

Cindy Meister, Thomas Gahler, Christian Freßdorf, Peter Jamieson

**Microsoft Word-Programmierung**  
**Das Handbuch, 2. überarbeitete Auflage**



Cindy Meister, Thomas Gahler, Christian Freßdorf, Peter Jamieson

# Microsoft Word-Programmierung Das Handbuch

**Microsoft<sup>®</sup>**  
*Press*

Cindy Meister, Thomas Gahler, Christian Freßdorf, Peter Jamieson:  
Microsoft Word-Programmierung – Das Handbuch, 2. überarbeitete Auflage  
Microsoft Press Deutschland, Konrad-Zuse-Str. 1, D-85716 Unterschleißheim  
Copyright © 2008 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  
10 09 08

ISBN 978-3-86645-414-9

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

Fachlektorat: Georg Weiherer, Münzenberg  
Korrektorat: Jutta Alfes, 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

Vorwort .....	19
<b>Teil A</b>	
<b>Grundlagen der Arbeit mit VBA .....</b>	<b>25</b>
1 Word-Makros .....	27
2 VBA-Grundlagen .....	63
3 Windows-APIs in VBA nutzen .....	125
<b>Teil B</b>	
<b>Das Objektmodell von Word .....</b>	<b>159</b>
4 Überblick über die Arbeit mit Objekten .....	161
5 Grundlagen des Word-Objektmodells .....	173
6 Professionelle Dokumente .....	241
7 Word und Datenstrukturen .....	339
8 Ereignisse in Word .....	447
<b>Teil C</b>	
<b>Steuern und gesteuert werden .....</b>	<b>479</b>
9 Grundlagen der Fernsteuerung .....	481
10 Word von anderen Umgebungen aus steuern .....	499
11 Andere Programme von Word aus steuern .....	559
12 Eingebettete OLE-Objekte .....	589
<b>Teil D</b>	
<b>Optimierung der Benutzerschnittstelle .....</b>	<b>625</b>
13 Anwendungsoptionen .....	627

14	Speicherort der Anpassungen .....	655
15	Mit Dialogfeldern arbeiten .....	665
16	Menüs und Symbolleisten .....	707
17	Multifunktionsleisten (Ribbons) .....	723
18	Tastaturbelegungen .....	761
19	Aufgabenbereiche .....	779
20	Interne Word-Befehle übersteuern .....	807
21	Zugriff auf den Visual Basic-Editor (VBE) .....	813

## Teil E

### XML und Smart-Technologien

	XML und Smart-Technologien .....	859
22	Word und XML: Eine Einführung .....	861
23	Smarttags .....	909
24	Smart Documents .....	943

## Teil F

### Anhang

	Anhang .....	999
A	Namenskonventionen für Word-VBA .....	1001
B	Feldfunktionen .....	1005
C	Bezeichnungen von Symbolleisten .....	1013
D	Startoptionen von Word .....	1025
E	Bei Problemen – Word zurücksetzen .....	1029
F	Nützliche Links ins Internet .....	1033
G	Inhalt der CD-ROM .....	1035
H	Über die Autoren .....	1041
	Verzeichnis zum Objektmodell .....	1045
	Stichwortverzeichnis .....	1053

# Inhaltsverzeichnis

Vorwort .....	19
Für wen wurde das Buch geschrieben? .....	20
Word-Versionen .....	21
Wie ist das Buch aufgebaut? .....	22
Stichwortverzeichnisse .....	23
Die CD-ROM zum Buch .....	23
Danksagung .....	24
 <b>Teil A</b>	
<b>Grundlagen der Arbeit mit VBA .....</b>	<b>25</b>
 <b>1 Word-Makros .....</b>	<b>27</b>
Aller Anfang ist schwer .....	28
Programmierhilfen .....	29
Den Makrorekorder einsetzen .....	29
Unterstützende Werkzeuge des VB-Editors .....	36
Die Objektmodell-Hilfe – eine versteckte Schatzkammer .....	41
Makros in die Benutzerschnittstelle einbinden und verwalten .....	47
Makro einer Symbolschaltfläche zuweisen .....	47
Makro einer Tastaturkombination zuweisen .....	49
Makros kopieren .....	51
Makrosicherheit .....	52
Sicherheitsstufe anpassen .....	52
Eigene Signatur mittels <i>selfcert.exe</i> erstellen .....	57
VBA-Projekte mit einer Signatur versehen .....	58
Zertifikat zur Signatur erstellen .....	59
Zertifikat einlesen .....	60
Zusammenfassung .....	61
 <b>2 VBA-Grundlagen .....</b>	<b>63</b>
Variablen .....	64
Standard-Datentypen .....	65
Gültigkeit bzw. Sichtbarkeit .....	67
Umwandlung von Datentypen .....	70
Weiterreichen von Variablen an Prozeduren .....	71
Variablen in Datenfeldern ablegen .....	74
Konstanten .....	77

Benutzerdefinierte Typen .....	79
Type-Anweisung .....	79
Enum-Anweisung .....	82
Nützliche VBA-Funktionen .....	86
Bedingungen .....	92
Schleifen .....	95
Code im VB-Editor debuggen .....	98
Der Überwachungsbereich .....	101
Fehlerbehandlung .....	105
Dateisystem-Operationen .....	112
Alle Dateien eines Verzeichnisses auflisten .....	112
Verzeichnisname mit Backslash ergänzen .....	113
Prüfen, ob eine bestimmte Datei vorhanden ist .....	115
Prüfen, ob ein bestimmter Ordner vorhanden ist .....	116
Prüfen, ob eine Datei von jemanden im Zugriff ist .....	117
Datei löschen .....	118
Letztes Speicherdatum einer Datei ermitteln .....	119
Größe einer Datei ermitteln .....	121
Zusammenfassung .....	123
 3 Windows-APIs in VBA nutzen .....	125
Aufbau der API-Funktionen .....	126
Kombinieren und Abschließen von Pfaden .....	127
Abschließen eines Pfades .....	128
Kombinieren von Pfaden .....	129
Dateinamen an Pfad anhängen .....	130
Dateierweiterung an Datei anhängen .....	130
Datei über die Dateieindung ausführen .....	131
Zugriff auf Registry-Einträge .....	135
Ermitteln von Registry-Einträgen und Werten .....	135
Zugriff auf INI-Dateien .....	140
Auslesen und Schreiben von Abschnitten .....	141
Ermitteln aller Abschnitte einer INI-Datei .....	143
Name des angemeldeten Benutzers ermitteln .....	145
Verarbeitung für eine bestimmte Zeit unterbrechen .....	147
Wave-Datei abspielen .....	148
Tastenstatus abfragen .....	151
Zusammenfassung .....	157
 <b>Teil B</b>	
<b>Das Objektmodell von Word .....</b>	<b>159</b>
 4 Überblick über die Arbeit mit Objekten .....	161
Arbeiten mit Objekten .....	162
Das gesuchte Objekt aufspüren .....	162
Objekte als Variablen .....	163

Auflistung von Objekten .....	167
Auflistung bearbeiten .....	168
Zusammenfassung .....	171
<b>5 Grundlagen des Word-Objektmodells .....</b>	<b>173</b>
Das <i>System</i> -Objekt .....	174
Die Anwendung: Das <i>Application</i> -Objekt .....	177
Die gegenwärtige Markierung: <i>Selection</i> und ähnliche Objekte .....	186
Der Kern der Sache: Das <i>Document</i> -Objekt .....	189
Dokumentvorlagen: Das <i>Template</i> -Objekt .....	203
Mit Bereichen arbeiten: Das <i>Range</i> -Objekt .....	207
Einen Bereich definieren .....	208
Einen Bereich bearbeiten .....	209
Wo befindet sich der Bereich? .....	215
Text aus einem Bereich lesen .....	217
Eine Formel berechnen .....	219
Die Nadel im Heuhaufen: <i>Find/Replace</i> einsetzen .....	221
Vor- und Nachteile des Makrorekorder-Resultats .....	221
Anpassung des aufgezeichneten Codes .....	226
Bekannte Probleme vermeiden oder beheben .....	239
Zusammenfassung .....	240
<b>6 Professionelle Dokumente .....</b>	<b>241</b>
Abschnitte im Dokument: Das <i>Section</i> -Objekt .....	242
Bereiche im Dokument: Das <i>StoryRanges</i> -Objekt .....	244
Die Seite definieren: Das <i>PageSetup</i> -Objekt .....	248
Die Seite gestalten: Das <i>HeaderFooter</i> -Objekt .....	256
Lange Dokumente .....	262
Zentraldokumente .....	263
Dokumente verknüpfen .....	266
Bausteine: Das <i>BuildingBlocks</i> -Objekt .....	267
Schnellbausteine .....	268
Die zentrale Bausteinvorlage <i>Building Blocks.dotx</i> .....	270
Die Dokumenteigenschaften .....	271
Felder (Feldfunktionen) .....	272
Der Schnellbaustein-Katalog .....	272
Die Bausteine ( <i>BuldingBlockTypes</i> ) .....	277
Der Organizer für Bausteine .....	280
Formatieren mit Stil: Das <i>Style</i> -Objekt .....	282
Formatvorlagen .....	282
Benutzerschnittstellen für Formatvorlagen .....	283
Formatierungseinschränkungen .....	284
Word-interne Formatvorlagen .....	289
Formatvorlagen erstellen und modifizieren .....	292
Wilde Formatvorlagen .....	296

Automatische Nummerierung mit Listen .....	298
Listeigenschaften ermitteln .....	298
Listvorlagen ( <i>ListTemplates</i> ) .....	300
Grafiken: Die <i>InlineShape</i> - und <i>Shape</i> -Objekte .....	309
Grafiken einfügen .....	310
Verknüpfungen erstellen und verwalten .....	315
Layout-Optionen .....	319
Zeichnungsobjekte ( <i>AutoFormen</i> ) .....	335
Zusammenfassung .....	338
<b>7 Word und Datenstrukturen .....</b>	<b>339</b>
Zielscheibe Textmarke: Das <i>Bookmark</i> -Objekt .....	340
Inhalt mit Tabellen strukturiert darstellen .....	349
Tabellen erstellen .....	350
Tabellen formatieren .....	356
Informationen aus Tabellen holen .....	370
Feldfunktionen .....	378
Verknüpfungen, Tabellen und Berechnungen .....	385
Formulare: Das <i>FormField</i> -Objekt .....	391
Die Alternative zu Formularfeldern: <i>ContentControls</i> -Objekt .....	399
Die Grundlagen .....	400
Inhaltssteuerelemente und XML .....	407
Inhaltssteuerelemente im Objektmodell .....	412
Der Seriendruck: Das <i>MailMerge</i> -Objekt .....	427
Datenquelle einbinden .....	441
Einige Beispiel-Datenverbindungen .....	443
Zusammenfassung .....	446
<b>8 Ereignisse in Word .....</b>	<b>447</b>
Auto-Makros als Pseudo-Ereignisse .....	448
Ereignisse auf Dokumentebene .....	450
Ereignisse auf Applikationsebene .....	453
Klassenmodule einrichten und initialisieren .....	455
Klassenmodule einrichten .....	455
Klassenmodule initialisieren .....	457
Übersicht über die verfügbaren Ereignisse .....	458
WindowActivate .....	458
WindowDeactivate .....	459
WindowSize .....	459
WindowBeforeDoubleClick .....	460
WindowBeforeRightClick .....	461
WindowSelectionChange .....	463
NewDocument-Ereignis .....	465
DocumentOpen .....	466
DocumentChange .....	466
DocumentBeforeClose .....	467
DocumentBeforeSave .....	468

DocumentBeforePrint .....	469
DocumentSync .....	471
Seriendruckereignisse .....	472
MailMergeBeforeMerge .....	472
MailMergeBeforeRecordMerge .....	472
MailMergeAfterRecordMerge .....	473
MailMergeAfterMerge .....	473
Beispiel: 1:n-Liste im Seriendruckresultat .....	473
Zusammenfassung .....	478
 <b>Teil C</b>	
<b>Steuern und gesteuert werden</b> .....	479
 <b>9 Grundlagen der Fernsteuerung</b> .....	481
Verweise auf externe Bibliotheken im VB-Editor .....	482
Verweise in Visual Studio 2005 .....	487
Early versus Late Binding .....	489
Vorteile von Early Binding .....	489
Vorteile von Late Binding .....	491
Ganz clever, wer beide Arten kombiniert .....	492
Late Binding in Visual Studio .NET .....	495
Zusammenfassung .....	497
 <b>10 Word von anderen Umgebungen aus steuern</b> .....	499
Fernsteuern von Microsoft Word .....	500
Makros von außen anstoßen .....	500
Word effektiv fernsteuern .....	504
Versteckte Word-Instanz steuern .....	506
Programmzeilen zentral verwalten und gemeinsam nutzen (Add-Ins) .....	508
Dokumentvorlagen (.dot) als Add-In .....	508
Add-In-Prozeduren in anderen Umgebungen nutzen .....	512
Fernsteuerung aus Office-fremden Umgebungen .....	513
Die Word-Anwendung über Visual Studio .NET fernsteuern .....	513
Eine laufende Instanz ansprechen sowie Word starten .....	514
COM-Anwendung-Ressourcen freigeben .....	517
VSTO 2005-Dokumentlösungen .....	519
Eine VSTO-Lösung anlegen .....	520
Der Code »hinter dem Dokument« .....	523
Das VSTO-Dokument vorbereiten .....	524
Die Benutzerschnittstellen einer VSTO-Lösung .....	527
Der Datenaustausch bei geschlossenem Dokument .....	535
VSTO-Lösung von einem Dokument abtrennen .....	539
VSTO-Dokumentlösungen in Word 2007 .....	540
Verteilung von VSTO-Lösungen .....	541

VSTO COM-Add-Ins .....	543
Das Anlegen eines COM-Add-In Projekts .....	545
Word 2007 COM-Add-In-Benutzerschnittstellen .....	546
Das Beispiel ausprobieren .....	557
Zusammenfassung .....	558
<b>11 Andere Programme von Word aus steuern .....</b>	<b>559</b>
Microsoft Excel fernsteuern .....	560
Versteckte Excel-Instanz steuern .....	561
Berechnungen von Excel erledigen lassen .....	562
Microsoft PowerPoint fernsteuern .....	563
Versteckte PowerPoint-Instanz steuern .....	563
Text der Präsentation als Inhaltsstruktur in ein Dokument übernehmen .....	566
Microsoft Visio fernsteuern .....	568
Versteckte Visio-Instanz steuern .....	568
Microsoft Outlook fernsteuern .....	570
Versteckte Outlook-Instanz steuern .....	570
Kontakt als Empfängeradresse im Brief nutzen .....	572
Die aktuelle Datei über Outlook versenden .....	576
Microsoft Access fernsteuern .....	578
Auf Datenbanken zugreifen .....	580
Access-Datenbank ansprechen .....	580
Excel-Tabelle ansprechen .....	586
Zusammenfassung .....	588
<b>12 Eingebettete OLE-Objekte .....</b>	<b>589</b>
Excel-Tabellenobjekte .....	593
Excel-Diagramme .....	600
MS Graph-Diagramme .....	606
WordArt .....	610
ActiveX-Steuerelemente .....	615
Das Navigieren .....	618
Datengültigkeit prüfen .....	620
Steuerelement-Referenzen in VBA-Code .....	623
Zusammenfassung .....	624
<b>Teil D</b>	
<b>Optimierung der Benutzerschnittstelle .....</b>	<b>625</b>
<b>13 Anwendungsoptionen .....</b>	<b>627</b>
Dokumentvorlage <i>Normal.dot</i> konfigurieren .....	628
Vorlage »UrNormal.dot« erstellen .....	628
Benutzereinstellungen abspeichern .....	641
<i>SaveSetting</i> -Anweisung .....	643
<i>GetSetting</i> -Funktion .....	643



<i>PrivateProfileString</i> -Eigenschaft .....	644
Informationen mit MSXML schreiben und lesen .....	648
Dokumenteinstellungen abspeichern .....	649
Dokumenteigenschaften bearbeiten .....	650
Dokumentvariablen bearbeiten .....	653
Zusammenfassung .....	654
<b>14 Speicherort der Anpassungen .....</b>	<b>655</b>
Der Speicherort einer Anpassung .....	656
Auswahl des Speicherorts .....	657
Den Speicherort programmtechnisch festlegen .....	659
Kontext für COM-Add-Ins .....	661
Hierarchie der Anpassungen .....	662
Zusammenfassung .....	663
<b>15 Mit Dialogfeldern arbeiten .....</b>	<b>665</b>
Benutzerdefinierte Dialogfelder .....	666
UserForm in verschiedenen Projekten nutzen .....	667
Das <i>UserForm</i> -Objekt .....	668
Steuerelemente einbinden .....	676
Steuerelemente und ihre Besonderheiten .....	679
Anforderungen an ein Dialogfeld .....	682
Dialogfelder zur Laufzeit beeinflussen .....	684
Interne Dialogfelder .....	690
Dialogfelder »anzeigen«, »anzeigen und ausführen« oder »ausführen« .....	691
Dialogfelder konfigurieren, vorbelegen und auswerten .....	693
Übersicht über die in Word enthaltenen Dialogfelder .....	695
<i>FileDialog</i> -Objekt .....	696
Übersicht über die verschiedenen <i>FileDialog</i> -Typen .....	697
Dialogfelder »anzeigen« oder »anzeigen und ausführen« .....	698
Definieren von Dateiauswahlfilter .....	699
Anwendung des <i>FileDialog</i> -Typs <i>msoFileDialogOpen</i> .....	701
Anwendung des <i>FileDialog</i> -Typs <i>msoFileDialogSaveAs</i> .....	702
Anwendung des <i>FileDialog</i> -Typs <i>msoFileDialogFilePicker</i> .....	704
Anwendung des <i>FileDialog</i> -Typs <i>msoFileDialogFolderPicker</i> .....	704
Zusammenfassung .....	706
<b>16 Menüs und Symbolleisten .....</b>	<b>707</b>
CommandBars .....	708
Symbolschaltflächen .....	713
Symbolschaltflächen erstellen .....	714
Bearbeitungsfelder und Dropdownlisten .....	717
Symbolleisten erstellen .....	719
Zusammenfassung .....	721

<b>17 Multifunktionsleisten (Ribbons)</b>	<b>723</b>
Grundlagen	724
Die neue Terminologie	725
Die Rolle von <i>CommandBars</i> in Word 2007	726
Das Ribbon erweitern	727
Werkzeuge	727
Die Grundlagen	728
Vertieftes Wissen	733
Zusammenfassung	760
<b>18 Tastaturbelegungen</b>	<b>761</b>
Tastenbelegungen im Objektmodell	763
Bestehende Tastenbelegungen ermitteln	765
Tastenbelegungen eines Befehls ermitteln	770
Tastenkombinationen verwalten	771
Tastenbelegungen entfernen	771
Tastenkombinationen definieren	772
Tastenbelegung mit COM-Add-In verbinden	773
Ein Beispiel	774
Das Beispiel installieren	776
Zusammenfassung	777
<b>19 Aufgabenbereiche</b>	<b>779</b>
Allgemeines zum Aufgabenbereich oder, was steckt dahinter	780
Objektmodell-Schnittstellen für Work Panes	781
Registry-Einträge	782
Fehlermeldungen	783
Der programmtechnische Umgang mit Aufgabenbereichen (Task Panes)	784
Allgemeine VBA-Schnittstellen für die Arbeit mit Aufgabenbereichen	787
Bestimmte Aufgabenbereiche manipulieren	792
Zusammenfassung	806
<b>20 Interne Word-Befehle übersteuern</b>	<b>807</b>
Word-Befehl außer Kraft setzen	808
Zusammenfassung	812
<b>21 Zugriff auf den Visual Basic-Editor (VBE)</b>	<b>813</b>
Notwendige Verweise und Sicherheitseinstellungen	814
Der Visual Basic-Editor	815
Die VBA-Projekte	817
Übersicht über alle VBA-Projekte	818
Das aktive VBA-Projekt	819
Zugriff auf die in einem Projekt enthaltenen Komponenten	821

Auslesen des VBA-Codes von Komponenten .....	823
Allgemeine Zugriffe auf die Code-Zeilen .....	824
Zugriff auf den Deklarationsbereich .....	825
Zugriff auf die Code-Zeilen einzelner Prozeduren .....	826
Auflisten und Durchsuchen aller Projekte .....	829
Ersetzen und Entfernen von VBA-Code-Zeilen .....	831
Hinzufügen von Komponenten zu einem Projekt .....	834
Eine vorhandene Komponente in ein Projekt importieren .....	835
Eine neue Komponente einem Projekt hinzufügen .....	836
Entfernen von Komponenten aus einem Projekt .....	845
Anzeigen von dynamisch erzeugten UserForms .....	846
Verweise auf Bibliotheken und Dateien ermitteln und zur Laufzeit setzen .....	849
Übersicht über die gesetzten Verweise .....	849
Neuen Verweis setzen .....	850
Ungültige Verweise korrigieren bzw. entfernen .....	853
Zusammenfassung .....	857

## Teil E

<b>XML und Smart-Technologien .....</b>	<b>859</b>
---	------------

<b>22 Word 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 .....	884
Das XML-Dokument-Objektmodell (DOM) .....	884
XSLT .....	885
XML in Word ab Version 2003 .....	890
WordProcessingML .....	890
Benutzerdefiniertes XML .....	894
Die Word-Schemabibliothek .....	896
Transformationen und Lösungen (Solutions) .....	900
Schemas und Transformationen im Word-Objektmodell .....	906
WordProcessingML außerhalb von Word generieren .....	907
Zusammenfassung .....	908

<b>23</b>	<b>Smarttags</b>	<b>909</b>
	Die Arbeit mit Smarttags	910
	Smarttags aus dem Blinkwinkel des Benutzers	910
	Smarttag »Recognizers« und »Actions«	912
	Smarttag-Optionen und Ausnahme-Listen	913
	Die Entwicklung von Smarttags	914
	Unterschiede zwischen den Office-Versionen	915
	MOSTL-Smarttags entwickeln	915
	Ein Smarttag mit VB6 entwickeln	923
	VSTO-Smarttags	931
	Smarttags im Word-Objektmodell	933
	SmartTag-Objekte in Word-VBA verwenden	937
	Zusammenfassung	942
<b>24</b>	<b>Smart Documents</b>	<b>943</b>
	Was ist ein Smart Document?	944
	Ein MOSTL-Smart Document	946
	Das MOSTL-Beispiel installieren	947
	Eine benutzerfreundlichere Lösung	952
	So funktioniert's	958
	Eine Smart Document-DLL	970
	Mit VB6 eine Smart Document-DLL erstellen	970
	Das DLL-Beispiel installieren	971
	Wie es funktioniert: Das <i>SmartDocument</i> -Interface	976
	Die DLL in VB6 erstellen	997
	Das Erweiterungspaket-Manifest digital signieren	997
	Zusammenfassung	998
	<b>Teil F</b>	
	<b>Anhang</b>	<b>999</b>
<b>A</b>	<b>Namenskonventionen für Word-VBA</b>	<b>1001</b>
	Variablen	1002
	Benutzerdefiniertes Dialogfeld	1002
	Entwicklungsumgebung	1003
	Word-Objekte	1003
<b>B</b>	<b>Feldfunktionen</b>	<b>1005</b>
	Allgemeines zum Thema Feldfunktionen	1006
<b>c</b>	<b>Bezeichnungen von Symbolleisten</b>	<b>1013</b>

<b>D</b>	Startoptionen von Word .....	1025
	Die Befehlszeilenargumente von Word .....	1026
<b>E</b>	Bei Problemen – Word zurücksetzen .....	1029
<b>F</b>	Nützliche Links ins Internet .....	1033
<b>G</b>	Inhalt der CD-ROM .....	1035
	Die Beispieldateien der CD-ROM zum Buch .....	1036
	Zusatz-Software: Die Dateien im Anhang .....	1038
<b>H</b>	Über die Autoren .....	1041
	Verzeichnis zum Objektmodell .....	1045
	Stichwortverzeichnis .....	1053



# Vorwort

## **In diesem Vorwort:**

Für wen wurde das Buch geschrieben?	20
Word-Versionen	21
Wie ist das Buch aufgebaut?	22
Danksagung	24

Wir Autoren waren erstaunt, als uns der Verlag mitteilte, dass eine zweite Auflage des vorliegenden Buchs gedruckt werden soll. So machten wir uns daran, das Werk auf Vordermann zu bringen und unser Wissen über die neue Version, Word 2007, sowie einiges zum Thema Visual Studio 2005 Tools for Office einfließen zu lassen.

