 Feldresultate lesen und schreiben 383  
 Formularfeld-Ereignisse 386  
 Gültigkeitsprüfung 387  
 Optionenfelder 386  
 schützen 384  
 Formularfeld  
 in Tabellenzelle einfügen 762  
 Formularfeld-Schattierung 841  
 Fortschrittsbalken 780, 791  
 Aufbau 791  
 zur Laufzeit aktualisieren 792  
 Fußzeile siehe Kopf- und Fußzeilen

## G

GetSetting 588  
 Grafiken 334  
 Anker 345, 776  
 aus Kopfzeilen entfernen 809  
 AutoFormen mit VBA einfügen und bearbeiten 354  
 beschriften 343, 822  
 einfügen 334  
 in Kopfzeilen einfügen 803, 806  
 in Positionsrahmen 344  
 Konvertierung zwischen Shape und InlineShape 341  
 mit dem Text verschieben 346  
 positionieren 345  
 Reihenfolge 350  
 Textfluss-Formatierung 348  
 Verknüpfung ändern 340  
 Verknüpfung auflösen 342  
 Verknüpfung erstellen 340  
 GUID  
 erstellen 965  
 für Office 751

## H

Hauptdokumentbereich 781  
Hilfe 66  
    \*.chm-Dateien 67  
    Frage hier eingeben 66  
    Objektkatalog 69  
HTML 864

## I

IISAM 408  
Information siehe Range-Objekt  
INI-Dateien 152  
    als Zwischenspeicher 483  
    Werte lesen und schreiben 191  
Interop Assemblies 477  
Interoperabilität  
    Formel-Editor 823  
    MS Graph 555  
    WordArt 559

## K

Kalender-Steuerelement 762  
Klassenmodul  
    einrichten 773  
    initialisieren 773  
Kompatibilitäts-Optionen 585  
Komponentenbereiche  
    Kopf- und Fußzeilen 783  
    Haupttextbereich 783  
Konstante 91  
Konvertierfunktionen  
    Zentimeter nach Points 348  
Kopf- und Fußzeile 368, 371, 800  
    Absenderadresse 813  
    Briefkopf 813  
    Dateiname setzen 815  
    Datum setzen 815  
    Falz- und Lochmarke setzen 810  
    Firmenlogo einfügen 803  
    Seitenzahlen eintragen 801  
    Wasserzeichen eintragen 806  
Kopfzeile siehe Kopf- und Fußzeile  
Kursivformatierung 221

## L

Lange Dokumente 377  
Late Binding 470, 543, 571  
Lochmarke setzen 810

## M

MacroContainer 720  
Makro  
    Ausführung unterbinden 193  
    einer Symbolleiste zuweisen 39

einer Tastenkombination  
    zuweisen 41  
erscheint nicht in der Liste der  
    Benutzerschnittstelle 40  
kopieren 42  
löschen 42  
organisieren 42  
Reihenfolge bei  
    Namensgleichheit 708  
Speicherort 37  
synchron abarbeiten 482  
verschieben 42  
Zielgruppe 37  
zur Laufzeit nachladen 485  
Makrorekorder 31  
    anhalten 32  
    Arbeitsweise 33  
    Ergebnis betrachten 32, 57  
    starten 31  
Makrosicherheit  
    Projekt signieren 48  
    Sicherheitsstufe anpassen 43  
    Signatur erstellen 47  
    umgehen 193  
    Zertifikat einlesen 49  
    Zertifikat erstellen 48  
MAPI 405  
Markierung  
    Typ 827  
Mathematische Funktionen 512  
Menüleiste siehe Symbolleiste  
Menüs immer vollständig anzeigen 655  
MergeFormat 323  
Mitteilungen an den Benutzer  
    MsgBox 101  
MS Graph im Word-Dokument 555  
MsForms siehe Benutzerdefinierte Dia-  
logfelder  
MSXML-Parser siehe XML, Parser

## N

Namenskonflikt 185  
Namenskonventionen 1000  
Namespaces siehe XML, Namensräume  
Neue Zeile in einer Zeichenkette 104  
Nodes siehe Knotenpunkte  
Normal.dot  
    defekt 1026  
    konfigurieren 576