Zum Objektmodell gibt es eigentlich wenig Neues zu berichten. Zwar wurden Word neue Funktionalitäten hinzugefügt, die meisten davon finden aber im Objektmodell kein Gegenstück. Warum das?

Die Welt um Microsoft Office befindet sich im Umbruch. Vor gut fünfzehn Jahren offerierte Microsoft die Kernanwendungen (Word, Excel, PowerPoint und Access) gebündelt in einem Office-Paket, das dann zunehmend zum Standard in den meisten Unternehmen wurde.

Die weiterhin fallende Preisentwicklung ist nicht aufzuhalten. Durch die Angebote im Internet meinen viele Anwender, alles müsse kostenlos zu haben sein. So konkurrieren derzeit beispielsweise Linux, OpenOffice und die neuen Google-Anwendungen mit Microsoft Office. Zudem kaufen Großfirmen nur zögerlich Lizenzen für neue Versionen, denn die eingesetzten Produkte decken aktuelle Bedürfnisse bei Weitem ab.

Aus geschäftlicher Sicht musste also eine Entscheidung getroffen werden: entweder Office nicht weiterzuentwickeln oder neue Wege zu gehen. Als Konsequenz wird Office nun vermehrt an den Bedürfnissen von Großfirmen ausgerichtet. Mit 2007 Microsoft Office System wurde deshalb die Palette an Server-Produkten ausgebaut. Zudem nimmt XML eine zentrale Rolle ein, um die Arbeit mit Office-Dateien in Server-Umgebungen zu unterstützen. So wird beispielsweise die Benutzerschnittstelle, also die Multifunktionsleiste bzw. der »Ribbon«, mittels einer XML-Datei definiert und die alten binären Dateiformate werden ersetzt durch OpenXML als standardmäßiges Dateiformat. Folglich müssen darin enthaltene Objekte durch XML definiert werden... Und so gibt es für die neuen Diagramme und »SmartArts« keine automatisierbaren OLE-Schnittstellen. Alles wird auf den professionellen Entwickler ausgerichtet; die kleine Software-Firma, der unabhängige Berater und der Power-User bewegen sich in einem immer enger werdenden Umfeld.

## **Für wen wurde das Buch geschrieben?**

Obwohl sich die Ausrichtung von Office und Word verändert, sind unsere Beweggründe noch die gleichen. Unser gemeinsames Wissen zum Thema Word-Steuerung soll in schriftlicher Form festgehalten und weitergegeben werden, um allen zu helfen, die Word ihren Bedürfnissen entsprechend anpassen wollen. Die Möglichkeiten hierzu sind weiterhin vorhanden, es gilt nur, sie zu kennen und korrekt einzusetzen.

Nach wie vor richtet sich dieses Buch an ein breites Spektrum von Lesern – so breit wie der Funktionalitätsumfang von Word. Der eine freut sich über eine »Super-Schreibmaschine«, der kaufmännische Sachbearbeiter interessiert sich für Berichte, und der Redakteur will damit umfangreiche Handbücher verwalten. Anderswo in einer stillen Ecke sitzt der Autor, der darin das Werkzeug sieht, um seinen nächsten Bestseller zu verfassen. Daneben steht der professionelle Entwickler, der mit wenigen bis keinen Word-Kenntnissen diese Funktionalität den Bedürfnissen einer Großfirma anpassen muss. Dass eine 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 auseinandersetzen 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 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

Die erste Auflage dieses Buchs basiert auf Version 2003. Und abgesehen von neuen Funktionalitäten gelten die meisten Angaben sowohl für die Vorgänger-Versionen Word 2002 und Word 2000 als auch für Word 2007. Änderungen der aktuellen Version fanden hauptsächlich in der Benutzerschnittstelle und im Dateiformat statt. Intern ist weiterhin der gleiche Kerncode im Einsatz und es wird mit dem binären Dateiformat gearbeitet. Folglich hat sich im Objektmodell nichts Wesentliches geändert.

Diese zweite Auflage wurde um die neuen Funktionalitäten Building Blocks sowie Content Controls erweitert. Zudem enthält sie ein neues Kapitel über die Anpassung der neuen Benutzerschnittstelle, der Multifunktionsleiste.



2007

Alle Codebeispiele wurden in Word 2007 getestet, weitgehend für gut befunden und belassen; auf Rückwärtskompatibilität hat das Word-Team geachtet. Wo Bezug auf die Benutzerschnittstelle genommen wird, befinden sich Angaben zu den neuen sowie alten Menüfolgen. Um gezielt Word 2007-spezifische Informationen zu finden, halten Sie bitte nach dem nebenstehend dargestellten Symbol Ausschau.

Die C#-Beispiele der ersten Auflage, in Visual Studio 2003 geschrieben, wurden in das Visual Studio 2005-Format konvertiert und funktionieren, wie der VBA-Code, weiterhin. Allerdings passt Visual Studio die Verweise auf die Office- und Word-Objektmodelle nicht automatisch an. Falls Sie mit Office 2007 arbeiten, werden Sie die bestehenden Referenzen zu den 2003-Versionen löschen und neu hinzufügen müssen (siehe dazu Kapitel 9).

Die grundlegenden Aussagen dieses Werks können auch auf Word 97 angewendet werden. Hier gilt es jedoch zu beachten, dass unterdessen die fünfte 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. Die nachfolgenden Word-Versionen enthalten VBA 6.

### Die Zukunft von VBA

Die Rückwärtskompatibilität von VBA ist für die nächste Version von Word (»Office 14« – die Zahl »13« wird übersprungen) weiterhin gewährleistet. Somit wird das aus dem vorliegenden Buch Erlernte auch für diese Version seine Gültigkeit haben. Auf Grund der großen Zahl von VBA-Lösungen, die weltweit in Firmen eingesetzt werden, ist nicht anzunehmen, dass die Unterstützung von VBA in nächster Zukunft abgeschafft wird. Stattdessen werden die Forderungen der .NET-Entwickler vorläufig auf anderen Wegen zufrieden gestellt.

### VSTO

Auffällig ist das Bemühen von Microsoft, die Fernsteuerung von Word (sowie den übrigen Office-Anwendungen) für die meisten Entwickler überflüssig zu machen. Dank der neuen OpenXML-Dateiformate ist es nun möglich, Dokumente zu erstellen, bearbeiten und lesen, ohne Word zu starten. Ja, Word muss nicht einmal installiert sein. Nur der Entwickler, der interaktiv mit dem Benutzer arbeitet, muss sich um die Fernsteuerung kümmern. Diese Aufgabe wird immer mehr dem Werkzeug VSTO (Visual Studio Tools for Office) zugeteilt, um den Umgang mit den COM-Anwendungen für den .NET Entwickler zu »entschärfen«. Aus diesem Grund wird VSTO mehr Platz gewidmet als in der ersten Auflage.

## Wie ist das Buch aufgebaut?

Dieses Buch besteht aus fünf 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 A** – Eine Einführung in die Welt der Makros und VBA. Er vermittelt das grundlegende Wissen zu den Möglichkeiten von Makros und stellt die Werkzeuge der eigentlichen Programmierung – den Visual Basic-Editor – vor. Eine Zusammenfassung zu den VBA-Grundlagen, erste allgemeine Beispiele und das Einbinden von Windows-API runden diesen Teil ab.

**Teil B** – 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-APIs veranschaulichen. Anhand dieser Kenntnisse sollte sich der .NET-Programmierer Zugang zum gesamten Objektmodell verschaffen können.



2007

Die neuen Funktionalitäten, Bausteine, Inhaltssteuerelemente und ihre Objektmodelle werden in diesem Teil eingehend vorgestellt.

**Teil C** – 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 2005 vorgestellt.

**Teil D** – 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, das Anpassen von Symbolleisten bzw. der Multifunktionsleiste und Tastaturkombinationen.

Teil E – 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.

## Die CD-ROM zum Buch

Dem Buch ist eine CD-ROM beigelegt. Diese enthält unter anderem alle Kapitel im PDF-Format. Damit steht das Buch unterwegs als elektronisches Nachschlagewerk zur Verfügung.

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 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*.

Damit in der zweiten Auflage Platz für die neuen Themen geschaffen werden konnte, mussten einige Seiten entfernt werden. In erster Linie betrifft dies die praxisbezogenen Lösungsbeispiele, die in der ersten Auflage im Teil V ausgeliefert wurden. Damit diese nützlichen Informationen weiterhin zur Verfügung stehen, sind die entsprechenden Seiten in elektronischer Form auf der Buch-CD als Bonusteil enthalten.

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 zusammengefasst.

# Danksagung

An dieser Stelle möchten wir uns ganz besonders nochmals bei Elisabeth Wilke-Thissen bedanken. Sie hat sich erneut ohne Zögern bereit erklärt, die neuen Texte in eine lesbare, deutsche Fassung zu bringen. Lisa wurde im April 2003 in das MVP-Programm (Word) aufgenommen, ist Co-Autorin des Buches »Microsoft Excel – Die Expertentipps« sowie Co-Übersetzerin von Stephanie Kriegers Basiswerk »Advanced Microsoft Office Documents 2007 Edition« (»Office 2007 – Das Profibuch«).

Bedanken möchten wir uns auch bei Markus Hahner ([www.hahner.de](http://www.hahner.de)), seit vielen Jahren Fachredakteur und EDV-Autor. Als wir ihn bei einem Seminar zu Office 2007 als Trainer kennen lernten, sagte er spontan zu, alle im Teil B dieses Buches enthaltenen Codebeispiele unter Word 2007 zu testen. Ohne diese Leistung wäre eine rechtzeitige Veröffentlichung dieses Werkes unmöglich gewesen.

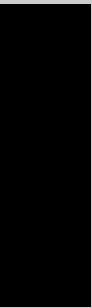
*Das Autorenteam, im November 2007*

# Teil A

## Grundlagen der Arbeit mit VBA

### In diesem Teil:

<b>Kapitel 1</b>	Word-Makros	27
<b>Kapitel 2</b>	VBA-Grundlagen	63
<b>Kapitel 3</b>	Windows-APIs in VBA nutzen	125



## Kapitel 1

# Word-Makros

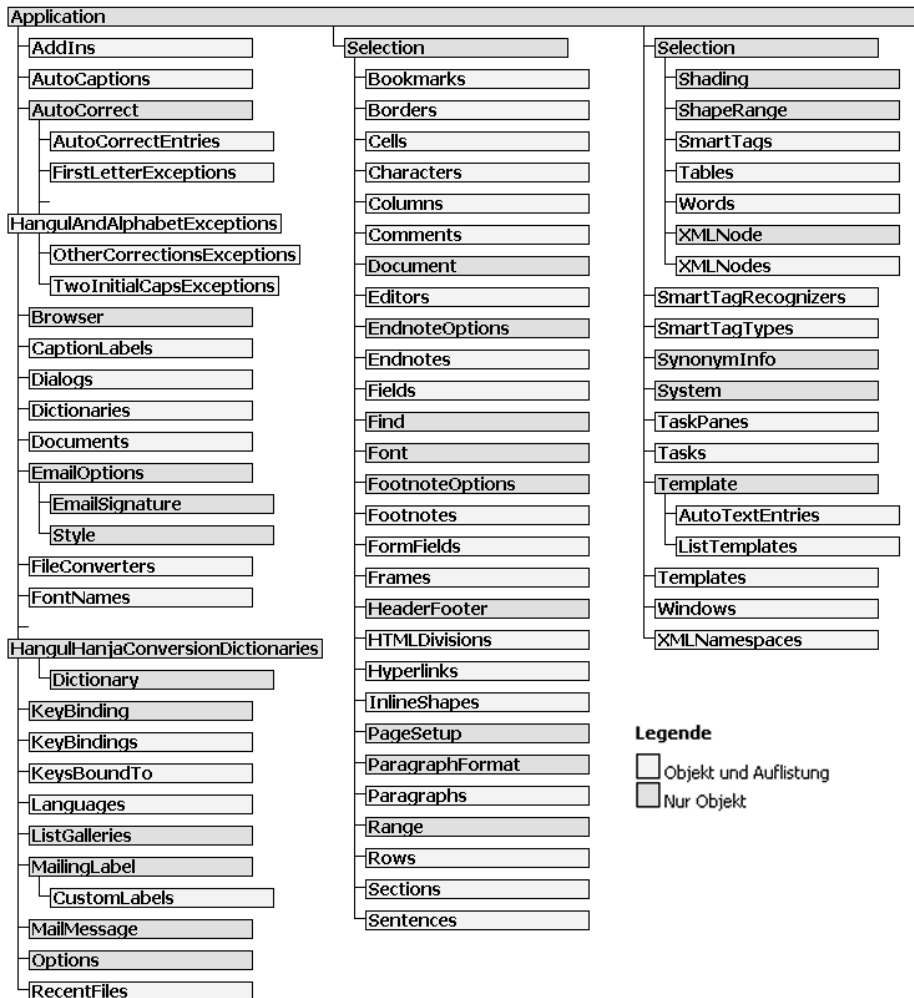
**In diesem Kapitel:**

Aller Anfang ist schwer	28
Programmierhilfen	29
Makros in die Benutzerschnittstelle einbinden und verwalten	47
Makrosicherheit	52
Zusammenfassung	61

# Aller Anfang ist schwer

Jede 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. Das Diagramm in Abbildung 1.1 zeigt lediglich die oberste Ebene des Objektmodells für Word 2003 an. Das Word 2007-Objektmodell ist noch umfangreicher und auf der Microsoft-Website unter <http://msdn2.microsoft.com/en-us/library/bb288731.aspx> einzusehen.

Abbildg. 1.1 Die oberste Ebene des Objektmodells (Word 2003)



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.



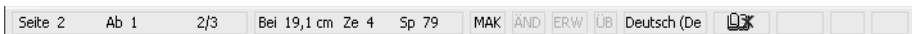
# Programmierhilfen

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 nachgeschlagen (siehe dazu den Abschnitt »Die Objektmodell-Hilfe – eine versteckte Schatzkammer« weiter hinten in diesem Kapitel). Danach wird der bereinigte Code in das effektive Makro oder in die Prozedur eingebaut.

## Den Makrorekorder einsetzen

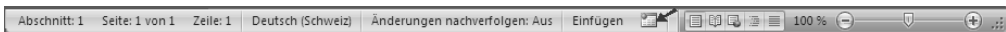
Den Makrorekorder rufen Sie über die Befehlsfolge *Extras/Makro/Aufzeichnen* oder mit einem Doppelklick auf das Kästchen *MAK* in der Statusleiste auf (Abbildung 1.1).

Abbildg. 1.2 Die Word 2003-Statusleiste mit aktivierter Makrorekorder-Funktionalität



In Word 2007 finden Sie den Befehl *Makros* in der Multifunktionsleiste auf der Registerkarte *Ansicht*. Mit einem Klick der rechten Maustaste auf die Symbolschaltfläche *Makros* kann der entsprechende Befehl über das daraufhin geöffnete Kontextmenü auch in die Symbolleiste für den Schnellzugriff eingefügt und von dort direkt aufgerufen werden.

Abbildg. 1.3 Die Word 2007-Statusleiste mit aktivierter Makrorekorder-Funktionalität



In dem nun eingeblendeten Dialogfeld *Makro aufzeichnen* (Abbildung 1.4) 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 Symbolschaltfläche oder einer Tastenkombination in der Benutzeroberfläche zuzuweisen.

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 Sie in Kapitel 14 nachschlagen.

### 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. 1.4 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).



2007

In Word 2007 wird die Makroaufzeichnung über die Schaltfläche *Makros* auf der Registerkarte *Ansicht* oder über die Statusleiste gesteuert.

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 Dialogfelds wird nicht aufgezeichnet, sondern das Endresultat der darin vorgenommenen Einstellungen. (Beispiel: Sie blenden das Dialogfeld *Datei öffnen* ein und wählen ein Dokument. Der Makrorekorder zeichnet das Öffnen dieses Dokuments auf, nicht aber das Einblenden des Dialogfeldes.) Wie Sie ein Dialogfeld einblenden lassen, wird in Kapitel 15 vorgestellt.
- 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 nicht aufgezeichnet, da sie in der Excel-Umgebung vorgenommen werden. Um über den Umgang mit eingefügten Objekten zu lesen, schlagen Sie bitte in Kapitel 12 nach.
- 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] + [F4]** vorgenommen werden. Die Automatisierung anderer Office-Anwendungen wird in Kapitel 11 diskutiert.
- 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 die **[Esc]**-Taste.
- Um eine Markierung oder Handlung vorzunehmen, die mit der Tastatur oder einem Menübefehl nicht ausführbar 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.

- Wegen der neuen Benutzerschnittstelle musste der Makrorekorder angepasst werden. Als Folge werden verschiedene in Word 2007 enthaltene Funktionalitäten nicht aufgezeichnet. Für einige Teile hat Microsoft sogar keine Objektmodellschnittstelle zur Verfügung gestellt, was bedeutet, dass auch dafür keine Aufzeichnung stattfindet. Dies betrifft hauptsächlich Objekte, die mit den neuen Grafiken in Office zusammenhängen, wie Diagramme und SmartArt. Falls Sie ein vom Objektmodell nicht unterstütztes Element in einem Dokument erstellen müssen, geht es nur über das OpenXML-Format. Das Thema OpenXML wird in diesem Buch nicht ausführlich beschrieben. Zusätzliche Informationen finden Sie jedoch in Kapitel 22.

Mit der Schaltfläche *Aufzeichnung beenden* wird die Aufzeichnung beendet. Wechseln Sie zum Visual Basic-Editor (`(Alt) + (F11)`), um das Ergebnis im Modul *NewMacros* anzuschauen und anzupassen.

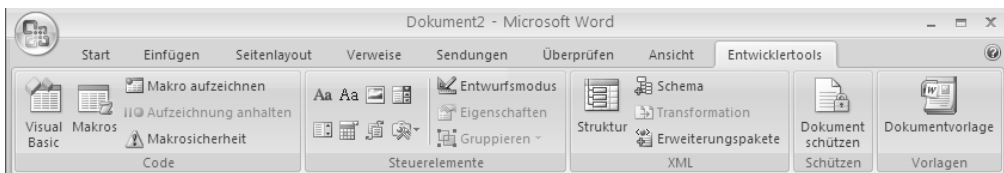
### HINWEIS

In den obigen Anweisungen haben wir die standardmäßig zur Verfügung stehenden Befehlsfolgen angegeben. Zusätzlich gibt es jedoch noch eine Registerkarte *Entwicklertools* mit einer erweiterten Auswahl von Befehlen (Abbildung 1.5). Um diese einzublenden, öffnen Sie über die *Office*-Schaltfläche das Dialogfeld *Word-Optionen*. In der Kategorie *Häufig verwendet* aktivieren Sie das Kontrollkästchen *Entwicklerregisterkarte in der Multifunktionsleiste anzeigen*.



Abbildg. 1.5

Die Registerkarte *Entwicklertools*



## 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 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 leeren 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 1.6). Damit das aufgezeichnete Makro wunschgemäß arbeitet, muss der Anwender die Einfügemarke vor dem Ausführen unbedingt in der ersten Zelle einer Tabelle 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. 1.6** 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
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
```

## HINWEIS Für .NET-Entwickler



Die Dokumentation zum Word-Objektmodell ist VBA-orientiert, weil VBA die Office-Programmiersprache ist. Zudem widerspiegelt das Objektmodell eher die Benutzerschnittstelle und Words Arbeitsweise und kann daher etwas »fremd« vorkommen. Im Abschnitt »Die Objektmodell-Hilfe – eine versteckte Schatzkammer« weiter hinten in diesem Kapitel werden wir etwas näher darauf eingehen. Aber genau weil dem so ist, kann auch Ihnen der Makrorekorder behilflich sein.

Vergleichen Sie Listing 1.2. Diese Prozedur erfordert lediglich, dass sich die Einfügemarke innerhalb der Tabelle befindet und erzeugt keine Fehlermeldung, falls dies nicht zutrifft. 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 1.7 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 oder sind keine Dokumente vorhanden, wird eine entsprechende Meldung angezeigt und die Ausführung abgebrochen. Nach erfolgreicher Ausführung wird ebenfalls eine Meldung eingeblendet. Am Schluss wird das `wdApp`-Objekt wieder freigestellt.

---

**HINWEIS** Mehr zum Thema Automatisierung von Word aus der .NET-Umgebung erfahren Sie in Kapitel 10.

---

**Abbildg. 1.7** Die oberste Schaltfläche führt den C#-Code von 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.GetObject("Word.Application");
        if (wdApp == null)
        {
            MessageBox.Show("Word läuft nicht.");
            return;
        }
        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
        MessageBox.Show("Fertig!");
        wdApp.Activate();
        wdMarshal.ReleaseComObject(wdApp);
        wdApp = null;
    }
    catch (System.Runtime.InteropServices.COMException ex)
    {
        MessageBox.Show(ex.Message);
    }
}
```

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 aufgezeichneten 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 diesem Kapitel die Programmierhilfen vor. Das Kapitel 2 enthält 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 sowie die Datei *Kap01\_CS.zip* mit dem C#-Beispiel finden Sie auf der CD-ROM zu diesem Buch im Ordner *\Beispiele\Kap01*.

## Unterstützende Werkzeuge des VB-Editors

In allen Office-Anwendungen außer InfoPath 2007 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 der Umgang mit der VB-Editor-Umgebung nicht schwer.

Wenn Sie vorhaben, Word mit einer anderen Programmiersprache zu automatisieren – beispielsweise aus dem klassischen Visual Basic 6 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 in allen Word-Versionen durch Drücken der Tastenkombination **Alt + F11** auf. Er ist auch erreichbar über die Schaltfläche *Visual Basic* auf der Word 2007-Registerkarte *Entwicklertools* bzw. über die Menüfolge *Extras/Makros/Visual Basic-Editor* in früheren Word-Versionen.

Alle Teile und Befehle des VB-Editors werden in der Hilfedatei zur »Microsoft Visual Basic Documentation« erläutert. In diesem Abschnitt zeigen wir Werkzeuge auf, die bei der Arbeit mit dem Word-Objektmodell behilflich sind.

### 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 21 vorgestellt.

## Code speichern



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.

Für große Projekte oder zwecks Code-Austausch ist es oft ratsam, Sicherheitskopien zu erstellen. Dies geht über den Menübefehl *Datei/Datei exportieren* des VB-Editors. Im Gegensatz zu den in Abschnitt »Makros kopieren« 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). Die Vorteile reiner Text-Dateien liegen auf der Hand: Es besteht keine Gefahr des Verlusts durch Dokumentbeschädigung, und sie können jederzeit und überall geöffnet werden.

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



**PROFITIPP**

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.)

## IntelliSense

Im Abschnitt »Den Makrorekorder einsetzen« 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 1.8 entspricht dem Listing aus jenem Abschnitt. 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?

**Abbildg. 1.8** VBA im Code-Fenster des VB-Editors

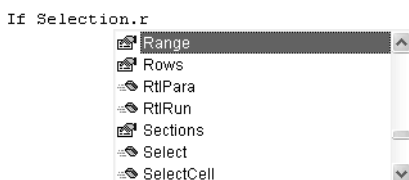
```
Sub TabelleVorbereiten2()
    Dim tbl As Word.Table
    Dim rw As Word.Row



    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
```

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 1.9 dargestellt.

**Abbildg. 1.9** Die IntelliSense-Funktion 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 - oder -Taste wird der Vorschlag übernommen.

**TIPP** Für alle IntelliSense-Listen gilt: Drücken Sie , um die Liste ungenutzt zu schließen.

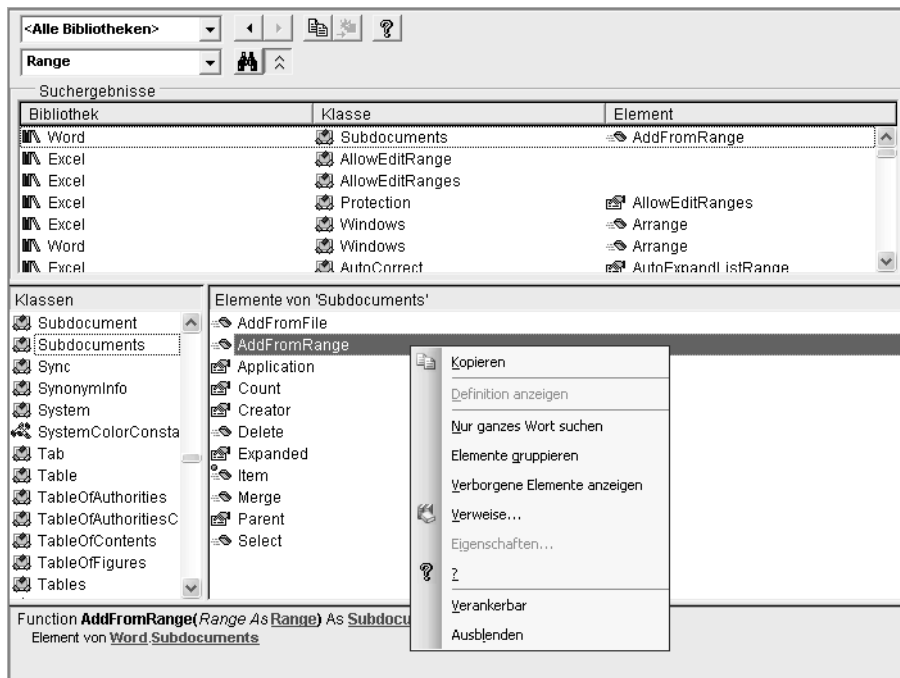
## Wo alle Fäden zusammenlaufen: Der Objektkatalog



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 .


Wie in Abbildung 1.10 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 Dropdownlisten 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. 1.10 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 9.

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.)

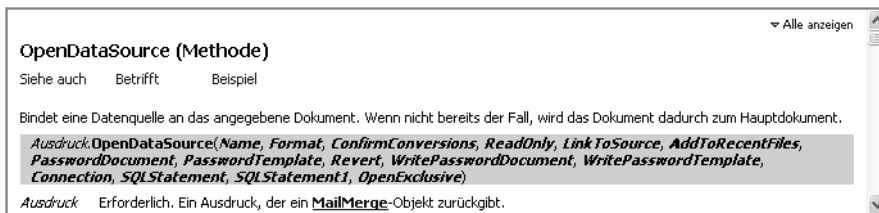
Die Suchergebnisse erscheinen im zweiten Teil des Objektkatalogs. Alle Elemente, in denen die gesuchte Zeichenkette enthalten ist, werden rechts in der dritten Spalte aufgelistet. Das Objekt (Klasse), dem sie angehören, befindet sich in der Spalte links daneben, und die zugehörige Anwendung (Bibliothek) 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 1.10, die Prozedursyntax. Die unterstrichenen Begriffe dienen jeweils als Hyperlink, worüber weitere Informationen in der Liste *Elemente* angezeigt werden.

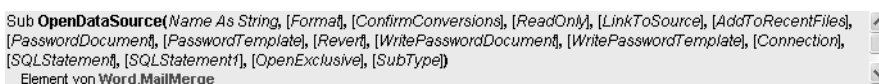
Sie gelangen zur Hilfe für ein markiertes Element durch Drücken der Taste  oder über das Kontextmenü.

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 1.11 mit der Abbildung 1.12. 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. 1.11** Die Methodenunterschrift für die *OpenDataSource*-Methode laut Hilfedokumentation



**Abbildg. 1.12** 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-Programmierregeln 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 1.13 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

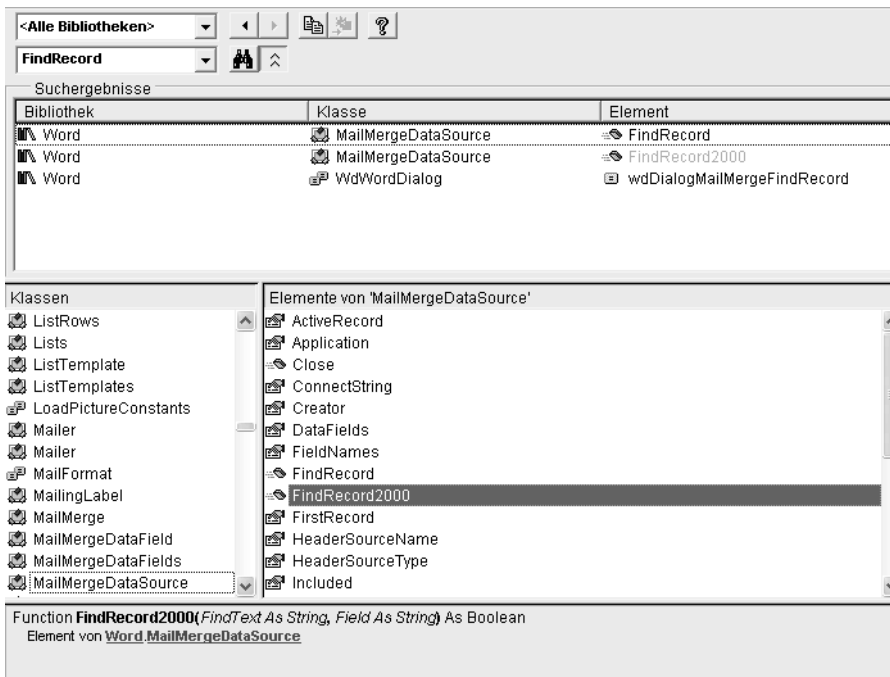
Eigentlich befasst sich dieses Buch nicht mit der Theorie der Programmierung. Aber auch auf dieser Ebene sind Lehren aus der Fehlüberlegung, die hinter `FindRecord2000` steht, zu ziehen. Deshalb wird hier etwas ausführlicher beschrieben, was hinter den verborgenen Einträgen steckt und warum sie problematisch 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 nicht auf die Idee kam, `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. 1.13 In Word 2002 gibt es die Methoden *FindRecord* sowie *FindRecord2000*. Letztere ist normalerweise verborgen.



## Die Objektmodell-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 zwölf Jahre regelrecht umgekrempelt. Dies gilt gleichermaßen für deren Inhalt sowie Struktur. Leider können wir nicht behaupten, dass diese Änderungen immer 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 Objektmodell-Hilfe-Schnittstellen in Word 2000 bis 2007 vorstellen. Insbesondere werden Möglichkeiten, die Hilfe aufzurufen und Informationen zu finden, diskutiert.

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 vier Versionen jedoch grundsätzlich unterscheidet:

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

Das VBA-Hilfenfenster 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 ausgewähltem 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 freistehend 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.

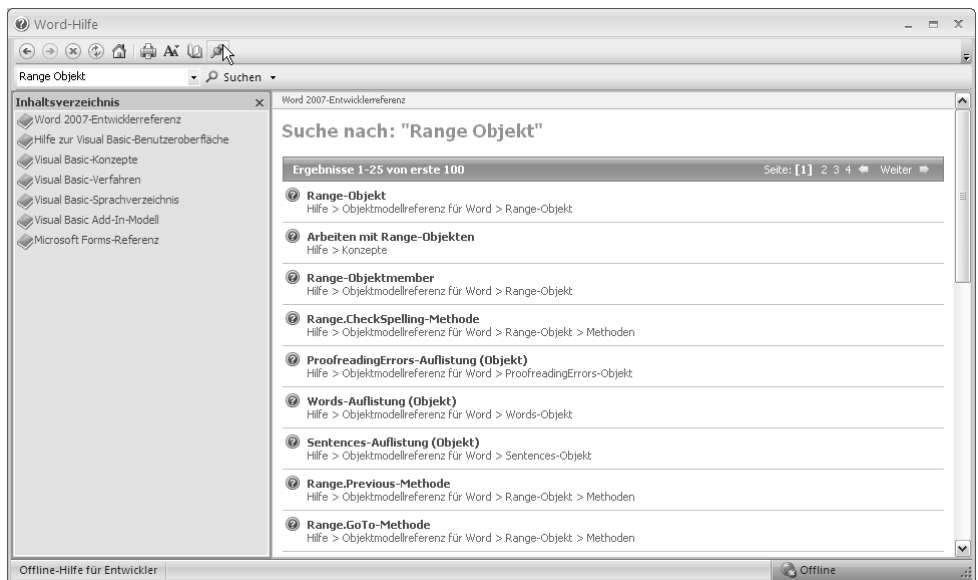
Bei Auswahl eines Themas wird ein separates Hilfefenster geöffnet, das entweder am rechten Rand neben dem Aufgabenbereich eingefügt wird (der VB-Editor wird entsprechend schmaler) oder über dem anderen Fenster liegt.

- **Word 2007:** Der Aufgabenbereich ist wieder verschwunden. Es wird ein unabhängiges Fenster eingeblendet, wie in Abbildung 1.14 ersichtlich. Der gesamte Helpbereich für Word 2007 ist durch die Dropdownliste *Suchen* erreichbar. Im Gegensatz zur Hilfe für die Word-Anwendung befinden sich alle Informationen zum Objektmodell lokal auf dem Rechner statt teilweise im Internet.



Abbildg. 1.14

Das Office 2007-Hilfefenster



Das Hilfefenster können Sie mit der in der Abbildung hervorgehobenen Schaltfläche vor allen übrigen »festnageln«. Das Ergebnis einer Suche sehen Sie im unteren Fensterteil, rechts.

Auffallend ist, dass die Objektmodell-Hilfe neu nach dem Muster des Visual Studio organisiert wurde. Die Abbildung 1.15 veranschaulicht dies anhand der Auflistung aller Methoden und Eigenschaften des Range-Objekts.

Abbildg. 1.15

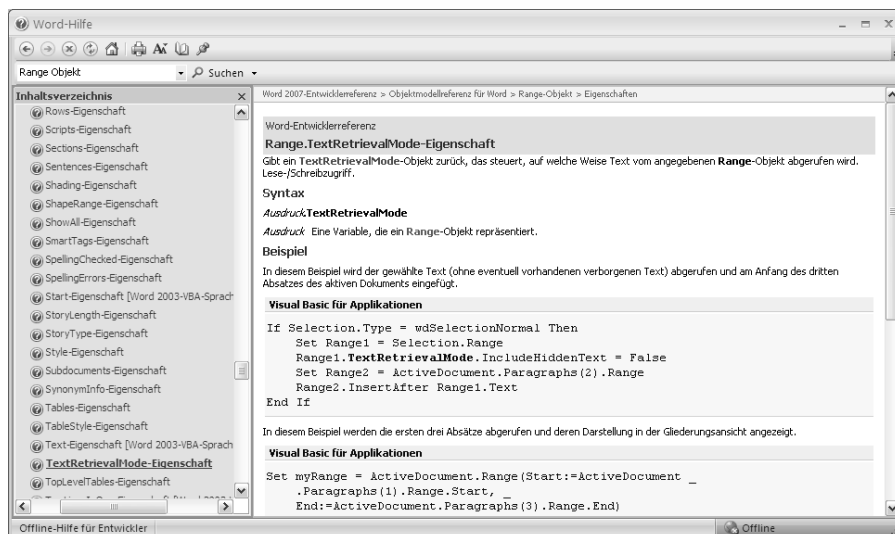
Auflistung aller Methoden und Eigenschaften eines Objekts



Eine kurze Beschreibung steht neben den meisten Einträgen. Per Klick auf einen Eintrag wird die Hilfe zum Thema eingeblendet (Abbildung 1.16). Falls das Inhaltsverzeichnis (links) eingeblendet ist, haben Sie auch dort Zugriff auf die gleichen Begriffe wie in Abbildung 1.15 und damit immer eine Übersicht zur Hand.

Abbildg. 1.16

Die Hilfe für ein bestimmtes Objektelement




**HINWEIS**

Wenn Sie in der .NET-Umgebung entwickeln, stehen die VBA-Hilfdateien über die üblichen .NET-Hilfeschnittstellen 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 gesamte Entwickler-Referenz für Word 2007 steht in englischer Sprache auf MSDN zur Verfügung (<http://msdn2.microsoft.com/en-us/library/bb244391.aspx>).

Die Entwicklerreferenz für Word 2003 steht ebenfalls in englischer Sprache auf MSDN zum Herunterladen bereit (<http://www.microsoft.com/downloads/details.aspx?familyid=179bee82-e6e6-4b78-aff9-9a541167541f&displaylang=en>).

Die Onlineversion derselben befindet sich auf [http://msdn2.microsoft.com/en-us/library/aa272078\(office.11\).aspx](http://msdn2.microsoft.com/en-us/library/aa272078(office.11).aspx).

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 führt das Blättern im Indexverzeichnis 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 Abschnitt 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 in Word 2002 sowie 2003 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. In diesen Versionen haben Sie aber immer die Möglichkeit, die Hilfdateien direkt zu öffnen. (Das ist in Office 2007 nicht mehr möglich, aber das beschriebene Problem wurde dort aufgehoben.) Noch besser: Diese verfügen über die Registerkarten *Inhalt* und *Suchen*, welche in Office 2003 nicht mehr angezeigt werden (Abbildung 1.17). Hilfdateien 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 1.1 finden Sie eine Namensliste der Office-Hilfdateien.

**Tabelle 1.1** Die Hilfdateien der Microsoft Office 2003-Anwendungen

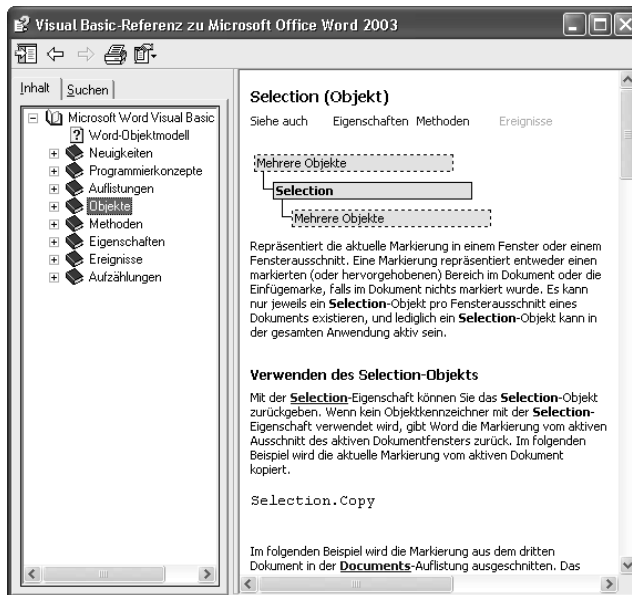
Anwendung	Hilfdatei
Microsoft Access	ACMAIN11.CHM (Benutzerschnittstelle) VBAAC10.CHM (Entwicklerreferenz)
Microsoft Graph	GRAPH10.CHM (Benutzerschnittstelle) VBAGR10.CHM (Entwicklerreferenz)
InfoPath	IPMAIN11.CHM (Benutzerschnittstelle) INFMAIN.CHM (Benutzerschnittstelle) INFREF.CHM (Entwicklerreferenz)
OLAP	MCE.CHM
Document Imaging	MSPHELP.CHM
MS Query	MSQRY32.CHM



**Tabelle 1.1** Die Hilfedateien der Microsoft Office 2003-Anwendungen (Fortsetzung)

Anwendung	Hilfedatei
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)
Microsoft Office Word	<i>WDMAIN11.CHM</i> (Benutzerschnittstelle) <i>VBAWD10.CHM</i> (Entwicklerreferenz)
Microsoft Office Excel	<i>XLMAIN11.CHM</i> (Benutzerschnittstelle) <i>VBAXL10.CHM</i> (Entwicklerreferenz) <i>XLADDIN.CHM</i> (Entwicklerreferenz) <i>XLMACRO.CHM</i> (Entwicklerreferenz)
Microsoft XML Parser	<i>XMLSDK5.CHM</i> (Entwicklerreferenz)
Hilfedateien für die Office-Programmierung befinden sich standardmäßig unter <i>C:\Programme\Gemeinsame Dateien\Microsoft Shared\VBA\VBA6\1031</i>	
Visual Basic Umgebung (enthält eine Schnittstelle zu den folgenden Bibliotheken)	<i>VBUI6.CHF</i>
Formulare (UserForms)	<i>FM20.CHF</i>
Visual Basic allgemein (Theorie)	<i>VBCN6.CHF</i>
Visual Basic allgemein (Begriffslexikon)	<i>VBENDF98.CHM</i>
Visual Basic allgemein (Debugging)	<i>VBHW6.CHF</i>
Visual Basic allgemein (Objektmodell)	<i>VBLR6.CHF</i>
Visual Basic Editor	<i>VBOB6.CHF</i>

Abbildg. 1.17 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 1.18 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. 1.18 Sie finden auf der Hilfeseite nützliche Links zu verwandten Themen.



# Makros in die Benutzerschnittstelle einbinden und verwalten

Wie im Abschnitt »Den Makrorekorder einsetzen« weiter vorne 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 Symbolschaltfläche zuweisen



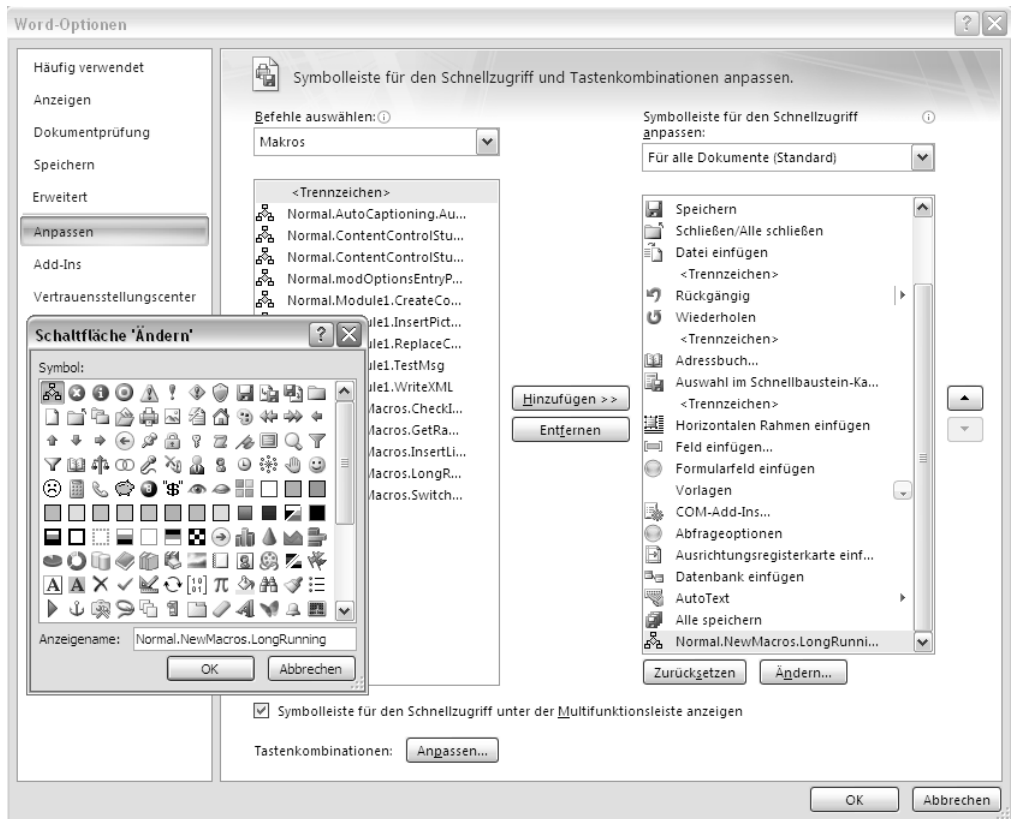
2007

Um in Word 2007 ein Makro einer Symbolschaltfläche in der Symbolleiste für den Schnellzugriff zuzuweisen, führen Sie die folgenden Schritte durch (die Abbildung 1.19 veranschaulicht diesen Vorgang):

1. Klicken Sie zunächst auf die *Office*-Schaltfläche und anschließend auf die Schaltfläche *Word-Optionen*.
2. Öffnen Sie die Kategorie *Anpassen*.
3. Aus der Dropdownliste *Symbolleiste für den Schnellzugriff anpassen* wählen Sie den gewünschten Speicherort für die Anpassung (siehe dazu Kapitel 14).
4. In der Dropdownliste *Befehle auswählen* wählen Sie den Eintrag *Makros* aus.
5. Klicken Sie zunächst auf den Namen des Makros, das Sie der Symbolleiste für den Schnellzugriff hinzufügen möchten, und anschließend auf die Schaltfläche *Hinzufügen*. Die Makrobezeichnung erscheint in der Liste rechts.
6. Um die Beschriftung oder das Symbol der Schaltfläche zu ändern, klicken Sie auf die Schaltfläche *Ändern*.

### HINWEIS

Nur die Symbolleiste für den Schnellzugriff kann in Word 2007 vom Benutzer in der Word-Umgebung angepasst werden. Dazu stehen lediglich die Symbole im Dialogfeld *Schaltfläche 'Ändern'* zur Verfügung. Es ist nicht möglich, diese anzupassen oder eine eigene Grafik dafür zu verwenden. Wie die Multifunktionsleiste ergänzt wird, erfahren Sie in Kapitel 17.

**Abbildg. 1.19** Makros der Symbolleiste für den Schnellzugriff zufügen und die Symbolschaltfläche anpassen



In allen vorherigen Versionen von Word gehen Sie wie folgt vor, um ein Makro einer Symbolleiste oder einem Menü zuzuweisen:

1. Blenden Sie das Dialogfeld *Anpassen* über die Befehlsfolge *Extras/Anpassen* ein.
2. Wechseln Sie zur Registerkarte *Befehle*.
3. Legen Sie den Speicherort fest (siehe dazu Kapitel 14).
4. Wählen Sie die Kategorie *Makros*.
5. Wählen Sie den gewünschten Eintrag in der rechten Liste aus (Abbildung 1.21) 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. Klicken Sie mit der rechten Maustaste auf die neu erstellte Schaltfläche, öffnet sich ein Kontextmenü, 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), *Schaltflächen-symbol 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 *Schaltflächen-symbol und Text*.



**HINWEIS** In Kapitel 16 wird erläutert, wie Symbolleisten und deren Schaltflächen durch das Objektmodell erstellt und verwaltet werden.

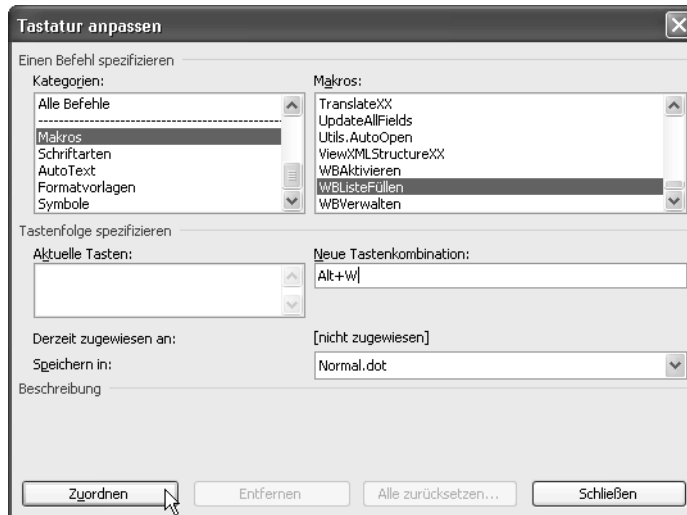
## Makro einer Tastaturkombination zuweisen

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

1. Blenden Sie das Dialogfeld *Anpassen* über die Befehlsfolge *Extras/Anpassen* ein.
2. Klicken Sie auf die Schaltfläche *Tastatur* und legen Sie im geöffneten Dialogfeld *Tastatur anpassen* (Abbildung 1.20) den Speicherort fest (siehe dazu Kapitel 14).
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. 1.20

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



Das Anpassen der Tastatur ist in Word 2007 auch weiterhin möglich, und zwar über die Schaltfläche *Anpassen* in der Kategorie *Anpassen* des Dialogfeldes *Word-Optionen* (Abbildung 1.19). Die Liste *Kategorien* im Dialogfeld *Tastatur anpassen* enthält, ähnlich wie die Liste *Befehle auswählen* der Kategorie *Anpassen*, Einträge aus der Multifunktionsleiste.