## O

Objekte  
    als Variablen verwenden 175  
    eingebettete 540  
    freigeben 178  
    im Objektmodell aufspüren 174  
    Standardeigenschaft 177

Objektkatalog 69  
ODBC  
    Textdateien 415  
OLE 540  
    Dateigröße 550  
    In-place-Aktivierung 540, 550  
    Objekt aktivieren 541  
    Objekt deaktivieren 542  
    Objekt einfügen 540  
    OLE-Client 540  
    OLE-Server 540  
Option Explicit-Anweisung 738  
Optionen  
    Options-Objekt 194  
    Registerkarte Bearbeiten 582  
    Registerkarte Kompatibilität 585  
    Registerkarte Rechtschreibung und  
        Grammatik 585  
    Registerkarte Sicherheit 584  
    Registerkarte Speichern 582  
Outlook siehe Fernsteuern

## P

Pfade  
    Backslash anhängen (API) 140  
    Datei an Pfad anhängen (API) 142  
    Kombinieren zweier Pfade  
        (API) 139  
    um Dateierweiterung ergänzen  
        (API) 142  
    Zwei Pfade kombinieren (API) 141  
PI (Processing Instruction) siehe Verar-  
beitungsanweisung  
Platzhalter siehe Dateisystem  
Positionsrahmen 826  
    Anker 345  
    einfügen 344  
    mit dem Text verschieben 346  
    positionieren 345  
PowerPoint siehe Fernsteuern  
PrivateProfileString siehe System-Objekt  
Programmbibliothek siehe Add-In  
ProgressBar-Control 795  
ProgressBar-Steuerelement 780, 796  
Projekt  
    kann nicht angezeigt werden 56

## R

Rechtschreibfehler 832  
Registry  
    Data Schlüssel 684  
    Editor starten 190  
    Sicherung 147  
    Task Pane-Einträge 686  
    Werte lesen und schreiben 190  
    Zugriff auf die Registry (API) 147  
Registry-Zugriff (API) 147