**HINWEIS** Wie Tastenkombinationen durch das Objektmodell definiert werden, wird in Kapitel 18 erklärt.

### Rangfolge 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?

Daran hat auch Microsoft gedacht und die unten stehende Reihenfolge für Word festgelegt:

- Makros werden durch *ProjektName.ModulName.MakroName* identifiziert, wie auch die Liste *Befehle* in Abbildung 1.21 zeigt. Es ist deshalb wichtig, jedes VBA-Projekt eindeutig zu benennen (siehe dazu Kapitel 9). 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. 1.21** Durch Ziehen eines Eintrags aus der Liste *Befehle* das Makro einer Symbolleiste oder einem Menü hinzufügen



**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:

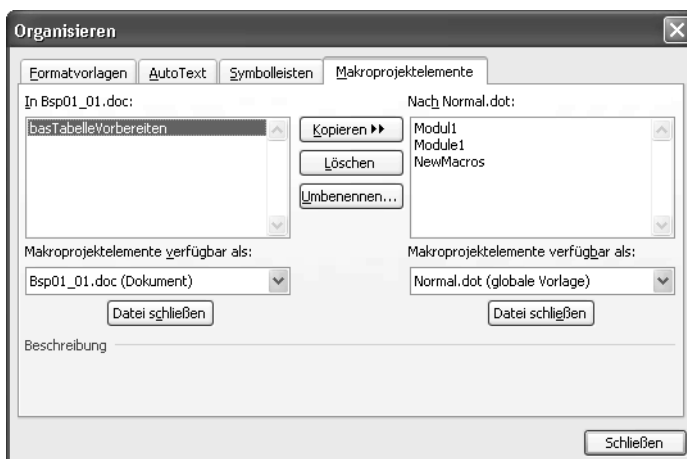
- Prozeduren, die als *Private* bezeichnet sind (mehr zu *Private* lesen Sie in Kapitel 2)
- Prozeduren, für die Argumente definiert sind
- Prozeduren, die einen Wert zurückgeben (Funktionen)
- Prozeduren, die sich in einem Modul befinden, das die Anweisung *Option Private Module* enthält
- Prozeduren, die in einem Klassenmodul oder UserForm-Modul stehen

## 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, oder mittels einer Sicherheitskopie, wie im Abschnitt »Code speichern« beschrieben, ausgetauscht 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 welchem einzelne Code-Module (oder auch Formatvorlagen, und in Word 2003 und früher Symbolleisten und AutoTexte) zwischen Word-Dateien kopiert werden können: das Dialogfeld *Organisieren* (siehe Abbildung 1.).

**Abbildg. 1.22** Makros mit dem Dialogfeld *Organisieren* problemlos verwalten und zwischen Word-Dateien kopieren



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 austauschen, 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.



*Organisieren* gibt es in Word 2007 weiterhin, allerdings ist die Funktionalität auf Makroprojekte sowie Formatvorlagen beschränkt. Das Dialogfeld öffnen Sie über die Schaltfläche *Dokumentvorlage* auf der Registerkarte *Entwicklertools* der Multifunktionsleiste (um die Registerkarte *Entwicklertools* sichtbar zu machen, siehe den Abschnitt »Den Makrorekorder einsetzen« weiter vorne in diesem Kapitel).

#### 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 21 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 konnte der Anwender festlegen, ob die Makros innerhalb von Dokumenten überhaupt aktiviert und ausgeführt werden, oder nicht.

Zehn Jahre später, in Word 2007, wurden die Sicherheitsmaßnahmen nochmals verfeinert und angepasst. Damit muss sich sowohl der professionelle Entwickler als auch der Benutzer, der seine Arbeit mit Makros unterstützt, befassen.

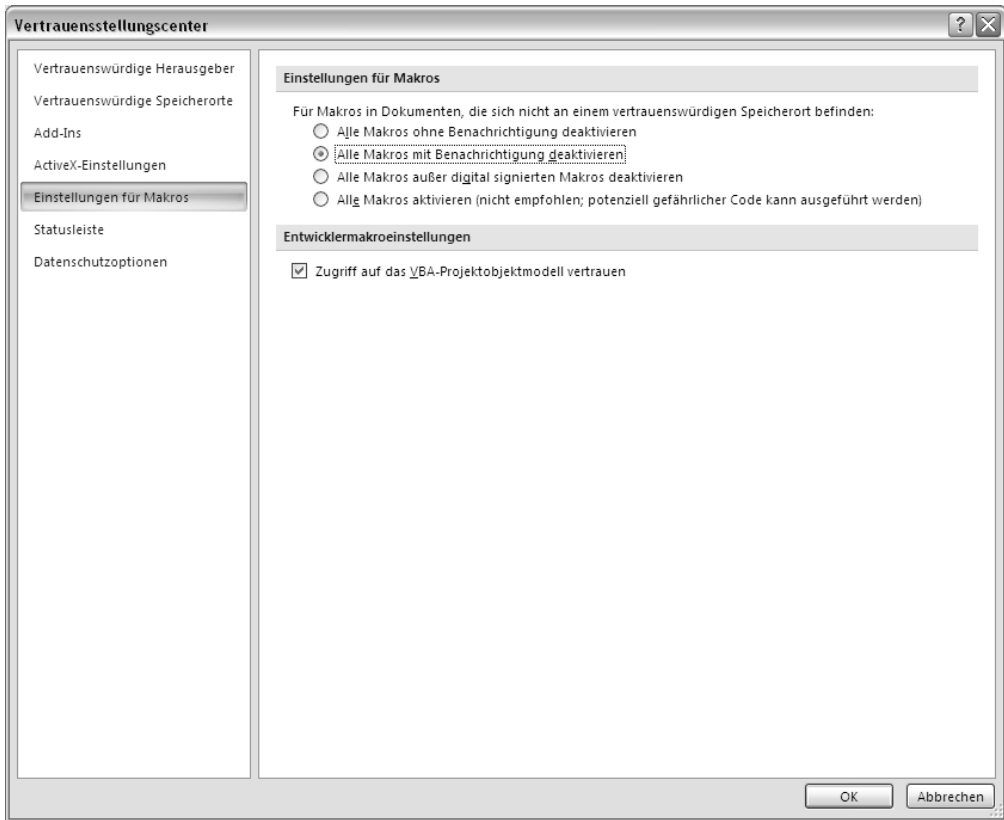
## Sicherheitsstufe anpassen



Um die Sicherheitseinstellungen in Word 2007 festzulegen, öffnen Sie über die *Office*-Schaltfläche und einen Klick auf die Schaltfläche *Word-Optionen* das zugehörige Dialogfeld und wählen dort die Kategorie *Vertrauensstellungszentrum* aus. Diese Seite enthält Links zu weiteren Informationen über die Sicherheitsmaßnahmen sowie die Schaltfläche *Einstellungen für das Vertrauensstellungszentrum*. Klicken Sie darauf, um das Dialogfeld in Abbildung 1.23 einzublenden.



Abbildg. 1.23 Das Word 2007-Vertrauensstellungscenter



Das Dialogfeld umfasst die in Tabelle 1.2 aufgeführten Kategorien. Diese ermöglichen eine verfeinerte Einstellung der Zugriffsmöglichkeiten für die Fernsteuerung von Word gegenüber früheren Versionen. Für Einzelheiten, schlagen Sie in der Word 2007-Hilfe den Ausdruck »Vertrauensstellungscenter« nach.

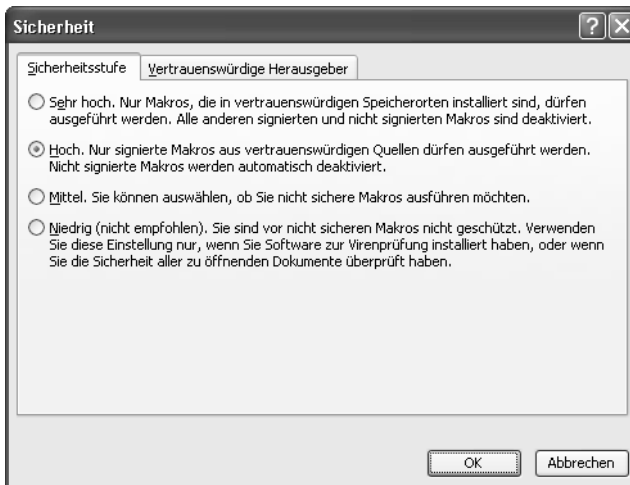
Tabelle 1.2 Die Kategorien des Dialogfeldes Vertrauensstellungscenter

Kategorie	Zweck
Vertrauenswürdige Herausgeber	Ermöglicht die Verwaltung der Liste vertrauenswürdiger Herausgeber für die Word-Anwendung. Hier geht es darum, digital signiertem Code vertrauter Entwickler die Ausführung zu erlauben.
Vertrauenswürdige Speicherorte	Neu in Word 2007 können Dateien in beliebigen Ordnern auf dem Rechner vertraut werden. Das bedeutet, Makro-Code in diesen Dateien ist aktiviert; Makros in anderen Speicherorten werden gesperrt. Automatisch vertraut werden den Speicherorten [Laufwerk]\Programme\Microsoft Office\Templates sowie [Laufwerk]\Programme\Microsoft Office\Office12\Startup.

**Tabelle 1.2** Die Kategorien des Dialogfeldes *Vertrauensstellungscenter* (Fortsetzung)

Kategorie	Zweck
Add-Ins	Enthält erweiterte Sicherheitseinstellungen für COM-Add-Ins. Sie können auf digital signierte Add-Ins begrenzt oder gänzlich gesperrt werden.
ActiveX-Einstellungen	Betrifft ActiveX-Steuerelemente in Dokumenten, die sich nicht in einem vertrauten Speicherort befinden. Die Einstellungen reichen von der vollständigen Deaktivierung bis zur Erlaubnis der Ausführung für alle ActiveX-Steuerelemente.
Einstellungen für Makros	Ist dem Dialogfeld älterer Word-Versionen ähnlich, es fehlt jedoch die Option für die »mittlere« Sicherheitsstufe. Makros können gänzlich – mit oder ohne Meldung – deaktiviert werden, nur aktiviert werden, wenn sie mit einer vertrauten digitalen Signatur versehen sind, oder frei zugelassen werden. Dateien in vertrauten Speicherorten unterliegen diesen Beschränkungen nicht.
Statusleiste	Hier geht es nicht um die Word-Statusleiste, sondern um einen Balken, der zwischen der Multifunktionsleiste und dem Dokument mit Sicherheitsmeldungen eingeblendet wird. In dieser Kategorie wird ihre Aktivierung geregelt.
Datenschutzoptionen	Legt fest, welche Funktionalität in Word auf das Internet zugreifen darf. Hier geht es um die Hilfe, die Rechtschreibprüfung sowie im Dokument gespeicherte Daten (Kommentare, Änderungen, u.ä.). Einige dieser Optionen befanden sich früher in <i>Extras/Optionen/Sicherheit</i> .

Um die Sicherheitsstufe älterer Word-Versionen anzupassen, wählen Sie den Menübefehl *Extras/Makro/Sicherheit*. Diese Registerkarte entspricht in etwa der Kategorie *Einstellungen für Makros* im Vertrauensstellungscenter von Word 2007. (Die in Abbildung 1.24 ersichtliche Sicherheitsstufe *Sehr hoch* wurde erst in Word 2003 eingeführt.)

**Abbildg. 1.24** Setzen Sie die Sicherheitsstufe gemäß unserer Empfehlung auf *Hoch*


In früheren Word-Versionen ist es nicht möglich, vertraute Speicherorte nach Wunsch festzulegen. In Word 2003 wird den folgenden Speicherorten automatisch vertraut, sofern das Kontrollkästchen *Allen installierten Add-Ins vertrauen* in der Registerkarte *Vertrauenswürdige Herausgeber* aktiviert ist.

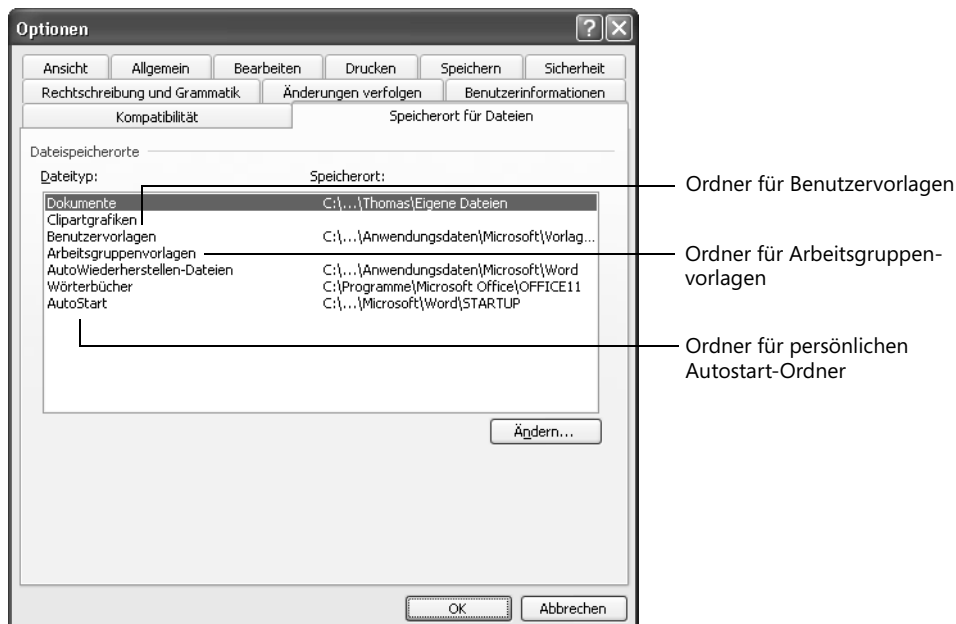
- *C:\Programme\Microsoft Office\Office11\Startup* als allgemeiner Autostart-Ordner
- *C:\Dokumente und Einstellungen\[Benutzername]\Anwendungsdaten\Microsoft\Word\Startup* als persönlicher Autostart-Ordner, sofern in den Optionen kein anderer Ordner eingetragen ist
- *C:\Dokumente und Einstellungen\[Benutzername]\Anwendungsdaten\Microsoft\Vorlagen* 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

#### HINWEIS

Wie in Abbildung 1.25 ersichtlich, können die einzelnen Speicherorte bei Bedarf individuell festgelegt werden. Werden keine Einträge gesetzt, so 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. 1.25 Legen Sie die gewünschten Verzeichnisse für die Add-Ins und Vorlagen fest



Wir Autoren empfehlen die standardmäßigen Einstellungen für die Makrosicherheit. Für Word 2007 bedeutet dies, dass Sie die Option *Alle Makros außer digital signierten Makros deaktivieren* auswählen und für Word 2003 die Sicherheitsstufe *Hoch*. So stellen Sie sicher, dass nur Makros ausgeführt werden, deren Code Sie kennen oder deren Quellen Sie vertrauen.

Zusätzlich empfehlen wir für Word 2003, das Kontrollkästchen *Allen installierten Add-Ins und Vorlagen vertrauen* auf der Registerkarte *Vertrauenswürdige Herausgeber* zu aktivieren. Wir stellen uns auf den Standpunkt, dass das Abspeichern von Dateien in diesem Ordner viel bewusster vorgenommen wird und deshalb diesen Dateien im Allgemeinen vertraut werden kann.

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.

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

Zertifikat

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>1</sup>. Diese gewährleistet, dass der Antragssteller 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 Antragssteller durch die Zertifizierungsstelle ein Schlüsselpaar (privater und öffentlicher Schlüssel<sup>2</sup>) geliefert. Um die Integrität dieses Zertifikats zu garantieren, wird es mit dem privaten Schlüssel der Zertifizierungsstelle digital unterschrieben.<sup>3</sup>

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

Signatur

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. ►

<sup>1</sup> Bekannte und anerkannte Zertifizierungsstellen sind beispielsweise VeriSign (<http://www.verisign.de>) oder Thawte (<http://www.thawte.com>).

<sup>2</sup> 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.

<sup>3</sup> Digitales Unterschreiben eines Zertifikats: Ein Hash wird über das Zertifikat gebildet, mit dem privaten Schlüssel der Zertifizierungsstelle verschlüsselt und als Signatur dem Zertifikat angehängt.

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 niemand Unbefugtem 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).

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

## Eigene Signatur mittels *selfcert.exe* 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.



2007

Die Schritte für Word 2007 sind in der Anwendungshilfe ausführlich beschrieben. Suchen Sie den Begriff »Digitales Signieren eines Makroprojekts« und wählen Sie das gleichnamige Thema aus. Die Schritte unterscheiden sich nicht wesentlich von den nachfolgend beschriebenen.

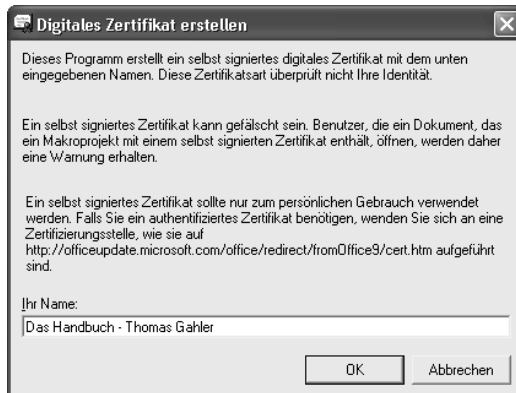
Zum Lieferumfang von Microsoft Office gehört das kleine Programm *selfcert.exe*. Die Datei wird, sofern die entsprechende Option ausgewählt wurde, während der Programminstallation im Verzeichnis *C:\Programme\Microsoft Office\Office11* (bzw. *Office12* bei Office 2007) angelegt. Mit diesem Hilfsprogramm 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. 1.26**

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. Sie sollten auf keinen Fall für professionell erstellte 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 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:

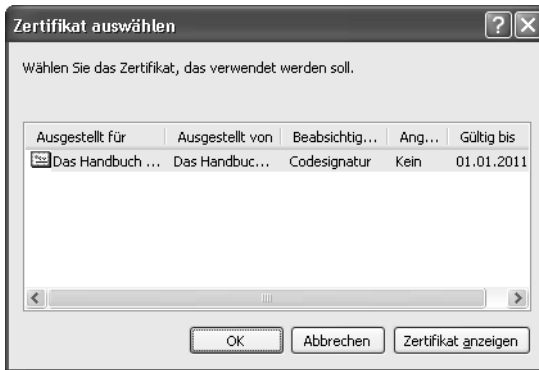
1. Öffnen Sie die entsprechende Datei und wechseln Sie in den Visual Basic-Editor.
2. Wählen Sie den Menübefehl *Extras/Digitale Signatur* und betätigen Sie im Dialogfeld *Digitale Signatur* die Schaltfläche *Wählen*.
3. Markieren Sie im Dialogfeld *Zertifikat auswählen* das gewünschte Zertifikat in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

**HINWEIS**

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

Abbildg. 1.27

Weisen Sie das gewünschte Zertifikat dem Projekt zu



## Zertifikat zur Signatur erstellen



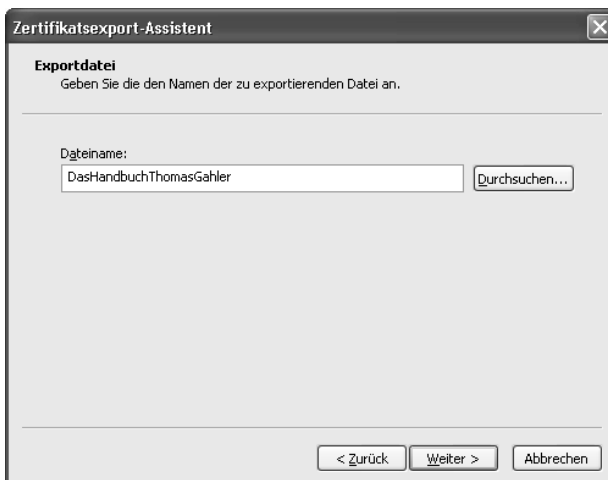
Wurden Makros mit der gemäß obigem Abschnitt »Eigene Signatur mittels *selfcert.exe* erstellen« erzeugten Signatur signiert, muss in einem zweiten Schritt das zugehörige Zertifikat erzeugt werden, damit es zusammen mit den zum Projekt gehörenden Dateien weitergereicht werden kann. Dieses Zertifikat stellt 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*. Klicken Sie im Dialogfeld *Digitale Signatur* auf die Schaltfläche *Wählen*.
2. Im Dialogfeld *Zertifikat auswählen* betätigen Sie die Schaltfläche *Zertifikat anzeigen* und wechseln dann zur Registerkarte *Details*.
3. Betätigen Sie die Schaltfläche *In Datei kopieren* und folgen Sie den Anweisungen des Assistenten.
4. Weisen Sie der zukünftigen Datei im vierten Arbeitsschritt einen aussagekräftigen Namen zu und folgen Sie weiter den Anweisungen des Assistenten.

Abbildg. 1.28

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

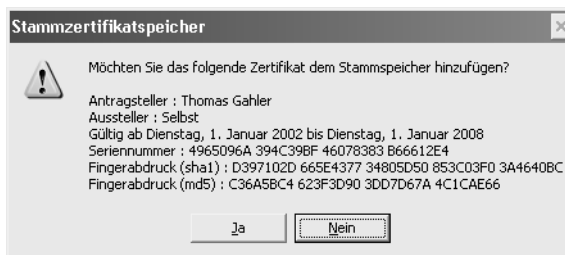
Wird eine Datei, wie in Abschnitt »VBA-Projekte mit einer Signatur versehen« in diesem Kapitel beschrieben, signiert und weitergereicht, muss das zugehörige Zertifikat vorab auf der Arbeitsstation eingelesen werden, damit die Makros aktiviert werden können.

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 und bestätigen Sie die abschließende Sicherheitsabfrage mit *Ja*.

Abbildg. 1.29

Nehmen Sie das Zertifikat in den Zertifikatspeicher auf



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 1.30).



Abbildg. 1.30 Sicherheitswarnung quittieren und Makros aktivieren



## Zusammenfassung

In diesem 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 und wie diese in einem zweiten Schritt nachbearbeitet werden können (Seite 29 ff.).
- Behilfliche Werkzeuge im VB-Editor wurden ab Seite 36 vorgestellt, unter anderen die Hilfe und der Objektkatalog.
- Weiter wurde beschrieben, wie der Benutzer seine Makros in der Word-Umgebung über eine Schaltfläche oder Tastaturkombination zugänglich macht (Seite 47 ff.).
- Ein weiterer Abschnitt beschäftigte sich mit der integrierten Makrosicherheit von Word (Seite 52 ff.) und wie ein VBA-Projekt digital signiert werden kann (Seite 57 ff.).



## Kapitel 2

# VBA-Grundlagen

**In diesem Kapitel:**

Variablen	64
Konstanten	77
Benutzerdefinierte Typen	79
Nützliche VBA-Funktionen	86
Bedingungen	92
Schleifen	95
Code im VB-Editor debuggen	98
Fehlerbehandlung	105
Dateisystem-Operationen	112
Zusammenfassung	123

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 Enumerations, 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 gezeigt, 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://msdn2.microsoft.com/en-us/library/aa192490\(office.11\).aspx](http://msdn2.microsoft.com/en-us/library/aa192490(office.11).aspx)
- [http://msdn2.microsoft.com/en-us/library/keh31dz1\(VS.80\).aspx](http://msdn2.microsoft.com/en-us/library/keh31dz1(VS.80).aspx)

## 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
```

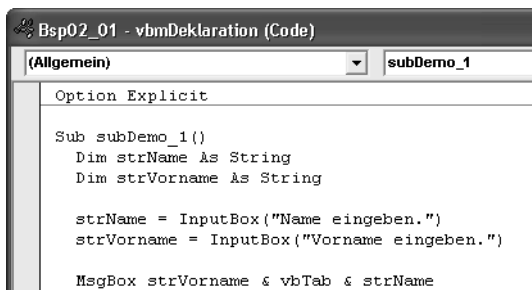
## Standard-Datentypen

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. 2.1 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

#### Hinweis zu .NET

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      Hinweis zu .NET**

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. Das Typkennzeichen für Long dient das Et-Zeichen (&). Der Standardwert ist Null.

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

---

**HINWEIS      Hinweis zu .NET**

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).

---

<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.

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

### HINWEIS Hinweis zu .NET

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 Kompilervorgangs 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 Variable:

Prozedur  
Modul  
Projekt

- Auf der Ebene der Prozedur bzw. Funktion. Die Variable kann nur innerhalb der aktuellen Prozedur verwendet werden.
- Auf der Ebene des Programmmoduls. Die Variable kann innerhalb des ganzen Moduls verwendet werden.
- 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.

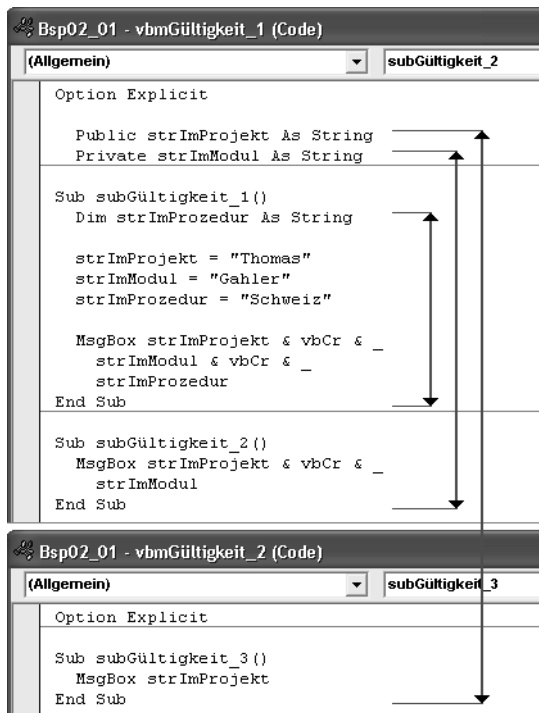
Public

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. 2.2 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« weiter hinten in diesem Kapitel.
- 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. Die Autoren empfehlen, in *Extras/Optionen/Editor* des VB-Editors die Option *Variablendeklaration erforderlich* zu aktivieren.

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 2.1

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 instanziiieren
    Set doc = Documents.Add(
        Template:=ThisDocument.FullName)
    'Tabelle instanziiieren
    Set tbl = doc.Tables(1)
    'Zelle instanziiieren
    Set cel = tbl.Cell(1, 1)
    cel.Range.Text = Now
    'Alle Objekte freigeben
    Set cel = Nothing
```

**Listing 2.1** Variablen vom Typ *Object* werden in umgekehrter Reihenfolge freigegeben (Fortsetzung)

```
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 2.1 aufgeführt sind. Weitere Informationen finden Sie in der Online-Hilfe von VBA zum Thema »Typ-Umwandlungsfunktionen«.

**Tabelle 2.1** 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 2.1.

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**    **Hinweis zu .NET**

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

Das gilt jedoch nicht für C#. Falls Sie VBA-Code nach 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 der 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.
- 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

#### Hinweis zu .NET

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 2.2 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 2.2

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
```

<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.