**S**

- SaveSetting 588
- Schaltflächen mittels Textfelder simulieren 794
- Schema.ini 415–416
- Schemabibliothek 868, 896
  - Schema laden 897
- Schriftfarbe 287
- Schritt für Schritt
  - Benutzerdefiniertes Dialogfeld erstellen 626
  - Userform erstellen 626
- Schritt-für-Schritt
  - AutoKorrektur-Eintrag erstellen 581
  - AutoText entfernen 579
  - Digitale Signatur erstellen 47
  - Digitale Signatur zuweisen 48
  - Early und Late Binding 475
  - Interne Word-Befehle auflisten 710
  - Makro einer Symboleiste zuweisen 39
  - Makro einer Tastenkombination zuweisen 41
  - Nachträglich ein geöffnetes Dokument transformieren 902
  - Normal.dot automatisch anlegen 577
  - Schema in die Schemabibliothek laden 897
  - Tabellenformatvorlage erstellen 292
  - Textdrucker installieren 213
  - Transformation mit einem Schema verbinden 905
  - XML-Datei beim Speichern transformieren 901
  - XML-Datei erstellen 863
  - Zertifikat einlesen 49
  - Zertifikat exportieren 48
- Seitenlayout 363
  - Bundsteg 366
  - festlegen 578
  - Ränder 365
- Seitenumbruch einfügen 223
- Seitenzahlen 374, 801
- SelfCert.exe 47, 994
- Seriendruck 388
  - Datenquelle einbinden 400
  - Datensatz suchen 393
  - Datensätze ausschließen 390
  - Datensätze navigieren 389
  - Datenverbindungen 403
  - Datenverbindungen (DDE, Access) 407
  - Datenverbindungen (DDE, Excel) 409
  - Datenverbindungen (Konvertierfilter) 404
  - Datenverbindungen (Konvertierfilter, Adressbücher) 405
  - Datenverbindungen (Konvertierfilter, Dokumente) 404
  - Datenverbindungen (Konvertierfilter, Textdateien) 405
  - Datenverbindungen (ODBC) 410
  - Datenverbindungen (ODBC, Access) 412
  - Datenverbindungen (ODBC, dBase) 419
  - Datenverbindungen (ODBC, Excel) 414
  - Datenverbindungen (ODBC, FoxPro) 420
  - Datenverbindungen (ODBC, MySQL) 423
  - Datenverbindungen (ODBC, Oracle) 422
  - Datenverbindungen (ODBC, Schema.ini) 416
  - Datenverbindungen (ODBC, SQL-Server) 417
  - Datenverbindungen (ODBC, Text) 415
  - Datenverbindungen (OLE DB) 424
  - Datenverbindungen (OLE DB, Access) 425
  - Datenverbindungen (OLE DB, dBase) 427
  - Datenverbindungen (OLE DB, Excel) 426
  - Datenverbindungen (OLE DB, FoxPro) 428
  - Datenverbindungen (OLE DB, Oracle) 429
  - Datenverbindungen (OLE DB, Paradox) 427
  - Datenverbindungen (OLE DB, SQL-Server) 425
  - Datenverbindungen (OLE DB, Textdatei) 426
  - Etiketten 398
  - Hauptdokument 389
  - Hauptdokument öffnen 394
  - Seriendruck-Assistent 394
  - Umschläge 396
  - zusammenführen 396
- Shape-Objekt
  - TextFrame-Eigenschaft 788
- Shapes
  - Textfelder 788
- Sicherheit
  - Outlook-Sicherheitswarnung 525
- Sicherheit siehe Makrosicherheit
- Sicherheitswarnung (Outlook) 525
- Signatur 46
- Silbentrennung
  - konfigurieren 580
- Smart Document 944
  - ControlCaptionFromID 976
- ControlCount 975
- ControlID 975
- ControlIndex 975
- ControlNameFromID 976
- ControlTypeFromID 977
- DLL erstellen 967
- DLL-Aktionen 973
- DLL-Erweiterungspaket 969
- DLL-Lösung 966
- DLL-Manifest 992
- Ereignisse des Interface 978
- Erweiterungspaket anfügen 948
- Erweiterungspaket hinzufügen 947
- InvokeControl 984
- ISmartDocument-Interface 973
- Lösung entfernen 952
- Manifest 963
- Manifest digital signieren 994
- mit Erweiterungspaket verbinden 955
- MOSTL 945
- MOSTL aktualisieren 948
- MOSTL installieren 946
- MOSTL und Unicode 951, 958
- MOSTL und XSLT 952, 956
- MOSTL, Dokumentaktionen 949
- MOSTL-Aktionen 958
- MOSTL-Komponente über Erweiterungspaket installieren 952
- MOSTL-Lösung-Datei 958
- OnListOrComboSelectionChange 983
- OnPaneUpdateComplete 979
- PopulateHelpContent 980
- PopulateListOrComboContent 981
- PopulateRadioGroup 982
- SDK 945
- SmartDocInitialize 978
- SmartDocXmlCaption 974
- SmartDocXMLTypeCount 973
- SmartDocXMLTypeName 974
- speichern 954
- Speicherort der Komponenten 953
- Steuerelemente ansprechen 984
- Type Name 958
- XMLTypeID 973
- Smarttag 912
  - Aktionen 914
  - Aktionen festlegen (MOSTL) 924
  - Ausnahmeliste 915
  - Automatische Aktualisierung 918, 921
  - COM DLL 916, 924
  - COM DLL installieren 931
  - eigenes entwickeln 916
  - Erkennungsausdrücke festlegen (MOSTL) 923
  - Grundtypen 912
  - in der Benutzeroberfläche 912
  - in VB6 entwickeln 924
  - ISmartTagAction-Interface 930