Um den Prozeduraufbau zur Berechnung der Dreiecksfläche so einfach wie möglich zu halten, können die Variablen auch auf Modulebene deklariert werden. In Listing 2.3 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.

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 2.3** 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 2.4 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 2.4**

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 2.5 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 2.6 ein Datenfeld für die einzelnen Adresszeilen erstellt wird.

**Listing 2.5** 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 2.6 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 Modulebene 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
```

ReDim  
ReDim  
Preserve

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

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

#### 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 2.6**

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 2.6 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 das Kapitel 5.



Die aufgeführten Codesequenzen finden Sie in der Beispieldatei *Bsp02\_01.doc* auf der CD-ROM zum Buch im Ordner `\Beispiele\Kap02`.

## 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 gänzlich 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 2.7** Verwenden von Konstanten in Umrechnungsfunktionen

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

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 2.7 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 2.7 zeigen, dass die passende Deklaration oft erst auf den zweiten Blick zu erkennen ist.



Die aufgeführten Codesequenzen finden Sie in der Beispieldatei *Bsp02\_02.doc* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap02*.

# 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
```

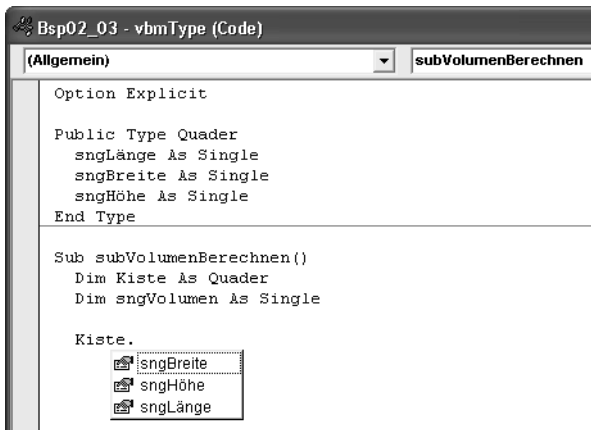
Benutzerdefinierte Typen können grundsätzlich nur auf Modulebene deklariert werden. Es besteht jedoch die Möglichkeit, die Gültigkeit für den neuen Typ auf das entsprechende Modul zu beschränken, und zwar durch Verwendung des Schlüsselworts `Private` innerhalb der Deklarationszeile.

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. 2.3** 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**

#### **Hinweis zu .NET**

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 2.8 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. 2.4

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)	"Gähler"	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 2.8

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 2.9 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 2.9

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")
```

**Listing 2.9** Verwalten einer dynamischen Anzahl von Mitarbeitern, ein mäßiges Codebeispiel (Fortsetzung)

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

    intZähler = intZähler + 1
Loop
End Sub
```

In Listing 2.10 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 2.10** 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 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 EWochentag
    eMontag = 1
    eDienstag = 2
    eMittwoch = 3
    eDonnerstag = 4
    eFreitag = 5
    eSamstag = 6
    eSonntag = 7
End Enum
```

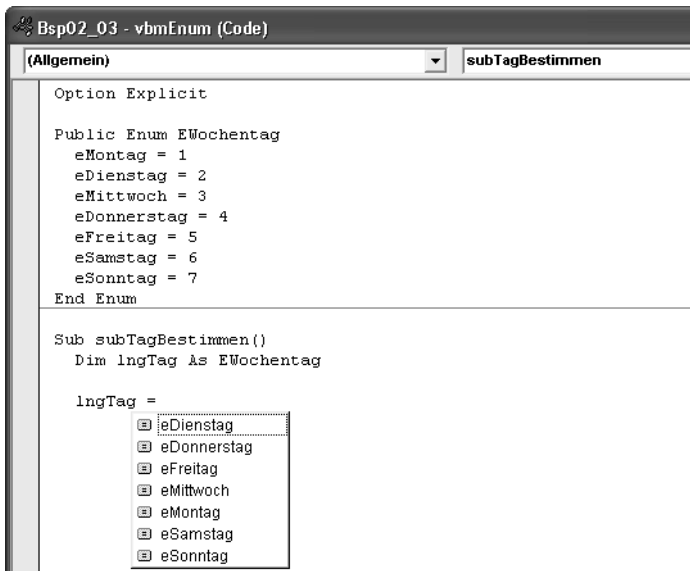
**HINWEIS** Ein manuelles Zuweisen von Zahlenwerten an die einzelnen Elemente der Aufzählung ist nicht zwingend notwendig. Werden keine spezifischen Werte zugewiesen, werden die Elemente automatisch in der erfassten Reihenfolge durchnummeriert.

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. 2.5** 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 regelmäßig 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 2.11 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 2.11** Auswerten der eingegeben 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
```



Listing 2.11 Auswerten der eingegeben Himmelsrichtung, ein schlechtes Codebeispiel (Fortsetzung)

```

    strHimmelsrichtung = "Westen"
    Case Else
        strHimmelsrichtung = "Falsche Zahl angegeben."
    End Select

    MsgBox CStr(IngHimmelsrichtung) & vbTab & strHimmelsrichtung
End Sub

```

In Listing 2.12 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 2.12 Auswerten der eingegeben Himmelsrichtung, ein gelungenes Codebeispiel

```

Option Explicit

Enum EHimmelrichtung
    eNorden = 1
    eOsten = 2
    eSüden = 3
    eWesten = 4
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) & vbTab & 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 2.13 dargestellt.

**Listing 2.13** 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) _
    As Single

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



Die aufgeführten Codesequenzen finden Sie in der Beispieldatei *Bsp02\_03.doc* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap02*.

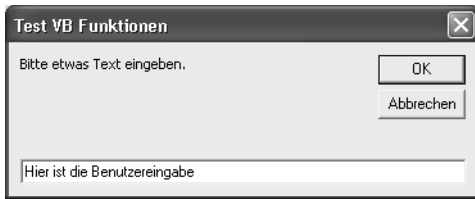
## Nützliche VBA-Funktionen

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 2.6 abgebildet.

Abbildg. 2.6 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 2.14 ist ein einfaches Beispiel dargestellt. In der Prozedur *InputBoxAnzeigen* werden zwei Variablen als Zeichenketten (String) deklariert. Der Aufforderungstext wird *strAufforderung* zugewiesen, den Wert für *strAntwort* gibt die InputBox in Abbildung 2.6 zurück. Der Text, den der Benutzer in das Eingabefeld einträgt, wird in einer MsgBox angezeigt, wie in Abbildung 2.7 dargestellt.

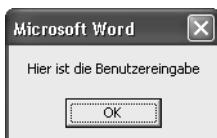
Listing 2.14 Zusammenwirken der Funktionen *InputBox* und *MsgBox*

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

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

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

Abbildg. 2.7 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 2.2 sind die möglichen Werte aufgelistet. Diese können miteinander addiert werden, um eine beliebige Kombination von Schaltflächen und Symbolen anzuzeigen, wie das Listing 2.15 und die Abbildung 2.8 demonstrieren.

**Tabelle 2.2** Werte für das Argument `buttons` der Funktion `MsgBox`

Konstante	Wert	Beschreibung
<code>vbOKOnly</code>	0	Nur die Schaltfläche <i>OK</i> anzeigen
<code>vbOKCancel</code>	1	Schaltflächen <i>OK</i> und <i>Abbrechen</i> anzeigen
<code>vbAbortRetryIgnore</code>	2	Schaltflächen <i>Abbruch</i> , <i>Wiederholen</i> und <i>Ignorieren</i> anzeigen
<code>vbYesNoCancel</code>	3	Schaltflächen <i>Ja</i> , <i>Nein</i> und <i>Abbrechen</i> anzeigen
<code>vbYesNo</code>	4	Schaltflächen <i>Ja</i> und <i>Nein</i> anzeigen
<code>vbRetryCancel</code>	5	Schaltflächen <i>Wiederholen</i> und <i>Abbrechen</i> anzeigen
<code>vbCritical</code>	16	 Meldung mit Stopp-Symbol anzeigen
<code>vbQuestion</code>	32	 Meldung mit Fragezeichen-Symbol anzeigen
<code>vbExclamation</code>	48	 Meldung mit Ausrufezeichen-Symbol anzeigen
<code>vbInformation</code>	64	 Meldung mit Info-Symbol anzeigen
<code>vbDefaultButton1</code>	0	Erste Schaltfläche ist Standardschaltfläche
<code>vbDefaultButton2</code>	256	Zweite Schaltfläche ist Standardschaltfläche
<code>vbDefaultButton3</code>	512	Dritte Schaltfläche ist Standardschaltfläche
<code>vbDefaultButton4</code>	768	Vierte Schaltfläche ist Standardschaltfläche
<code>vbApplicationModal</code>	0	An die Anwendung gebunden. Der Benutzer muss auf das Meldungsfeld reagieren, bevor er seine Arbeit mit der aktuellen Anwendung fortsetzen kann.
<code>vbSystemModal</code>	4096	An das System gebunden. Alle Anwendungen werden unterbrochen, bis der Benutzer auf das Meldungsfeld reagiert.
<code>vbMsgBoxHelpButton</code>	16384	Fügt die Schaltfläche <i>Hilfe</i> dem Meldungsfeld hinzu
<code>vbMsgBoxSetForeground</code>	65536	Setzt das Meldungsfeld in den Vordergrund
<code>vbMsgBoxRight</code>	524288	Der Text wird rechts ausgerichtet
<code>vbMsgBoxRTLReading</code>	1048576	Legt fest, dass Text auf hebräischen und arabischen Systemen von rechts nach links auszurichten ist

**Abbildg. 2.8** Eine *MsgBox* aussagekräftig mit verschiedenen Kombinationen von Schaltflächen und Symbolen gestalten



**Listing 2.15** 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 2.8, 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 2.3 aufgelistet. Wie Sie diese Information auswerten, ist im Abschnitt »Bedingungen« in diesem Kapitel beschrieben.

**Tabelle 2.3** 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 2.16. Das Resultat sehen Sie in Abbildung 2.9.

**Listing 2.16** Die Wirkungsweise der Zeichenketten-Funktionen *Left*, *Right* und *Mid*

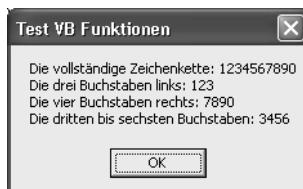
```
Sub TestVBZeichenkettenFunktionen1()
    Dim strAufforderung As String
    Dim strAntwort As String
    Dim lWert As Long

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

#### TIPP

In Listing 2.16 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. 2.9** Das Resultat von Listing 2.16



Instr,  
Instr-  
Rev,  
Replace

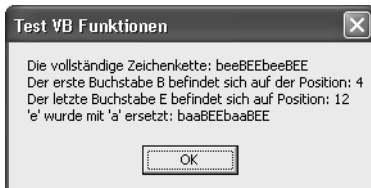
Es ist manchmal hilfreich zu wissen, ob ein Zeichen in einer Zeichenkette überhaupt vorkommt, und wenn ja, an welcher Stelle. Dafür stellt Visual Basic die Funktionen Instr und InstrRev bereit, die die Position des gesuchten Zeichens zurückgeben, wie in Abbildung 2.10 und Listing 2.17 zu sehen. Die Syntax 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. 2.10 Die Wirkung der Funktionen *Instr*, *InstrRev*, und *Replace*



Listing 2.17 Die Funktionen *Instr*, *InstrRev* und *Replace*

```
Sub TestVBZeichenkettenFunktionen2()

    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 & 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 2.11 bzw. in Listing 2.18. Diese Funktion wird auch für die Formatierung von Zahlen verwendet. Die Syntax lautet:

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

Abbildg. 2.11 Das aktuelle Datum wurde mit der Funktion *Format* nach dem Muster *t-mmm-jiji* 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 2.18** 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 2.19 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 2.12 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. 2.12** Die eingegebene Zahl wird mit der Funktion *Format* formatiert



**Listing 2.19** 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 2.20 um einen Else-Block erweitert. Ergibt die Auswertung der If-Codezeile den Wert »Falsch«, springt die Ausführung hierher und zeigt die Fehlermeldung aus Abbildung 2.13 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. 2.13** Buchstaben wurden versehentlich statt Zahlen für die Datumsangabe eingegeben



Zwei neue Visual Basic-Funktionen wurden in Listing 2.20 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-2007 oder 2007/01/01. Wichtig zu wissen ist, dass die Funktion ein gültiges Datum aktiv forciert. 8.24.2007 wird ebenso akzeptiert wie 24.8.2007; 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 2.10 vielleicht aufgefallen ist, unterscheiden die VB-Funktionen zwischen Groß- und Kleinschreibung.

**Listing 2.20** 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." & _
            & vbCr & "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 2.14. Der Benutzer hat drei Schaltflächen zur Auswahl. Es wäre durchaus möglich, den Rückgabewert mit If wie folgt auszuwerten:

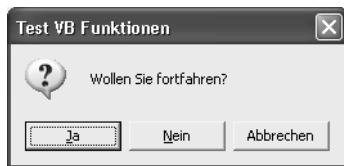
```

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 2.21. Zudem arbeitet sie meist langsamer.

**Abbildg. 2.14** 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 2.21** 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

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

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

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 ein Mal ausgeführt, egal, ob die Bedingung zutrifft.
- While bedeutet, der geprüfte Wert wird sich nur ein Mal 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 2.22 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 2.22 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)
```

**Listing 2.22** Die Do...Loop-Variationen (Fortsetzung)

```
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 2.23 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 2.23 Mit For...Next wird eine Schleife eine bestimmte Anzahl Male durchgelaufen

```

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 2.24.

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 4 behandelt.

Listing 2.24 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

    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 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 *Bsp02\_04.doc* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap02*.

## Code im VB-Editor debuggen

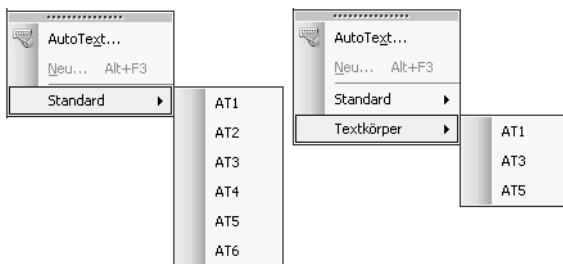
Vorausgesetzt, Sie haben Option `Explicit` eingeschaltet **und** verfügen über viel Erfahrung sowie eine übergroße Portion Glück, wird Ihr Code bei jedem ersten Mal ohne Fehler laufen. Ist Ihnen jedoch so viel Glück zuteil, wären Sie gut beraten, besser in einem Spielcasino anstatt vorm Bildschirm zu 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 2.15 stellt ein solches Problem dar, das an die Diskussion über die `For...Next`-Anweisung im Abschnitt »Schleifen« in diesem Kapitel anschließt.

Abbildg. 2.15 In einer *For Each*-Schleife wurde nur jeder zweite Eintrag bearbeitet



Die sechs AutoText-Einträge links in Abbildung 2.15 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 2.25 ein, die auch einwandfrei läuft. Pflichtbewusst schaut der Entwickler schnell im Menü (Abbildung 2.15) 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 2.25** Die Kategorie eines AutoText-Eintrags 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

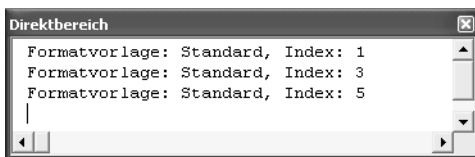
    Set tmpl = Documents("DebugTest.dot").AttachedTemplate
    Set rng = ActiveDocument.Range
    rng.Collapse wdCollapseEnd
    For Each AT In tmpl.AutoTextEntries
        If AT.StyleName = "Standard" Then
            Set rng = AT.Insert(Where:=rng, RichText:=True)
            rng.Style = ActiveDocument.Styles(wdStyleBodyText)
            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 2.16 ersichtlich.

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

**Abbildg. 2.16** 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.

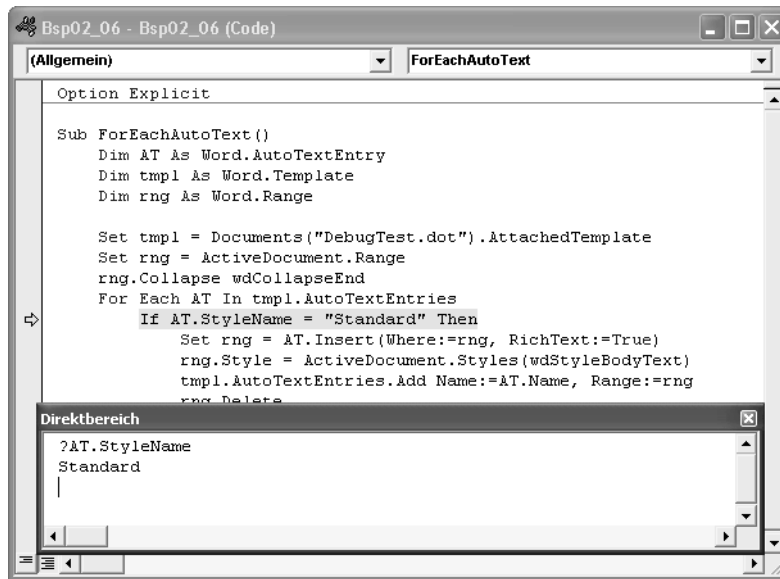
Der Befehl für die schrittweise Ausführung einer Prozedur befindet sich im Menü *Debuggen/Einzel-schritt*. Es wäre nun recht mühsam, für jede Codezeile das Menü zu öffnen und den Eintrag auszuwählen. Deshalb sollten Sie sich die Taste **[F8]** einprägen. Beim ersten Tastendruck wird die erste Codezeile (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 2.17).
- Die Werte dem Überwachungsfenster zuweisen, wie in Abbildung 2.18 ersichtlich.

Abbildg. 2.17

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

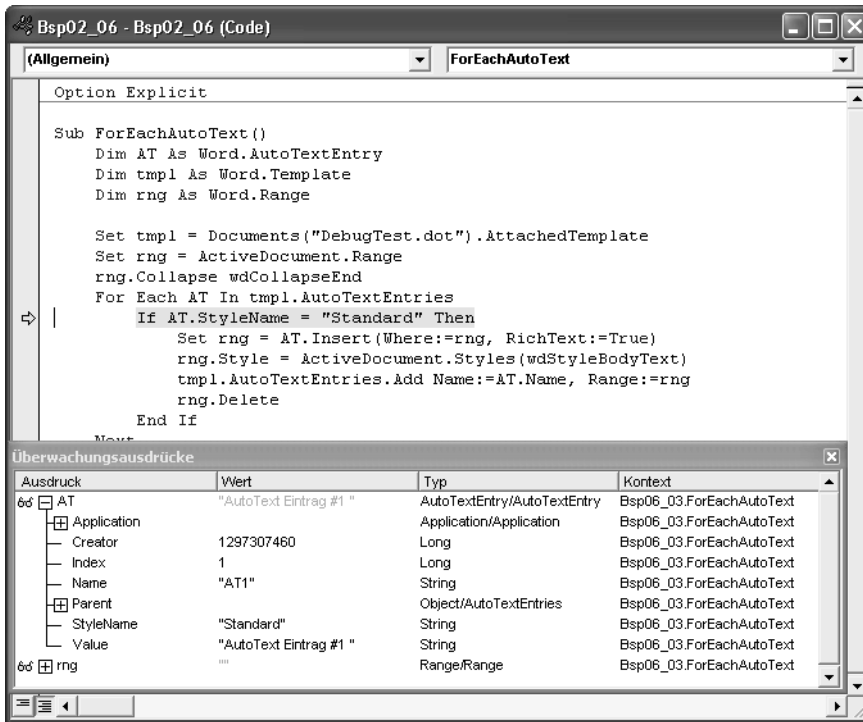


#### TIPP

Während der schrittweisen Ausführung dürfen Sie den Variablen auch andere Werte im Direktfenster zuweisen. Beispiel: `lEnde = 7` (aus Listing 2.24) und anschließend die `[↵]`-Taste drücken.



Abbildg. 2.18 Im Überwachungsbereich können mehrere Werte gleichzeitig bei jedem Schritt geprüft werden

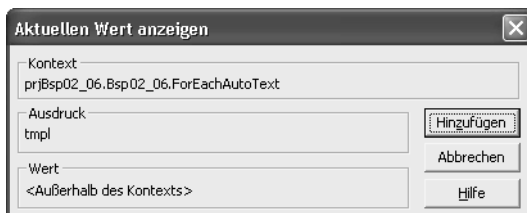


## Der Überwachungsbereich

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 **Alt** + **F9**. Das Dialogfeld *Aktuellen Wert anzeigen* (siehe Abbildung 2.19) wird eingeblendet; klicken Sie auf *Hinzufügen*. (Ein Ausdruck darf vor sowie während der Code-Ausführung hinzugefügt werden.)

Abbildg. 2.19 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 2.26. 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 irgendwie ist schief gegangen (vermutlich ein Problem in der Word-Anwendung).

**Listing 2.26** 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 Formatvorlage »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 2.20 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. 2.20** 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 2.27, testet nochmals und fügt ihn dann seiner Code-Bibliothek zu. (Die C#-Version finden Sie in Listing 2.28.)

**Listing 2.27** 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
```

**Listing 2.27** Alle AutoText-Einträge einer bestimmten Kategorie einer anderen Kategorie zuweisen (Fortsetzung)

```

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

strKategorieAlt = "Standard"
strKategorieNeu = "Textkörper"
Set tmpl = Documents("DebugTest.dot").AttachedTemplate
Set rng = ActiveDocument.Range
rng.Collapse wdCollapseEnd
lAnfang = tmpl.AutoTextEntries.Count
lEnde = 1
For lZaehler = lAnfang To lEnde Step -1
    Set AT = tmpl.AutoTextEntries(lZaehler)
    If AT.StyleName = strKategorieAlt Then
        Set rng = AT.Insert(Where:=rng, RichText:=True)
        rng.Style = ActiveDocument.Styles(strKategorieNeu)
        tmpl.AutoTextEntries.Add Name:=AT.Name, Range:=rng
        If lZaehler = lEnde Then
            tmpl.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 2.28** 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,
            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);
    }
}

```

Listing 2.28 Die C#-Version (Fortsetzung)

```

int lAnfang = tmp1.AutoTextEntries.Count;
int lEnde = 1;
for (int lZaehler = lAnfang; lZaehler >= lEnde; lZaehler --)
{
    object objATentry = (object) lZaehler;
    wd.AutoTextEntry AT = tmp1.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);
        tmp1.AutoTextEntries.Add(AT.Name, rngAT);
        if (lZaehler == lEnde)
        {
            tmp1.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 *Bsp02\_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\Kap02*.

## 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 Theorie 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 2.21, 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. 2.21** 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.

In Listing 2.29 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.

Fehler-  
behand-  
lung  
ein-  
schalten

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 2.29** 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 2.30.

**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 2.30** 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
    Dim aTest As Variant

    On Error GoTo Fehlerbehandlung
    aTest = Array("zero", "eins", "zwei", "drei")
Start:
    ' ... Code ...
Exit Sub
```

**Listing 2.30** 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. (Fortsetzung)

```

lAnfang = CLng(InputBox("Bitte die Anfangszahl eingeben"))
lEnde = CLng(InputBox("Bitte die Endezahl 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 Fehler: *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 2.31 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 2.31** 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 Endezahl eingeben"))
    For lZaehler = lAnfang To lEnde
        Debug.Print aTest(lZaehler)
    Next

EndPunkt:

```



**Listing 2.31** In diesem Beispiel werden mit einer *Select Case*-Anweisung in der Fehlerbehandlung die verschiedenen Fehler, die in der Prozedur vorkommen können, differenziert behandelt (*Fortsetzung*)

```
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 2.32 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 2.32** *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")
    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." _
            & vbCrLf & Err.Description & vbCrLf & "Fehlernummer: " & Err.Number
    End If
End Sub
```

**Listing 2.32** *On Error Resume Next* schaltet die VB-Fehlerfunktionalität vorübergehend aus. Es ist wichtig, diese mit *On Error GoTo* baldmöglichst wieder einzuschalten. (Fortsetzung)

```
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 2.33. 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 2.22 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 2.33** 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 Endezahl eingeben"))
    If lAnfang > lEnde Then
        Err.Raise lEINGABEFehler, "Fehlerbehandlung4",
            "Der Anfangswert ist größer als der Endwert."
    End If

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

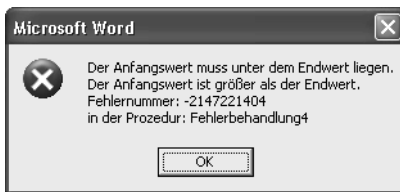
Listing 2.33 Beispiel für die Verwendung des *Err*-Objekts (Fortsetzung)

```

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 1EINGABEFEHLER
    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. 2.22 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 2.21).