Smarttag (*Fortsetzung*)  
 ISmartTagRecognizer-Interface 929  
 MOSTL 916–917  
 MOSTL lokalisieren 919  
 MOSTL-Speicherorte 919  
 Office Smart Tag SDK 916  
 Optionen 915  
 Property Bag 914  
 Recognizer 914  
 Smarttag-Typ 914  
 Untermenüs 917  
 Word-Objektmodell 932  
 Speichern  
 AutoWiederherstellen-Info 1026  
 Speicherort 31  
 Sprache der Word-Umgebung 194  
 Startaufgabenbereich 686  
 Startoptionen 1022  
 StartUp-Ordner 196, 215, 491, 717  
 Statistische Angaben eines Dokuments 204  
 Steuerelement 742, 794  
 einbinden 469  
 Ereignisse (VSTO) 503  
 ProgressBar 795  
 SpecialEffect-Eigenschaft 794  
 Zugriff auf unsichere ActiveX-Controls 470  
 Steuerelement-Ereignis  
 MouseDown 794  
 MouseUp 794  
 Suchen und Ersetzen 233  
 Dialogfeld zurücksetzen 253  
 im ganzen Dokument 244  
 in einer Schleife ausführen 236  
 nach Benutzereingriff zurücksetzen 252  
 Suche mit Mustervergleich nicht abgeschlossen 250  
 Unterschiede zur Benutzerschnittstelle 234  
 Zurücksetzen nach erfolgloser Suche mit Mustervergleich 251  
 Symbolleiste 650  
 Adaptive Menüs 655  
 alle Steuerelemente anzeigen 655  
 Anpassungen unterbinden 654  
 Bestimmte Steuerelemente immer anzeigen 655  
 erstellen 661  
 Kontextmenü erstellen 661  
 positionieren 651  
 schützen 654  
 Steuerlement-Toolbox 565  
 Symbolschaltfläche 655  
 Verfügbare auflisten 650  
 Symbolleiste  
 mit aktiven Schaltflächen 772  
 Symbolschaltfläche 655  
 ausführen 656

Bearbeitungsfeld 660  
 betätigte abfangen 658  
 Dropdownliste 660  
 identifizieren 655  
 mit Makro verbinden 658  
 mit Word-Befehl belegen 657  
 Popup-Menüs 656  
 Sichtbarkeit 655

## T

### Tabelle

als Kalenderblatt 762  
 aus Access einbinden 530  
 automatisieren 280  
 Daten lesen 300  
 dem Papierformat anpassen 764  
 Einzug 290  
 erstellen 823  
 Exakte Zeilenhöhe festlegen 764  
 Excel-Tabellenobjekt 543  
 formatieren 287, 823  
 Höhe 299  
 im Dokument finden 300  
 Kopfzeile wiederholen 290  
 mit Textfluss auf einer Seite behalten 297  
 positionieren 295  
 schattieren 288  
 Senkrechte Textausrichtung in einer Zelle 291  
 Spalte formatieren 289  
 Spaltenbreite 297  
 Tabellenformatvorlage 765  
 Text anpassen 291  
 verschachtelte 304  
 Zeile formatieren 289  
 Zeilenhöhe 298  
 Zellen teilen 300  
 Zellen verbinden 300, 763  
 Zellenbegrenzung 291  
 Zellenumbruch 291  
 Zellenwechsel 290

Tabellenformatvorlage 292  
 diagonale Rahmenlinien 294  
 Formatierung von Eckzellen 294  
 Kopfzeile wiederholen 294  
 Standard für neue Tabellen 294  
 Streifen 293  
 Tabelleneigenschaften 294

Task Pane 684  
 abfangen oder außer Kraft setzen 693  
 beim Starten einen bestimmten einblenden 692  
 Eigenen erstellen 684  
 einen bestimmten einblenden 692  
 Fehlermeldungen 687  
 Inhalt aktualisieren 693

Startaufgabenbereich kontrollieren 686  
 Typen 688  
 und der Makrorekorder 684  
 Welcher ist eingeblendet 691  
 Task Pane siehe Work pane  
 Tastenkombination  
 (Alt)+(F11) 32, 54  
 (Alt)+(F9) 323, 328, 341  
 (F1) 58, 67  
 (F2) 69  
 (F8) 113  
 (F9) 115  
 (Strg)+(Bild ab) 253  
 (Strg)+(Bild auf) 253  
 (Strg)+(F9) 116, 322  
 (Strg)+(G) 113  
 (Strg)+(S) 74  
 (Strg)+(Umschalt)+(F2) 73  
 (Strg)+(Umschalt)+(F7) 381  
 (Strg)+(Umschalt)+(F9) 116  
 (Umsch)+(Alt)+(-) 684  
 (Umschalt)+(F2) 73  
 (Umschalt)+(F9) 114  
 Alle auflisten 671  
 aussperren 675  
 entfernen 675  
 für bestimmten Befehl ermitteln 674  
 programmtechnisch anlegen 676  
 programmtechnisch ermitteln 669  
 programmtechnische Definition 667  
 Tastenstatus ermitteln (API) 163  
 Tastenwerte  
 ASCII-Werte 163  
 Virtuelle Tastenkonstanten 163  
 Textdateien  
 ODBC 415  
 Textfelder 788  
 auf Textinhalt prüfen 788  
 einfügen 818  
 formatieren 818  
 in Kopf- und Fußzeilen 788  
 Verknüpfte Textfelder 784  
 vertikale Textrichtung 818  
 Textmarke 271  
 Benennung von Textmarken 271  
 Textmarke (*Fortsetzung*)  
 Daten schreiben 275  
 einfügen 272  
 Inhalt lesen 277  
 verborgene 271  
 vordefinierte 278  
 ThisDocument-Modul 57, 434  
 ThisDocument-Objekt 734  
 TrueType-Schrift  
 Eigenschaften 583  
 einbetten 582

**U**

Umgebungsvariablen  
 Environ-Funktion 157  
 Umwandlungsfunktionen siehe Datentyp  
 Uniform Resource Identifier siehe URI  
 URI 880  
 Userform siehe Benutzerdefinierte Dialogfelder  
 UTF-8 siehe Codierung

**V**

Variable 78  
 deklarieren 79, 82  
 Gültigkeit 81  
 Sichtbarkeit 81  
 Werte nachprüfen 114  
 Variables-Auflistung siehe Document-Objekt  
 VBA-Code 714  
 VB-Anweisung  
 GetSetting 588  
 SaveSetting 588  
 VBA-Projekte 717  
 VBE  
 AddFromString-Methode 739  
 Application-Objekt 720  
 Document-Objekt 720  
 MacroContainer-Eigenschaft 720  
 Makro ausführen 733  
 Property Get 728  
 Property Let 728  
 Property Set 728  
 Property-Prozeduren 728  
 References-Auflistung 749  
 Show-Eigenschaft 746  
 Template-Objekt 720  
 UserForm-Objekt 746  
 UserForms-Auflistung 746  
 Verweise neu setzen 754  
 VB-Editor  
 Code speichern 74  
 Code-Fenster 59, 73  
 Debuggen 111  
 Direktbereich 113  
 Eigenschaftenfenster 57, 619  
 Fensterlayout 55  
 Haltepunkt festlegen 115  
 Haltepunkte entfernen 116  
 Hilfe 54  
 IntelliSense 74  
 Lange Codezeilen umbrechen 104  
 Lesezeichen 73  
 Nächste Anweisung festlegen 116  
 programmtechnisch steuern 716  
 Projekt-Explorer 56  
 Sicherheitskopie erstellen 74, 609  
 Symbolleisten 60  
 Überwachungsbereich 114  
 Werkzeugsammlung 619  
 VB-Funktion  
 OnTime 733  
 Replace 488  
 VB-Funktionen  
 DateDiff 492  
 Rnd 808  
 VB-Schlüsselwort  
 WithEvents 773  
 Verarbeitung unterbrechen (API) 159  
 Verknüpfungen 328  
 relative Pfadangaben 341  
 Verweis 466  
 auf Add-In setzen 491  
 Early Binding 470, 474, 510  
 Late Binding 471, 474, 510  
 manuell einfügen 467  
 unterschiedliche Versionen 473  
 Verzeichnisfelder 788  
 Visio siehe Fernsteuern  
 Visual Basic-Editor 714  
 Code-Fenster 715  
 Eigenschaftenfenster 715  
 Microsoft Visual Basic for Applications Extensibility 5.3 714  
 Option 'Zugriff auf Visual Basic-Projekt vertrauen' 714  
 Projekt-Explorer 715  
 Sicherheitsstufe anpassen 714  
 UserForm-Fenster 715  
 Werkzeugsammlung 795  
 Vorlage-Add-In  
 Code anzeigen 56  
 VSTO 495  
 Beispiel 499  
 Ereignisse 502  
 FindControl-Prozedur 503  
 Nachteile 496  
 Projekt erstellen 496  
 Symbolleiste erstellen 501  
 und Dokumenteigenschaften 497  
 vom Assistent generierter Code 498  
 Vorteile 495