Die Beispieldatei *Bsp02\_07.doc* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap02*.

# 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 2.34 wird dieser Umstand genutzt, um eine Liste aller vorhandenen Dokumente im aktuellen Verzeichnis auszugeben.

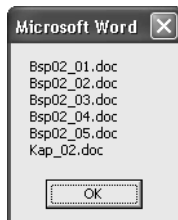
**Listing 2.34** Auflisten aller Dateinamen im aktuellen Verzeichnis

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

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

    MsgBox strDateiliste
End Sub
```

**Abbildg. 2.23** 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.docx*, *DateiB.docx* und *DateiC.docx* unter Verwendung des Platzhalters gleichzeitig gelöscht werden:

```
Kill "Datei?.docx"
```

\*

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

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

## 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 Ordernamen, 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 2.35 wird mittels einer eigenen Funktion diesem Umstand Rechnung getragen. Die Funktion hängt bei Bedarf den fehlenden Backslash an den Ordernamen an.

**Listing 2.35** Beim Verknüpfen mit Ordernamen 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

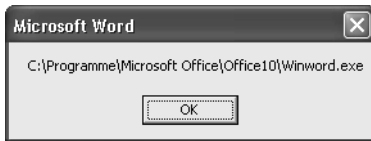
Public Function fktPfadInklBackslash( _
    ByVal strPfad As String) _
    As String
    'Die Funktion kontrolliert, ob der übergebene Pfad
    'als letztes Zeichen einen Backslash aufweist.
    'Falls nicht wird dieser angehängt.
    If Not (Right$(strPfad, 1) = Application.PathSeparator) Then
        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. 2.24 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.docx") = "" 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 2.34 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 2.36 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 2.36** Anhand der Dateiattribute prüfen, ob eine Datei überhaupt vorhanden ist

```
Sub PrüfenObDateiVorhandenIst()
    MsgBox fktExistiertDatei("C:\BOOTLOG.TXT")
    MsgBox fktExistiertDatei("C:\Temp\Test.docx")
    MsgBox fktExistiertDatei("C:\Programme")
End Sub

Public Function fktExistiertDatei( _
    ByVal strDateiname As String) _
    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
```

**Listing 2.36** Anhand der Dateiattribute prüfen, ob eine Datei überhaupt vorhanden ist (*Fortsetzung*)

```
On Error Resume Next
fktExistiertDatei = CBool((GetAttr(strDateiname) And intATTR_NOTFILE) = 0)
End Function
```

Wie aus Tabelle 2.4 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 2.4** 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 2.36 verwendet und leicht angepasst.

In Listing 2.37 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 2.37

Anhand der Dateiattribute prüfen, ob ein Ordner überhaupt vorhanden ist

```
Sub PrüfenObOrdnerVorhandenIst()
    MsgBox fktExistiertOrdner("C:\Programme")
    MsgBox fktExistiertOrdner("C:\")
    MsgBox fktExistiertOrdner("C:\Temp\Test.docx")
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 2.4 ersichtlich, sind den einzelnen Attributen Zahlenwerte zugewiesen. Der Rückgabewert wird bitweise mit der Konstante `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 2.38 genutzt, um die entsprechende Prüfung durchzuführen.

**Listing 2.38** 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 2.39 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 2.36 und Listing 2.38 bewerkstelligt.

**Listing 2.39** 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.docx")
    MsgBox fktDateiLöschen("C:\Temp\TestB.docx")
End Sub
```

Listing 2.39 Datei auf dem Datenträger löschen und mögliche Fehler bearbeiten (Fortsetzung)

```

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)

    'Ist die Datei weder schreibgeschützt noch versteckt
    If bFlag Then
        bFlag = CBool((GetAttr(strDateiname) And intATTR_NODELETE) = 0)

    'Datei löschen
    If bFlag Then
        Kill strDateiname

    'Konnte die Datei wirklich entfernt werden.
        bFlag = Not fktExistiertDatei(strDateiname)
    End If
    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 2.37 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 Funktionen aus der Sammlung bereits erfolgt ist.

In Listing 2.40 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 2.40** Ermitteln des Speicherdatums eines Verzeichniseintrags

```
Sub DateiDatumUndZeitErmittleIn()
    MsgBox fktDateiDatumUndZeitErmittleIn("C:\Temp\", "dd/ mmmm yyyy")
    MsgBox fktDateiDatumUndZeitErmittleIn("C:\Temp\Test.docx")
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 2.36 und Listing 2.37 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. 2.25 Unterschiedlicher Rückgabewert der Funktion in Abhängigkeit zum Parameter *strFormat*

## Größe einer Datei ermitteln

**FileLen** In Listing 2.41 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 2.41 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.docx", eDG_Byte)
    MsgBox fktDateiGrösseErmitteln("C:\Temp\TestA.docx", 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 2.36 auf. Damit das Beispiel ausgeführt werden kann muss diese Funktion im Zugriff sein.

Das gewünschte Ausgabeformat der Funktion *fmtDateiGrößeErmitteln* 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 *fmtDateiGrößeErmitteln* 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. 2.26 Unterschiedlicher Rückgabewert der Funktion in Abhängigkeit der beiden Parameter



Die aufgeführten Codesequenzen finden Sie in der Beispieldatei *Bsp02\_05.doc* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap02*.

# Zusammenfassung

In diesem Kapitel wurden Ihnen 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 64 ff.).
- In einem zweiten Abschnitt wurden die Konstanten (Seite 77 ff.) und die benutzerdefinierten Typen (Seite 79 ff.) erläutert.
- Ein weiterer Abschnitt widmete sich nützlichen VBA-Funktionen (Seite 86 ff.), Programmbedingungen und Schleifen (Seite 92 ff.).
- Ebenso wurde aufgezeigt, wie der Programmcode mit dem Debugger untersucht (Seite 98 ff.) und eventuelle Programmfehler mittels einer Fehlerbehandlung aufgefangen werden können (Seite 105 ff.).
- Praktische Beispiele zum Dateisystem (Seite 112 ff.) runden das Erlernte in diesem Kapitel ab.





## Kapitel 3

# Windows-APIs in VBA nutzen

### In diesem Kapitel:

Aufbau der API-Funktionen	126
Kombinieren und Abschließen von Pfaden	127
Datei über die Dateieindung ausführen	131
Zugriff auf Registry-Einträge	135
Zugriff auf INI-Dateien	140
Name des angemeldeten Benutzers ermitteln	145
Verarbeitung für eine bestimmte Zeit unterbrechen	147
Wave-Datei abspielen	148
Tastenstatus abfragen	151
Zusammenfassung	157

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 *Bsp03\_1.doc* auf der Buch-CD finden Sie im Ordner *\Beispiele\Kap03* alle Beispiele zu den einzelnen Abschnitten in einzelnen Modulen, teilweise mit zusätzlichen Anwendungsbeispielen. Zusätzlich enthält dieser Ordner die Datei *Bsp03-1.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 3.1 erläutert):

```
[Public|Private] Declare Sub|Function Name Lib "LibName" [Alias "AliasName"] [[(arglist)]]
[As Type]
```

Tabelle 3.1 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 3.1 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 3.1 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 3.1 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 Backslashes überprüft werden, bevor sie kombiniert werden können, da andernfalls Fehler bezüglich unbekannter oder falscher Pfade auftreten können.

Listing 3.2 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 Verwendungen 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 3.2):

```
Declare Function PathAddBackslash Lib "shlwapi.dll" Alias "PathAddBackslashA" _
    (ByVal pszPath As String) As Long
```

Tabelle 3.2 Parameter der API-Funktion *PathAddBackslash*

Parameter	Bedeutung	Ein-/Ausgabe
pszPath	Variable mit zu prüfendem Pfad	Ein/Aus

Diese Funktion erwartet als Ein- und Ausgabeparameter 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 Variable (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 3.3 veranschaulicht dieses Prinzip.

Listing 3.3 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. 3.1 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 3.3):

```
Declare Function PathCombine Lib "shlwapi.dll" Alias "PathCombineA" _
    (ByVal szDest As String, _
    ByVal lpszDir As String, _
    ByVal lpszFile As String) As Long
```

Tabelle 3.3 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

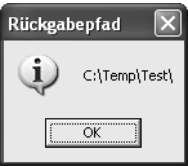
Diese Funktion erwartet neben den beiden Pfadangaben `lpszDir` und `lpszFile`, die kombiniert werden sollen, eine Variable `szDest`, in die das Ergebnis ausgegeben wird.

Die Buffer-Variable `szDest` muss dabei wieder so groß angelegt werden, dass der kombinierte Pfad auf jeden Fall hineinpasst, wie Listing 3.4 veranschaulicht.

Listing 3.4 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. 3.2 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 3.4):

```
Declare Function PathAppend Lib "shlwapi.dll" Alias "PathAppendA" _
    (ByVal pszPath As String, ByVal pMore As String) As Long
```

**Tabelle 3.4** 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 dem Dateinamen pMore; der kombinierte Pfad mit der Datei wird dabei in der Variable pszPath wieder ausgegeben. Die Buffer-Variable pszPath muss dabei wieder so groß angelegt werden, dass der kombinierte Pfad auf jeden Fall hineinpasst, wie Listing 3.5 veranschaulicht.

**Listing 3.5** 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. 3.3** 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:

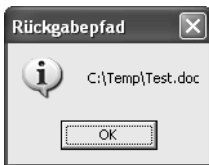
```
Declare Function PathAddExtension Lib "shlwapi.dll" Alias "PathAddExtensionA" ( _
    ByVal pszPath As String, _
    ByVal pszExt As String ) As Long
```

Diese Funktion erwartet neben dem Dateinamen `pszPath` die Erweiterung `pszExt`; der Dateiname mit Erweiterung wird dabei wieder in der Variable `pszPath` ausgegeben. Die Buffer-Variable `pszPath` muss dabei wieder so groß angelegt werden, dass der vollständige Dateiname auf jeden Fall hineinpasst, wie Listing 3.6 veranschaulicht.

**Listing 3.6** 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. 3.4** Ausgabe der angehängten Dateierweiterung



Die Beispiele zu diesem Abschnitt finden Sie in der Beispieldatei `\Beispiele\Kap03\Bsp03_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 3.7, die Erläuterung der Parameter ist in Tabelle 3.5 enthalten.

**Listing 3.7** 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 3.7** 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 3.5** 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 angeben, 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 3.8 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 3.8** 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 3.9 bis Listing 3.10 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 3.9 Öffnen einer Datei mittels *ShellExecute*

```
Sub subShellExecute()  
    ' Öffnet die Datei c_FilePath  
    Const c_FilePath = "C:\MVPBuch\Kap03\Bsp03-1.INI"  
    fktShellExecute c_FilePath, , "open"  
End Sub
```

Listing 3.10 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 3.11 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 3.11 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 3.11** 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 Dateierstreckung 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\Kap03\Bsp03_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 diese das Herzstück des Betriebssystems darstellen. 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, ist 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 3.6 listet die Hauptzweige der Registry auf.

Tabelle 3.6 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 3.7 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 3.7** 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 API *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 3.8 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 3.8** 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.

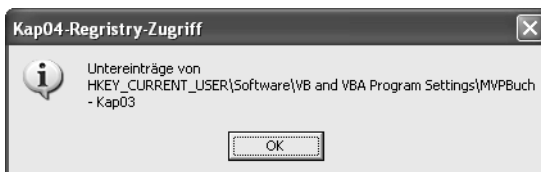
Das Listing 3.13 ermittelt im Registry-Eintrag der VB- und VBA-Einstellungen (siehe Kapitel 12) die Untereinträge im Eintrag *MVPBuch*. Wenn Sie das Makro *SaveVBSetting* in der Beispieldatei *Bsp03\_1.doc* von der Buch-CD (im Ordner *\Beispiele\Kap03*) ausgeführt haben, liefert die nachstehende Prozedur den Namen des Untereintrags zurück: *Kap03*.

**Listing 3.12** Ermitteln aller Untereinträge eines Registry-Eintrags

```
Sub EnumRegKeys()
    Const c_Title = "Kap03-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
    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. 3.5** 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 3.9

```
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 3.9** 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 REG_SZ, REG_MULTI_SZ, REG_EXPAND_SZ	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 3.12 dahingehend geändert, wie Listing 3.13 veranschaulicht, dass zuerst alle Untereinträge in einem Array gesammelt werden und anschließend für diese Array-Einträge die Schlüssel ermittelt werden. Dieses ist notwendig, da für 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 3.13** Ermitteln aller Schlüssel und Werte aller Untereinträge zu einem Registry-Eintrag

```
Sub EnumRegKeyValues()
    ' Prozedur zur Ermittlung aller Untereinträge eines Eintrags
    Const c_Title = "Kap03-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 3.13 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
    End If

```

**Listing 3.13** Ermitteln aller Schlüssel und Werte aller Untereinträge zu einem Registry-Eintrag (*Fortsetzung*)

```

'Durchlaufen der Schlüssel
While RegEnumValue lngKey, 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. 3.6** Ausgabe der gefundenen Untereinträge mit ihren Schlüssel und Werten


Die Beispiele zu diesem Abschnitt finden Sie in der Beispieldatei `\Beispiele\Kap03\Bsp03_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 3.14 dargestellt.



Listing 3.14 Beispiel einer INI-Datei mit sprachspezifischen Informationen

```
[1031]
1=Ihr Nachname
2=Ihr Vorname
3=Straße
4=Ort
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 eckige 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 3.10 aufgelistet):

```
Public Declare Function WritePrivateProfileSection Lib "kernel32" _
    Alias "WritePrivateProfileSectionA" _
    (ByVal lpAppName As String, _
    ByVal lpString As String, _
    ByVal lpFileName As String) As Long
```

Tabelle 3.10 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 3.15 schreibt den String

```
"Eintrag1=TestEintrag1" & vbCrLf & "Eintrag2=TestEintrag2"
```

in die INI-Datei *Bsp03-1.INI*.

**Listing 3.15** Schlüssel und Werte in einen Abschnitt schreiben

```
Sub subWriteINISection()  
    Const c_INIPath = "C:\MVPBuch\Kap03\Bsp03-1.INI"  
    Const c_SecName As String = "Kap03"  
    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, dessen 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 3.11.

**Tabelle 3.11** 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 3.16 veranschaulicht. Das Ergebnis ist in Abbildung 3.7 abgebildet.

Listing 3.16 Auslesen aller Schlüssel und Werte aus dem Abschnitt »Kap03«

```

Sub subReadINISection()
    Const c_Read = "Abschnitt einlesen"
    Const c_INIPath = "C:\MVPBuch\Kap03\Bsp03-1.INI"
    Const c_SecName As String = "Kap03"
    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. 3.7 Ausgabe der im Abschnitt »Kap03« 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 3.12 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 3.12** Parameter der API-Funktion *GetPrivateProfileSectionNames*

Parameter	Bedeutung	Ein-/Ausgabe
lpszReturnBuffer	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 3.17 ermittelt erst die Anzahl und Namen der vorhandenen Abschnitte in der INI-Datei *Bsp03-1.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 3.8 ersichtlich.

**Listing 3.17** Ermitteln und Auslesen aller Abschnitte in der Datei *Bsp03-1.INI*

```
Sub subWrite2INISection()
    Const c_INIPath = "C:\MVPBuch\Kap03\Bsp03-1.INI"
    Const c_SecName As String = "Kap03"
    Const c_SecName2 As String = "Kap04"
    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\Kap03\Bsp03-1.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
```

Listing 3.17 Ermitteln und Auslesen aller Abschnitte in der Datei *Bsp03-1.INI* (Fortsetzung)

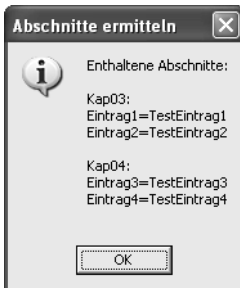
```

    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
    strTemp = Left$(strInput, lngRet)
    fktSplitString = Split(strTemp, vbNullChar)
End Function

```

Abbildg. 3.8 Ausgabe der gefundenen Abschnitte mit ihren Schlüsseln und Werten



Die Beispiele zu diesem Abschnitt finden Sie in der Beispieldatei *\Beispiele\Kap03\Bsp03\_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 3.18 veranschaulicht.

Listing 3.18 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 3.19 befindet. Eine Erläuterung der Parameter befindet sich in Tabelle 3.13.

**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 3.19 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 3.13 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 Buffervariable; 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 3.20 veranschaulicht den Gebrauch der API-Funktion.

**HINWEIS**

Auf einem Windows XP-Einzelplatz-System, das nicht an einem Domaincontroller angemeldet ist, liefert nur

```
NameSamCompatible = 2
```

den Benutzernamen zurück. Alle anderen Werte liefern keine Informationen zurück.

**Listing 3.20**

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\Kap03\Bsp03_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 3.14 und das Listing 3.21 veranschaulicht den Gebrauch.

**Tabelle 3.14** Parameter der API-Funktion *Sleep*

Parameter	Bedeutung	Ein-/Ausgabe
lngMilliseconds	Angabe der Zeit in Millisekunden, für die die Verarbeitung unterbrochen werden soll	Ein

**Listing 3.21** 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 3.22** 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\Kap03\Bsp03_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 3.15 aufgelistet:



```
Public Declare Function sndPlaySound Lib "WinMM.dll" Alias "sndPlaySoundA" ( _
    ByVal lpszSoundName As String, _
    ByVal uFlags As Long ) As Long
```

Tabelle 3.15 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 3.16 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. Der Parameter <b>ASYNC</b>
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 3.22 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 3.23 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
```

**Listing 3.23** Abspielen einer beliebigen Wave-Datei unter Word 2002 (XP) und 2003 (Fortsetzung)

```
Sub subPlaySoundWord2003()  
    ' Wav-Datei unter Word 2003 abspielen  
    Dim bRet As Boolean  
    Dim lRet As Single  
    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, sodass Ihnen dieses Dialogfeld mit eigenem Dateifilter nicht zur Verfügung steht. Der Code für diese Word-Version befindet sich in Listing 3.24.

**HINWEIS**

Das `FileDialog`-Objekt wird in Kapitel 14 näher vorgestellt.

**Listing 3.24** 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 3.25):

- Durch Angabe des NULL-Zeichens als Dateinamen
- Durch Verwendung des SND\_PURGE-Parameters

**Listing 3.25** 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\Kap03\Bsp03_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 3.17 aufgelistet.

```
Declare Function GetKeyState Lib "user32" ( _
    ByVal nVirtKey As Long ) As Integer
```

**Tabelle 3.17** 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 3.18 liefert eine Übersicht über die gängigen Tastenkonstanten.

















**Tabelle 3.18** Virtuelle Tastenkonstanten mit Bedeutung und Hexadezimalwert

Virtuelle Tastenkonstante	Taste	Hexadezimalwert
VK_LBUTTON	Linke Maustaste	&H1
VK_RBUTTON	Rechte Maustaste	&H2
VK_CANCEL		&H3
VK_MBUTTON	Mittlere Maustaste	&H4
VK_XBUTTON1	X1-Maustaste	&H5
VK_XBUTTON2	X2-Maustaste	&H6
VK_BACK		&H8
VK_TAB		&H9
VK_CLEAR		&HC
VK_RETURN		&HD
VK_SHIFT		&H10
VK_CONTROL		&H11
VK_MENU		&H12
VK_PAUSE		&H13
VK_CAPITAL		&H14
VK_ESCAPE		&H1B
VK_SPACE		&H20
VK_PRIOR		&H21
VK_NEXT		&H22
VK_END		&H23
VK_HOME		&H24
VK_LEFT		&H25
VK_UP		&H26
VK_RIGHT		&H27
VK_DOWN		&H28
VK_SELECT	Markieren	&H29
VK_PRINT		&H2A

Tabelle 3.18 Virtuelle Tastenkonstanten mit Bedeutung und Hexadezimalwert (Fortsetzung)

Virtuelle Tastenkonstante	Taste	Hexadezimalwert
VK_EXECUTE	Ausführen	&H2B
VK_SNAPSHOT	Bildschirm drucken	&H2C
VK_INSERT		&H2D
VK_DELETE		&H2E
VK_HELP	Hilfe	&H2F
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

**Tabelle 3.18** Virtuelle Tastenkonstanten mit Bedeutung und Hexadezimalwert (Fortsetzung)

Virtuelle Tastenkonstante	Taste	Hexadezimalwert
VK_F11		&H7A
VK_F12		&H7B
VK_F13		&H7C
VK_F14		&H7D
VK_F15		&H7E
VK_F16		&H7F
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 3.26 ersichtlich. Ist das Ergebnis wieder der Hexadezimalwert &HF000&, dann ist die Taste gedrückt.

**Listing 3.26** 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 &HFF00& 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
--------	---	------------------

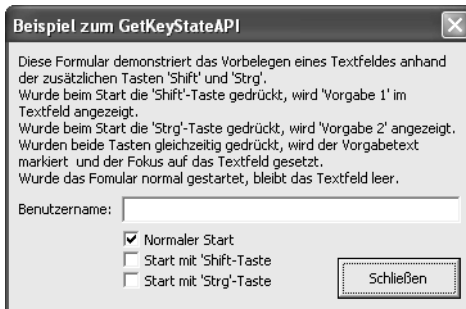
Als Beispiel wird der Aufruf der Userform frmGetKeyStateAPI in der Beispieldatei *Bsp03\_1.doc* verwendet. In Listing 3.27 wird die Abfrage evtl. gedrückter Tasten in einer Select Case-Anweisung geprüft, wobei mit Hilfe der Funktion funcIsTasteGedrückt aus Listing 3.26 der Tastenstatus ermittelt wird. Im ersten Case-Abschnitt wird geprüft, ob beide Tasten, und in den beiden nächsten Abschnitten, ob die beiden einzeln gedrückt wurden.

**Listing 3.27** 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
```

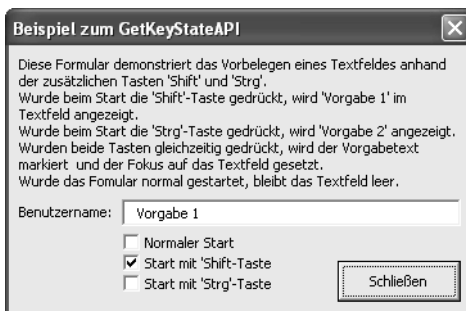
Öffnen Sie im Visual Basic-Editor mit einem Doppelklick die Userform frmGetKeyStateAPI. Starten Sie anschließend die Userform entweder über die Taste  oder über den Menüpunkt *Ausführen/ Sub/Userform ausführen*. Es wird die in Abbildung 3.9 gezeigte Userform mit einem leeren Feld *Benutzername* angezeigt.

Abbildg. 3.9 Normaler Start des Benutzerformulars ohne zusätzlich gedrückte Taste



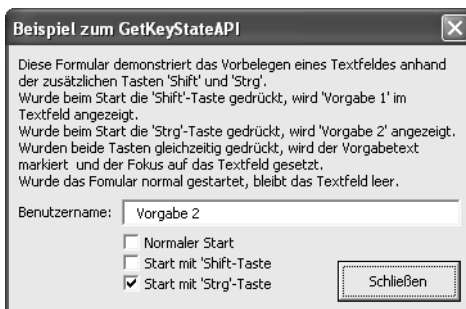
Wenn Sie beim Start zusätzlich die ☐-Taste gedrückt halten, wird das Benutzerformular mit dem Benutzernamen »Vorgabe 1« angezeigt. Wenn Sie die ☐-Taste gedrückt halten, wird der Benutzername »Vorgabe 2« angezeigt.

Abbildg. 3.10 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. 3.11 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\Kap03\Bsp03_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 126) wurden Ihnen API-Funktionen vorgestellt, mit denen Sie relativ einfach mehrere Pfade (Seite 127 ff.) und Dateinamen (Seite 130) syntaktisch korrekt miteinander kombinieren und sogar Dateien mit Erweiterungen ergänzen können, sofern diese noch keine Erweiterung besitzen (Seite 130).
- Des Weiteren wurde gezeigt, wie Sie eine Datei über ihre Dateiendung aufrufen können (Seite 131).
- Es wurden Funktionen vorgestellt, um gezielt auf Einträge in der Windows Registry zuzugreifen (Seite 135 ff.), und mit denen Sie Registry-Einträge und -Werte ermitteln können (Seite 135).
- Ein weiterer Schwerpunkt dieses Kapitels war das Auslesen von und Schreiben in INI-Dateien mittels API-Funktionen (Seite 140 ff.).
- Weitere Beispiele haben Ihnen gezeigt, wie Sie den Namen des gerade angemeldeten Benutzers ermitteln (Seite 145) und die Verarbeitung von Makros für eine bestimmte Zeit unterbrechen können (Seite 147).
- Anhand eines weiteren Beispiels wurde eine API-Funktion vorgestellt, um eine beliebige Wave-Datei abzuspielen (Seite 148 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 151).

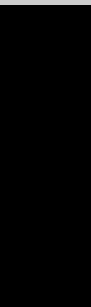


## Teil B

# Das Objektmodell von Word

### In diesem Teil:

<b>Kapitel 4</b>	Überblick über die Arbeit mit Objekten	161
<b>Kapitel 5</b>	Grundlagen des Word-Objektmodells	173
<b>Kapitel 6</b>	Professionelle Dokumente	241
<b>Kapitel 7</b>	Word und Datenstrukturen	339
<b>Kapitel 8</b>	Ereignisse in Word	447



## Kapitel 4

# Überblick über die Arbeit mit Objekten

### In diesem Kapitel:

Arbeiten mit Objekten	162
Auflistung von Objekten	167
Auflistung bearbeiten	168
Zusammenfassung	171

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 folgenden Abschnitte vermitteln zunächst einen kurzen Einblick in die Arbeit mit Objekten. Das eigentliche Objektmodell wird anschließend in den Kapiteln 5, 6 und 7 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. Sie wurden in Kapitel 1 schon vorgestellt; nachfolgend eine kurze Zusammenfassung.



Online-Hilfe

- Die *Online-Hilfe* wurde bereits in Kapitel 1 im Abschnitt »Die Objektmodell-Hilfe – eine versteckte Schatzkammer« vorgestellt. Aus unserer Sicht ist der wichtigste Hinweis auf jeder Hilfe-seite 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. 4.1

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 1 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 2 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 9.

## **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 bzw. 12.0 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 4.1 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")
```

Zudem lauert die Gefahr eines »Memory leak«, da der Platz im Speicher sofort beansprucht, jedoch noch keinem Objekt zugewiesen wurde. Falls das Objekt nicht erstellt wird, kann der Speicherplatz auch nie freigestellt werden, was im ungünstigsten Fall zu einem Absturz führt.

**Listing 4.1** Das Erzeugen der Objektvariablen und das Anlegen der Instanz wurden getrennt

```
Sub ExcelObjekt()  
    'Verweis auf Microsoft Excel 11.0 bzw. 12.0 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 beiden 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 4.1 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 4.2 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 4.2** Indexnummer der aktuellen Tabelle ermitteln

```
Sub IndexDerTabelleErmitteln()
    Dim rng As Word.Range
    Dim lngIndex As Long

    If Selection.Information(wdWithInTable) Then
```

**Listing 4.2** Indexnummer der aktuellen Tabelle ermitteln (*Fortsetzung*)

```

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

For...Next  
For  
Each...  
Next

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 2 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 4.3 dargestellt.

**Listing 4.3** 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 4.4 dargestellt.

Listing 4.4 Auflistung aller Elemente einer Auflistung mittels einer *For Each...Next*-Schleife

```

Sub Demo_ForEach_Next()
    Dim doc As Word.Document
    Dim strText As String

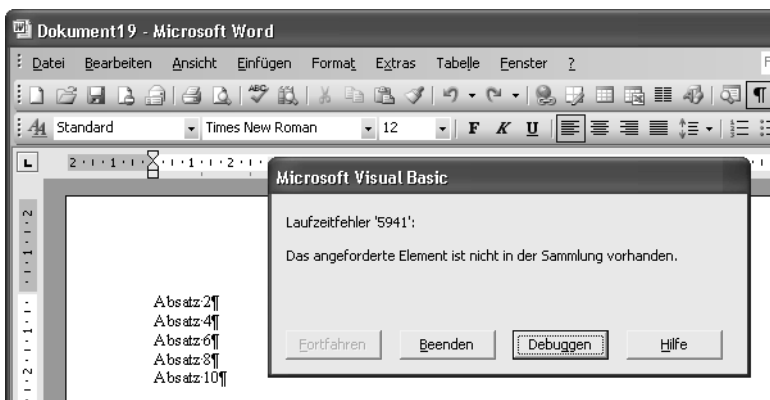
    For Each doc In Application.Documents
        strText = strText & doc.Name & vbCrLf
    Next doc

    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 4.5 erzeugen eben diesen Fehler.

Abbildg. 4.2 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 4.2, 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 4.5** 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 4.6 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 4.6** 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 4.7 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 4.7 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 4.7** 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.



Die aufgeführten Codesequenzen finden Sie in der Beispieldatei *Bsp04\_01.doc* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap04*.

## 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 162) und dieses einer Variablen zugewiesen werden kann (Seite 163).
- Es wurde aufgezeigt wie eine Auflistung von Objekten bearbeitet und wie auf deren einzelne Objekte zugegriffen wird (Seite 167 ff.).





## Kapitel 5

# Grundlagen des Word-Objektmodells

### In diesem Kapitel:

Das <i>System</i> -Objekt	174
Die Anwendung: Das <i>Application</i> -Objekt	177
Die gegenwärtige Markierung: <i>Selection</i> und ähnliche Objekte	186
Der Kern der Sache: Das <i>Document</i> -Objekt	189
Dokumentvorlagen: Das <i>Template</i> -Objekt	203
Mit Bereichen arbeiten: Das <i>Range</i> -Objekt	207
Die Nadel im Heuhaufen: <i>Find/Replace</i> einsetzen	221
Zusammenfassung	240

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 Objekt-namen 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 ...)

Die folgenden drei Kapitel sind daher kein Wiederkauen der VBA-Hilfedateien. 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 sowie in Word 2007 neue Funktionalität vor und erläutern anhand kurzer Code-Beispiele deren programmtechnischen Einsatz.



Ausführlichere Beispiele zum Zusammenspiel mehrerer Teile des Objektmodells finden Sie im Bonusteil »Lösungen« (Kapitel I bis VIII) auf der CD-ROM zum Buch.



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 C 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 3).





Damit lassen sich Informationen zu Rechner- und Bildschirmauflösung, zur Verfügung stehenden Prozessoren und Speicherplatz abfragen. Eigenschaften von besonderem Interesse stellen wir hier kurz vor.

**CountryRegion.** 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 5.1 aufgelistet.

Tabelle 5.1 Die verschiedenen Mauszeigerformen des `Cursor`-Objekts

WdCursor-Wert	Mauszeigerform
<code>wdCursorIBeam</code>	
<code>wdCursorNormal</code>	
<code>wdCursorNorthwestArrow</code>	
<code>wdCursorWait</code>	

#### 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 15.

**LanguageDesignation.** 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)«.

**OperatingSystem** sowie **Version.** Die Eigenschaft `OperatingSystem` liefert den Namen des Betriebssystems, die Eigenschaft `Version` und die zugehörige Versionsnummer. Allerdings werden Sie nicht »Windows 2000«, »Windows XP« oder »Windows Vista« erhalten. Diese Versionen gehören zur Familie »Windows NT«, wobei Windows 2000 der Version 5.0, Windows XP der Version 5.1, Windows Server 2003 der Version 5.2 und Windows Vista der Version 6.0 entspricht.

**PrivateProfileString.** Diese ist die nützlichste aller System-Eigenschaften. Damit können Einstellungen in der Registry (oder aber auch in einer INI-Datei) beliebig gelesen und geschrieben werden (im Gegensatz zur `ProfileString`-Eigenschaft, die nur den Registry-Abschnitt für Anwendungen

anspricht: HKEY\_CURRENT\_USER\Software\Microsoft\Office\<Version>\Word). So greifen Sie beispielsweise direkt auf die Einstellungen unter *Regions- und Sprachoptionen* der Systemsteuerung zu, um das Dezimal- und Listen-Trennzeichen 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* bzw. drücken Sie unter Windows Vista die Tastenkombination  + . Tippen Sie in das Feld *Öffnen* »regedit« (ohne Anführungszeichen) ein und bestätigen Sie dann mit *OK*.

Die Prozedur in Listing 5.1 zeigt, wie diese Einstellungen abgefragt und verwendet werden, um eine Zahl nach deutschem Muster zu formatieren. Das Resultat sehen Sie in Abbildung 5.1.

**Abbildg. 5.1** Die Tausender- und Dezimaltrennzeichen für die gegenwärtige Ländereinstellung wurden aus der Registry ermittelt und mit den deutschen ersetzt



**Listing 5.1** Einstellungen aus der Registry lesen

```
Private Const strMSGITEL As String = "Zahl formatieren"

Sub ZahlMitSystemEinstellungenFormatieren()
    Dim strDezimalTrennzeichen As String
    Dim strGruppierungSymbol 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")
        strGruppierungSymbol = .PrivateProfileString(FileName:="", _
            Section:="HKey_Current_User\Control Panel\International", _
            Key:="sThousand")
    End With
    If IsNumeric(strEingabe) Then
        '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
    End If
End Sub
```

## Listing 5.1 Einstellungen aus der Registry lesen (Fortsetzung)

```

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, strMSGTITEL
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 *Bsp05\_01\_System.doc* finden Sie auf der CD-ROM im Ordner *\Beispiele\Kap05*.

**HINWEIS**

Mehr zum Thema Registry, INI-Dateien und das Speichern von Benutzer- und Anwendungseinstellungen enthält Kapitel 13.

## 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 Anwen-

dungs-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 C 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

**ScreenUpdating.** 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.

**DisplayAlerts.** 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.

**AutomationSecurity.** 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.

**PathSeparator.** 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 Doppelpunkt (:) – 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 das englische Word haben Firmen mit Lizenzverträgen ab der Version 2000 bis und mit 2003 die Möglichkeit, das Office Multilingual User Interface Pack zu erwerben. Damit können Menüeinträge, Dialogfelder und die Hilfe in der ausgewählten Sprache angezeigt werden. Ab Office 2007 basieren alle Sprachversionen auf dem gleichen Quellcode, so dass der Startpunkt nicht zwingend eine englische Version voraussetzt. Zudem stehen die Schnittstellen als Teil eines »Microsoft Office 2007 Multi Language Packs« allen Kunden zur Verfügung. Mehr Informationen erhalten Sie auf <http://office.microsoft.com/de-de/suites/FX102113661031.aspx>.

**Options.** Mittlerweile enthält Word 2007 um die 230 Eigenschaften zum `Options`-Objekt (in Word 2003 sind es etwa 120). In Word 2003 und früher befindet sich für jede Option ein Eintrag im Menüfeld *Extras/Optionen*, *Extras/AutoKorrektur-Optionen/AutoFormat* oder *AutoFormat während der Eingabe*. In Word 2007 sind die meisten Einstellungen in den Kategorien der *Word-Optionen* unter der *Office*-Schaltfläche zu finden. 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 den *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. In Word 2007 ist diese Trennung etwas deutlicher geworden.

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.



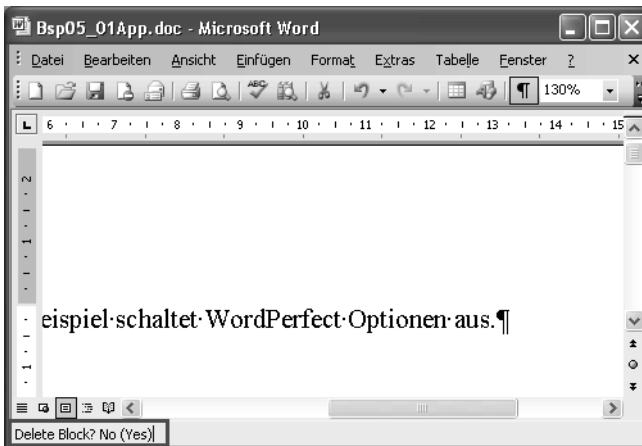
Wenn Sie für diese Aufgabe den Makrorekorder hinzuziehen, denken Sie daran, dass er alle Optionen einer Registerkarte (oder in Word 2007 einer Kategorie) aufzeichnet und nicht nur diejenigen, deren Einstellungen geändert wurden. Dies bedeutet:

- 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 5.2 bzw. Listing 5.3, 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 5.2 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. 5.2 Die Aufforderung »Delete Block?« ist die Auswirkung einer eingeschalteten WordPerfect-Option



Listing 5.2 Die WordPerfect-Optionen ausschalten

```
Sub DeleteBlockAusschalten()
    Application.Options.WPDocNavKeys = False
    Application.Options.WPHelp = False
End Sub
```

**Listing 5.3** C#-Version: Eine bestehende Verbindung zur Word-Anwendung über *wdApp* wird angenommen



```
private void DeleteBlockAusschalten_CS()
{
    wdApp.Options.WPDocNavKeys = false;
    wdApp.Options.WPHelp = false;
}
```

**DefaultFilePath-Option.** 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 5.2 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 5.2** 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> .

Tabelle 5.2 Pfadnamen zu Anwendungs-relevanten Ordnern (Fortsetzung)

WdDefaultFilePath-Enum	Wert	Beschreibung
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.)
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.

**WordBasic.** 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 D 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 5.3 aufgelistet. Ein Beispiel sehen Sie in Listing 5.4 bzw. in Listing 5.5, dessen Resultat in Abbildung 5.3 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 5.3** Nützliche *WordBasic*-Befehle

WordBasic-Befehl	Beschreibung
<code>WordBasic.SelectSimilarFormatting</code>	In Word 2002 wurde eine Funktionalität eingeführt, die es dem Benutzer ermöglicht, nicht zusammenhängende Dokumentbereiche gleichzeitig zu markieren ( <code>[Strg]</code> -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.  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.
<code>WordBasic.DisableAutoMacros</code>	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 <b>Document</b> - und <b>Application</b> -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«. <b>WordBasic.DisableAutoMacros 1</b> schaltet sie aus; <b>WordBasic.DisableAutoMacros 0</b> lässt die Ausführung zu.
<code>WordBasic.SortArray aArray()</code>	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.
<code>WordBasic.FilePrintSetup</code>	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.

Tabelle 5.3 Nützliche WordBasic-Befehle (Fortsetzung)

WordBasic-Befehl	Beschreibung
WordBasic.FileNameInfo\$()	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.
WordBasic.MailMergeUseOutlookContacts	Verbindet ein Seriendruckdokument mit den Outlook-Kontakten als Datenquelle in Word 2007.
WordBasic.CreateCommonFieldBlockFromSel()	Erstellt aus der Markierung einen Schnellbaustein, der auf dem Ribbon »Einfügen/Schnellbausteine« angezeigt wird. Einziger möglicher Parameter ist <b>Description</b> zur Angabe einer Beschreibung (siehe in Kapitel 6 den Abschnitt »Schnellbausteine«)
* 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> ).	

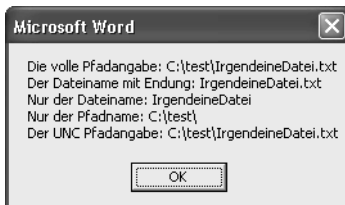


2007

**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. 5.3 Ergebnisse der Funktion WordBasic.FileNameInfo\$



Listing 5.4 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 5.5** 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-Pfangabe: {4}",
        oFI1.ToString(), oFI3.ToString(), oFI4.ToString(), oFI5.ToString(), oFI6.ToString()));
    oWdBsc = null;
}

private BindingFlags InvMethod = BindingFlags.InvokeMethod;
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 *Bsp05\_01App.doc* finden Sie auf der CD-ROM im Ordner *\Beispiele\Kap05*.

## Die gegenwärtige Markierung: *Selection* und ähnliche Objekte

Während unserer Arbeit als MVPs in den Newsgruppen und Foren 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 zugewiesen 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 Codefragment 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 5.4 aufgelistet.

**Tabelle 5.4** Die Art der Markierung herausfinden

WdSelectionType-Enum	Wert	Beschreibung
wdSelectionBlock	6	Ein Rechteck von Text wurde mit Festhalten der <code>[Alt]</code> -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

**SelectCurrent.** Gelegentlich möchte man einen Textblock mit einer bestimmten Formatierung finden. Die Funktionalität *Suchen* (Find, mehr dazu im Abschnitt »Die Nadel im Heuhaufen: Find/Replace einsetzen« in diesem Kapitel) kann das, aber meistens erstreckt sich der gefundene Bereich nicht über mehrere Absätze hinweg. Die Methoden `SelectCurrentAlignment` (Absatzausrichtung), `SelectCurrentColor` (Schriftfarbe), `SelectCurrentFont` (Schriftart), `SelectCurrentIndent` (Absatzzeinzug), `SelectCurrentSpacing` (Zeilenabstand), `SelectCurrentTabs` (Tabstopps) unterliegen dieser Beschränkung nicht. Die Markierung wird erweitert bis zu der Stelle, wo die bestimmte Formatierung aufhört.

**wdLine: zeilenweise.** Mit den verschiedenen Move-Methoden wird ein Bereich oder eine Markierung vergrößert oder verschoben. Über das Argument `Unit` wird 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` (Tabellezeile) oder `wdTable` (Tabelle). Zusätzlich zu den unterstützten Argumenten, wenn Move mit dem Selection-Objekt benutzt wird, gibt es noch `wdLine` (Textzeile). Beispielsweise erweitert die Codezeile

```
Selection.MoveEnd wdLine, 3
```

die Markierung über weitere zwei Zeilen (so dass sie sich über drei Zeilen erstreckt), wie in Abbildung 5.4 ersichtlich. Vor Ausführung der obigen Codezeile blinkte die Einfügemarke am Anfang der abgebildeten Markierung.



Das Ergebnis von `Selection.MoveEnd wdLine, 3`

Franz jagt im komplett verwahrlosten Taxi quer durch Bayern. Franz jagt im komplett  
verwahrlosten Taxi quer durch Bayern. Franz jagt im komplett verwahrlosten Taxi quer durch  
Bayern. Franz jagt im komplett verwahrlosten Taxi quer durch Bayern. Franz jagt im  
komplett verwahrlosten Taxi quer durch Bayern  
Franz jagt im komplett verwahrlosten Taxi quer durch Bayern. Franz jagt im komplett  
verwahrlosten Taxi quer durch Bayern. Franz jagt im komplett verwahrlosten Taxi quer durch  
Bayern. Franz jagt im komplett verwahrlosten Taxi quer durch Bayern. Franz jagt im  
komplett verwahrlosten Taxi quer durch Bayern

## Der Kern der Sache: Das *Document*-Objekt

Word ist eine Anwendung zur Textverarbeitung und dient primär dem Erstellen und Bearbeiten von Dokumenten. Hierzu stellt die Umgebung viele Werkzeuge zur Verfügung, aber letztlich dreht sich alles um Dokumente. Es überrascht also nicht, dass das Objektmodell ein Document-Objekt aufweist und dieses häufig in Word-Code auftaucht.

Der Makrorekorder verwendet ausschließlich das Objekt `ActiveDocument` (aktuell aktives Dokument). Das Problem hiermit ist, dass durch eine Handlung ein anderes Dokument als erwartet zum aktuellen (aktiven) werden könnte. Und plötzlich manipuliert der VBA-Code das falsche Dokument.

Folglich sollten Sie diesen Ausdruck in Ihrem Code möglichst durch eine Objektvariable des Typs Document ersetzen. Danach, egal welches Dokument in der Anwendung aktiv ist, spricht VBA immer das korrekte an. Falls eine Prozedur mit dem gegenwärtigen Dokument arbeiten soll, benutzen Sie folgendes Muster:

```
Dim doc As Word.Document
Set doc = ActiveDocument
```



```
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 5.6 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 Funktionsabrufe genau mit den Definitionen der Objektbibliothek übereinstimmen.

**Listing 5.6**

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 + "\\Bsp05_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;
}

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.

**ComputeStatistics.** 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 5.5) legt fest, was gezählt werden soll. Die allgemeine Syntax lautet:

```
ComputeStatistics(wdStatistic-Wert, [IncludeFootnotesAndEndnotes As Boolean])
```

Tabelle 5.5 Die Enumeration von *WdStatistic*

WdStatistic-Enum	Beschreibung
<code>wdStatisticCharacters</code>	Die Anzahl Zeichen, ohne Leerräume
<code>wdStatisticCharactersWithSpace</code>	Die Anzahl Zeichen; Leerräume werden mitgezählt
<code>wdStatisticFarEastCharacters</code>	Die Anzahl Zeichen in einer asiatischen Umgebung
<code>wdStatisticLines</code>	Die Anzahl Zeilen
<code>wdStatisticPages</code>	Die Anzahl Seiten
<code>wdStatisticParagraphs</code>	Die Anzahl Absätze
<code>wdStatisticWords</code>	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 5.6.

Tabelle 5.6 Die Enumeration von *WdDocumentType*

WdDocumentType-Enum	Wert	Beschreibung
<code>wdTypeDocument</code>	0	Dokument
<code>wdTypeTemplate</code>	1	Vorlage
<code>wdTypeFrameset</code>	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, FullName, Path.** 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:

```
Dim doc1 as Word.Document, doc2 as Word.Document
Set doc1 = ActiveDocument
Set doc2 = Documents.Add
doc2.SaveAs doc1.Path & Application.PathSeparator & "Doc2.doc"
```

**AttachedTemplate.** 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.

Über diese Eigenschaft kann ein Dokument auch mit einer anderen Vorlage verbunden werden, um ihm die darin enthaltenen Symbolleisten, 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 5.7 (C#-Version in Listing 5.8) 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 5.7** 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
```

**Listing 5.7** Ein Dokument für die Übermittlung außer Haus vorbereiten und mit der *Normal.dot* verbinden (Fortsetzung)

```

        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 5.8** 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);
    }
}

private void DokMitNormalVerbinden_CS(wd.Document doc)
{
    object objNormalTemplate = (object) wdApp.NormalTemplate;
    doc.set_AttachedTemplate(ref objNormalTemplate);
    try
    {doc.Save();}
    catch {}
}

```



Die Beispieldatei *Bsp05\_01\_Document.doc* finden Sie auf der CD-ROM im Ordner *\Beispiele\Kap05*.

**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 Dateart zu speichern – beispielsweise als Vorlage, Webseite oder Textdatei – legen Sie für den Parameter FileFormat ein gültiges SaveFormat fest in Form eines WdSaveFormat-Werts oder eines FileConverter-Objekts.
- 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 5.7 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 5.9 bzw. Listing 5.10. 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 5.9** Die Eigenschaft Saved prüft den Bearbeitungszustand eines Dokuments oder legt ihn fest

```
Sub DokSpeichern()
    Dim doc As Word.Document

    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 5.10** 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 einzublenden. Dadurch kann die Benutzerhandlung direkt ausgewertet werden. Der Umgang mit den internen Dialogfeldern von Word ist in Kapitel 15 beschrieben.



Die Beispieldatei *Bsp05\_01\_Document.doc* finden Sie auf der CD-ROM im Ordner *\Beispiele\Kap05*.



2007

Eine weitere Neuerung in Office 2007 ist das Speichern oder Veröffentlichen einer Datei im Format PDF (Portable Document Format) und XPS (XML Paper Specification).