**W**

Wasserzeichen 806  
 Wave-Dateien abspielen (API) 160  
 Wd-Konstantwert  
 WdFieldType 1005  
 Windows-Registrierung  
 Data-Schlüssel löschen 1026  
 Werte einlesen 588  
 Werte speichern 588  
 Word  
 Startoptionen 1022  
 Word siehe Fernsteuern  
 Word XML-Toolbox 900

WordArt 559  
 in Kopfzeilen einfügen 809  
 WordBasic 197  
 Word-Befehle  
 übersteuern 708  
 unabhängig von  
 Programmiersprache 711  
 Word-Meldungen unterbinden 193  
 WordML siehe WordProcessingML  
 WordProcessingML 868, 890  
 WordProcessingML XSLT Inference Tool 907  
 Work Pane 684  
 Objektmodell 685  
 Registry-Einträge 686

**X**

XHTML 864, 867  
 XML 862  
 Attribut 871  
 Aus Access exportieren 955  
 Benutzerdefiniertes Vokabular in Word 894  
 CDATA 873  
 Datei erstellen 863  
 Deklarationen 872  
 Dokument beim Öffnen  
 transformieren 902, 936  
 Dokument beim Speichern  
 transformieren 901  
 Dokumentinstanz 873  
 DOM 870, 885, 937  
 Element 870  
 Facetten 876  
 Gültiges Dokument 874  
 Knotenpunkte 885, 887  
 Markup 873  
 Namensräume 869, 878  
 Namensraumpräfix 880, 882–883, 896  
 Nutzen in Word 868  
 Parser 870, 908  
 Schema 875, 878  
 Schema in Schemabibliothek laden 897  
 Schema in Word einem XML-Dokument zuweisen 899  
 Schema programmtechnisch der Schemabibliothek hinzufügen 908  
 Schemaverletzungen einblenden 954  
 Solutions in Word 901  
 Tag 865, 870  
 Transformation mit dem DOM 908  
 Transform Inference Tool 956  
 Transformation automatisch erstellen 907  
 Transformation in Word 901

XML (*Fortsetzung*)

- Transformation mit dem  
DOM 936
- Transformation mit Schema  
verbinden 905
- transformieren 866
- über IncludeText einfügen 988
- Verarbeitungsanweisung 873, 893
- Vokabular 864
- Wohlgeformtes Dokument 874
- Wurzelelement 873, 882
- XPath 870
- XML-Erweiterungspaket
- Sicherheitsmeldung 948

- XMLSign.exe 994
- XSD siehe XML, Schema
- XSLT 866, 870, 886, 937
  - Werte aus Attributen lesen 991

**Z**

- Zeichencodierung siehe Codierung
- Zeichenkette
  - verknüpfen 85
- Zeichenkettenbearbeitung
  - Format 105
  - Instr 104

- InstrRev 104
- Left 103
- Mid 103
- Replace 104
- Right 103
- Zeilenweise verschieben 202
- Zeitungsspalten 367
- Zentral-/Filialdokumente 377
- Zertifikat 45
- Zoomfaktor festlegen 578
- Zwischenablage
  - Inhalte einfügen 337
  - Office 689