Voraussetzung dazu ist allerdings, dass das entsprechende Add-In »Add-In für Microsoft Office 2007: »Speichern unter – PDF oder XPS« installiert wird.

**HINWEIS**

Sie können das Add-In nach erfolgreicher Gültigkeitsprüfung von der folgenden Webseite herunterladen und installieren:

<http://www.microsoft.com/downloads/details.aspx?displaylang=de&FamilyID=4d951911-3e7e-4ae6-b059-a2e79ed87041>

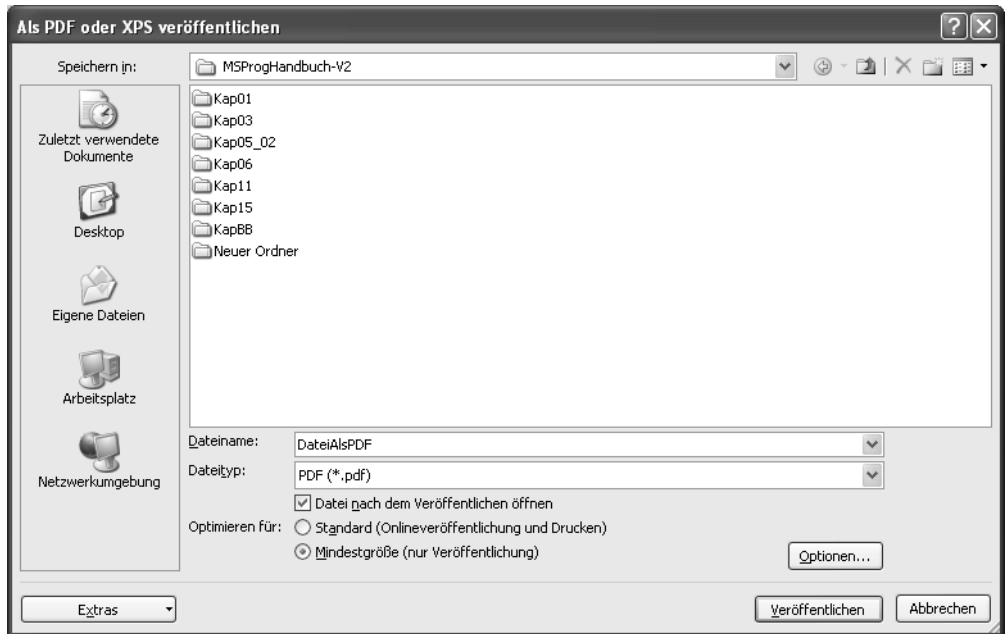
Nach der Installation stehen die beiden neuen Dateitypen PDF und XPS unter Office 2007 zur Verfügung.

Das Speichern aus VBA heraus erfolgt dann über die neuen wdFileFormat-Konstanten wdFormatPDF und wdFormatXPS.

```
ActiveDocument.SaveAs FileName:="C:\test\DateiAlsPDF.pdf", FileFormat:=wdFormatPDF
```

Wenn Sie über den Befehl *Speichern unter* der *Office*-Schaltfläche das Dokument als PDF speichern möchten, werden Ihnen eine Reihe von Optionen angeboten, mit denen Sie den Export als PDF-Datei beeinflussen können.

Abbildg. 5.5 Datei als PDF-Datei speichern (veröffentlichen)



Diese Optionen stehen beim SaveAs-Befehl nicht als Parameter zur Verfügung. Um diese dennoch im VBA-Code setzen zu können, müssen Sie die Methode `ExportAsFixedFormat` verwenden, wie in Listing 5.11

Listing 5.11 Datei aus VBA als PDF-Datei veröffentlichen

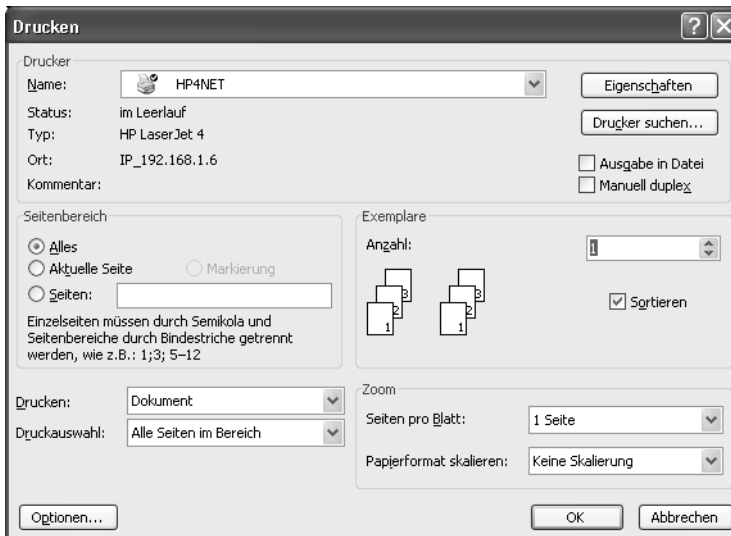
```
ActiveDocument.ExportAsFixedFormat OutputFileName:=
"C:\test\DateiAlsPDF.pdf", ExportFormat:=wdExportFormatPDF,
OpenAfterExport:=True, OptimizeFor:=wdExportOptimizeForOnScreen, Range:= _
wdExportAllDocument, From:=1, To:=1, Item:=wdExportDocumentContent, _
IncludeDocProps:=True, KeepIRM:=True, CreateBookmarks:= _
wdExportCreateHeadingBookmarks, DocStructureTags:=True, _
BitmapMissingFonts:=True, UseISO19005_1:=True
```

Die einzelnen Parameter der `Document.ExportAsFixedFormat`-Methode finden Sie in der Onlinehilfe ausführlich erläutert, so dass an dieser Stelle nicht weiter darauf eingegangen wird.

**PrintOut.** Die `PrintOut`-Methode stellt den Großteil der Funktionalität des Dialogfelds *Datei/Drucken* (Abbildung 5.6) (in Word 2007 der Befehl *Drucken* der Office-Schaltfläche) 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)
```



Abbildg. 5.6 Das Dialogfeld *Datei/Drucken*

Word verfügt über 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.

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 jeden Fall raten wir dringend, den Druck im Hintergrund während des Code-Ablaufs zu unterbinden, indem Sie das `Background`-Argument oder die Option `PrintBackground` auf `False` festlegen.

In Listing 5.12 bzw. in Listing 5.13 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 5.12** Drucken im Hintergrund unterbinden. Die Anwendereinstellung wird am Schluss wieder hergestellt

```
Sub DokAusdrucken()
    Dim doc As Word.Document
    Dim lDruckImHintergrund As Long

    lDruckImHintergrund = Application.Options.PrintBackground
    Application.Options.PrintBackground = False
    Set doc = ActiveDocument
    doc.PrintOut
    Application.Options.PrintBackground = lDruckImHintergrund
    doc.Close SaveChanges:=wdSaveChanges
End Sub
```

**Listing 5.13** 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, ref objMissing, ref objMissing,
        ref objMissing, ref objMissing);
}
```

Die Argumente `Range`, `From`, `To` und `Pages` 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 5.7 aufgelistet sind.

**Tabelle 5.7** Die `WdPrintOutRange`-Werte

WdPrintOutRange-Enum	Wert	Beschreibung
<code>wdPrintAllDocument</code>	0	Das gesamte Dokument; entspricht der Option <i>Alles</i> im Dialogfeld
<code>wdPrintCurrentPage</code>	2	Die aktuelle Seite
<code>wdPrintFromTo</code>	3	Die angegebenen Seiten von <i>n</i> bis <i>m</i> drucken

Tabelle 5.7 Die WdPrintOutRange-Werte (Fortsetzung)

WdPrintOutRange-Enum	Wert	Beschreibung
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 Word 2003 in den Word-Hilfedateien unter »Drucken eines Dokuments« im Abschnitt »Drucken bestimmter Seiten und Abschnitte«. In Word 2007 steht diese Information im Dialogfeld *Drucken*.

Ein Beispiel, wie die erste Seite jedes Dokumentabschnitts gedruckt wird, sehen Sie in Listing 5.14 bzw. Listing 5.15. 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 5.14 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 5.15 die PrintOut-Methode in eine getrennte Prozedur ausgelagert wird, und vergleichen Sie *DokDrucken* mit der Prozedur gleichen Namens in Listing 5.13. 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 5.15** Die C#-Version von Listing 5.14


```
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;
    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 Seitennummernangaben 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.

Unterhalb des Kastens *Seitenbereich* befinden sich zwei Dropdownfelder, die oft nicht beachtet werden. Im ersten, das dem Argument *Item* entspricht, wird bestimmt, was anstelle des Dokuments ausgedruckt ist: AutoText-Einträge, Änderungen und Kommentare, Tastaturbelegungen oder Formatvorlagen. Die gültigen Werte sind in Tabelle 5.8 aufgelistet.

**Tabelle 5.8** 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

Tabelle 5.8 Werte für das Argument *Item* (Fortsetzung)

WdPrintOutItem-Enum	Wert	Beschreibung
wdPrintDocumentWithMarkup	7	Druckt den Dokumenttext mit Änderungen und Kommentaren in Sprechblasen im Rand (ab Word 2002)
wdPrintEnvelope	6	Soll einen am Dokumentanfang angefügten Umschlag drucken, verursacht aber einen Laufzeitfehler. Ist auch im Dialogfeld seit Word 2000 nicht vorhanden.
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 (ab Word 2002)
wdPrintProperties	1	Druckt die im Dokument gespeicherten Dokumenteigenschaften
wdPrintStyles	3	Druckt die im Dokument verwendeten Formatvorlagen

Im zweiten Dropdownfeld, 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.

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* (unter Windows Vista *Generic*).
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 5.16 bzw. das Listing 5.17 enthält ein Code-Beispiel, das die AutoText-Einträge der aktuellen Umgebung in eine Textdatei exportiert.

Listing 5.16 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
```

**Listing 5.16** Ein Word-Dokument in eine Datei ausgeben, anstatt an den Drucker (*Fortsetzung*)

```
Set doc = ActiveDocument
strPfad = "C:\test\TastaturBelegung_" & doc.Name & Format(Date, "mmdd") _
    & "_" & Format(Time, "hh.ss") & ".prn"
strAktuellerDrucker = Application.ActivePrinter
Application.ActivePrinter = "Generic / Text Only"
Application.Options.PrintBackground = False
doc.PrintOut Item:=wdPrintAutoTextEntries, Append:=False, _
    OutputFileName:=strPfad, PrintToFile:=True
Application.ActivePrinter = strAktuellerDrucker
Application.Documents.Open strPfad
End Sub
```

**Listing 5.17** 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 pfad = String.Format("C:\\test\\AutoTextDrucken {0}{1}_{2}.prn",
        doc.Name, dt.ToString("MMMdd"), dt.ToString("hh.ss"));
    string aktuellerDrucker= wdApp.ActivePrinter;
    wdApp.ActivePrinter = "Generic / Text Only";
    wdApp.Options.PrintBackground = false;
    bool append = false;
    wd.WdPrintOutItem printOutItem = wd.WdPrintOutItem.wdPrintAutoTextEntries;
    bool printToFile = true;
    DokDrucken(doc, printOutItem, append, pfad, printToFile);
    wdApp.ActivePrinter = aktuellerDrucker;
    WordDokumentOeffnen_CS(wdApp, pfad);
}
```

#### **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 brauchen 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« unter <http://pubs.logicalexpressions.com/Pub0009/LPMArticle.asp?ID=101>.



Die Beispieldatei *Bsp05\_01\_Document.doc* finden Sie auf der CD-ROM im Ordner `\Beispiele\Kap05`.

# Dokumentvorlagen: Das *Template*-Objekt

Dokumentvorlagen sind ein wichtiger Bestandteil von Word. Zum einen dienen sie als Schablone für neue Dokumente eines bestimmten Typs und enthalten beispielsweise Briefkopf, Logo, Kopf- und Fußzeilen sowie Standardtexte. Zum anderen 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.

Programmetechnisch 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 5.7) 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 Word 2007 befindet sich die Schaltfläche *Vorlagen* auf der Registerkarte *Entwicklertools*.)

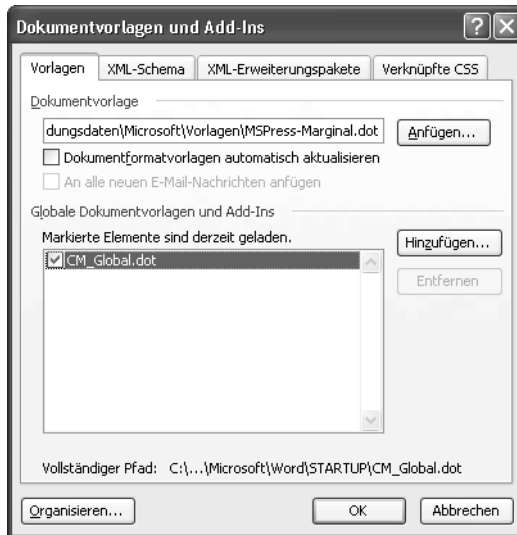
In zweiter Linie dienen Vorlagen als Behälter nützlicher Werkzeuge, die eine beliebige Kombination von Makros, Symbolleisten (oder im Fall von Word 2007 Steuerelemente in der Multifunktionsleiste und der Symbolleiste für den Schnellzugriff), AutoText-Einträgen, Bausteine (in Word 2007) und Tastaturbelegungen 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 5.7) geladen werden, indem man sie in der Liste aktiviert (oder hinzufügt und aktiviert). (In Word 2007 befindet sich die Schaltfläche *Vorlagen* auf der Registerkarte *Entwicklertools*.)

**Abbildg. 5.7** Dokumentspezifische und globale Add-Ins werden über dieses Dialogfeld verwaltet


**HINWEIS**

Add-Ins werden in Kapitel 14 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 5.18 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 5.18** `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 5.19 Die C#-Version



```
private void btnListing5_19_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 5.20 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 5.20 Ist eine Vorlage nicht vorhanden, kann sie in den Speicher als Add-In geladen werden

```
Sub MakroInAddinAusfuehren()
    Dim tmp1 As Word.Template
    Dim strAddinName As String
    Dim strMakroName As String
    Dim bAddinGeladen As Boolean
    Dim adin As Word.AddIn

    strAddinName = ThisDocument.Path & "\Bsp05_02_Template.dot"
    strMakroName = "MakroInAddin"
    bAddinGeladen = False
    For Each tmp1 In Application.Templates
        If tmp1.FullName = 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 5.21 Die C#-Version**


```
private void Listing5_21_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 + "\\Bsp05_02_Template.dot");
    string addinName = pfad;
    string makroName = "MakroInAddin";
    bool addinGeladen = false;
    foreach (wd.Template tmp1 in wdApp.Templates)
    {
        if (tmp1.FullName == addinName)
        {
            addinGeladen = true;
            RunWordMakro(makroName);
        }
    }
    if (! addinGeladen)
    {
        //Die Vorlage als Add-In laden
        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,
        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 5.9 aufgelistet.

**Tabelle 5.9** 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.

**OpenAsDocument.** 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 finden Sie in Kapitel 7).

**Listing 5.22** 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 5.23** Die C#-Version



```
private void Listing5_23_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 *Bsp05\_01\_Template.doc* mit den Code-Beispielen sowie *Bsp05\_02\_Template.dot* mit dem Makro *MakroInAddin* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap05*.

## 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 Doku-

ment 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 Endpunktswerten 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 festlegen.** 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.Content
```



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 ausfindig 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).Range.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 wichtigsten Eigenschaften und Methoden.

## Einen Bereich bearbeiten

**Text und Formatierung.** 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 bis Word 2003 unter dem Menü *Format* in der Benutzerschnittstelle. In Word 2007 sind die meisten in der Registerkarte *Start* untergebracht.

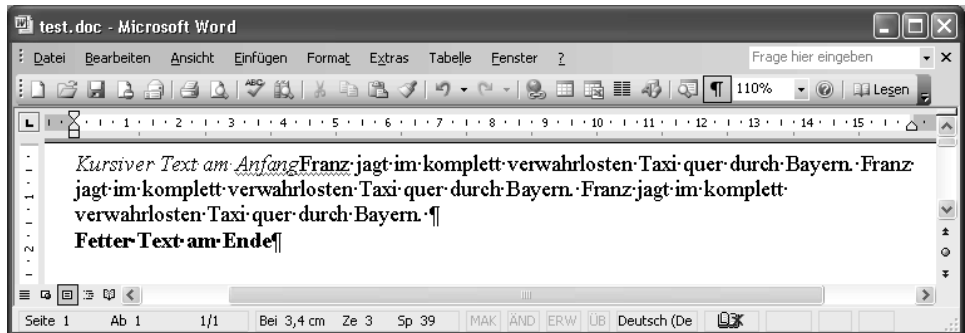
**Collapse.** Wenn Sie ein Dokument, das schon Text enthält, öffnen und den Inhalt einem Range zuweisen, umfasst der Range den gesamten Inhalt des Dokument-Haupttextes (MainTextStory). Wird diesem Range dann Text zugewiesen, ersetzt er den im Dokument bereits vorhandenen Text.

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 5.24 bzw. in Listing 5.25 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. 5.8

Das Resultat von Listing 5.24



Listing 5.24

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 & Application.PathSeparator & "Bsp05_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"
```

**Listing 5.24** Den Bereich (*Range*) auf einen Punkt verkleinern, bevor Text eingefügt wird (*Fortsetzung*)

```
rng.Italic = True
End Sub
```

**Listing 5.25** Hier ist Code aus der Klasse für den Abschnitt über das *Document*-Objekt integriert, um das Dokument zu öffnen



```
private void Listing5_25_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 + "\\Bsp05_01Range_Test.doc");
    wd.Document doc = Kap05.Kap05Document.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
    object objStartPunkt = (object) wd.WdCollapseDirection.wdCollapseStart;
    rng.Collapse(ref objStartPunkt);
    rng.Text = "Kursiver Text am Anfang";
    rng.Italic = -1;
}
```



Die Beispieldateien *Bsp05\_01Range.doc* und *Bsp05\_01Range\_Test.doc* finden Sie auf der CD-ROM im Ordner *\Beispiele\Kap05*.

**Insert-Methoden.** 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 in Kapitel 6) 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 5.10 aufgelistet.

**Tabelle 5.10** 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-Methoden.** 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

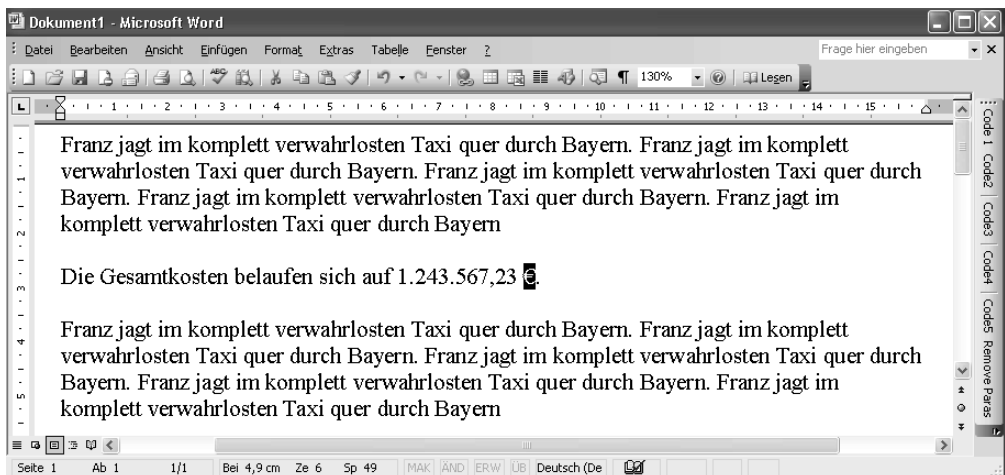
Denken Sie daran: Wenn eine MoveEnd-Methode mit einem positiven Unit-Wert verwendet wird, wird der Bereich erweitert; mit einem negativen Wert wird er verkleinert. Überschneiden sich dabei die End- und Startpunkte, wird der Bereich auf einen Punkt verkleinert und bleibt ein Punkt.

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 5.9). Jetzt soll, wie in Listing 5.26 vorgestellt, die voranstehende Zahl (falls vorhanden) auch in den Bereich aufgenommen werden.

Abbildg. 5.9 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 5.10).

**Abbildg. 5.10** ... und der Bereich mit der *MoveStartWhile*-Methode erweitert, bis er die ganze Zahl enthält



**Listing 5.26** 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 5.27** 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 *Bsp05\_02Range.doc* finden Sie auf der CD-ROM im Ordner *\Beispiele\Kap05*.

**GoTo.** Die *GoTo*-Methode entspricht in etwa dem Dialogfeld *Bearbeiten/Gehe zu* (in Word 2007 dem Eintrag *Gehe zu* der Schaltfläche *Suchen*, die sich in der Gruppe *Bearbeiten* der Registerkarte *Start* befindet). 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 direktes Gegenstück haben, wie etwa Seiten. Word 2003 wurde zwar um ein *Page*-Objekt ergänzt, es dient jedoch zum Ermitteln der grafischen Darstellung der Seite. Damit kann beispielsweise nicht eine bestimmte Seite angesprochen werden. Sie können nur mit *GoTo* einen Bereich für eine bestimmte Seite bekom-

men: Set rng = doc.Range.GoTo(What:=wdGoToPage, Which:=wdGoToAbsolute, Count:=3). Mehr zu diesem Thema steht in Kapitel 6 im Abschnitt über Textmarken.

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.

**Information.** 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 WordBasic-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.

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.

**InStory.** 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:

```
Dim rng1 as Word.Range, rng2 as Word.Range
Set rng1 = ActiveDocument.Words(3)
Set rng2 = rng1
MsgBox (rng1 Is rng2)
MsgBox (rng1.IsEqual(rng2))
```

Im nachstehenden Fall aber wird die Prüfung mit dem Operator `Is` »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.



Die Beispieldatei *Bsp05\_02Range.doc* finden Sie auf der CD-ROM im Ordner *\Beispiele\Kap05*.

**Duplicate.** Eine Alternative zum zweiten Listing, oben, wäre:

```
Set rng1 = ActiveDocument.Words(3)
Set rng2 = rng1.Duplicate
MsgBox (rng1 Is rng2)
MsgBox (rng1.IsEqual(rng2))
```

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
```

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?

**TextRetrievalMode.** 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 5.28 enthält eine Prozedur, die die Wirkung der verschiedenen Einstellungen demonstriert. Wird sie auf das Dokument in Abbildung 5.11 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 5.11 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 über Feldfunktionen in Kapitel 7.

**Listing 5.28** 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. 5.11** *TextRetrievalMode* beeinflusst, wie Feldcodes und verborgener Text zurückgegeben werden



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.

**FormattedText.** `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 5.29 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

```

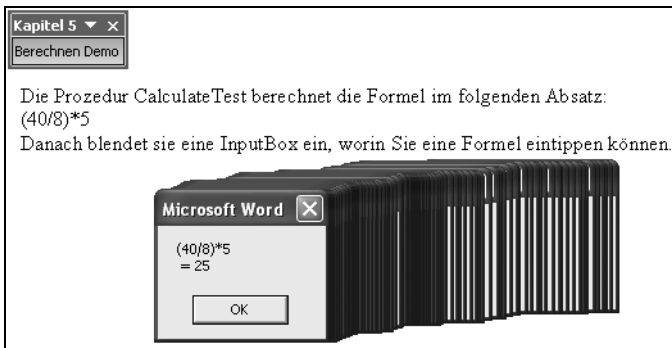


Die Beispieldatei *Bsp05\_03Range.doc* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap05*.

## Eine Formel berechnen

**Calculate.** 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 5.30 zeigt, wie es geht, und in Abbildung 5.12 sehen Sie einen Teil des Ergebnisses.

Abbildg. 5.12 Das passiert, wenn *ScreenUpdating* auf »falsch« gesetzt und ein Meldungsfeld verschoben wird

Listing 5.30 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

```

**Listing 5.30** Mit Formeln im Text rechnen (*Fortsetzung*)

```
'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 5.31** Ein Textfeld 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 + "\\Bsp05_04Range.doc");
        wd.Document doc = Kap05.Kap05Document.WordDokumentOeffnen_CS(wdApp, pfad);
        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
```



**Listing 5.31** Ein Textfeld in einer Windows-Form wird für die Benutzereingabe eingeblendet (Fortsetzung)

```

    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 *Bsp05\_04Range.doc* finden Sie auf der CD-ROM zum Buch im Ordner `\Beispiele\Kap05`.

## 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 deren Automatisierung vorstellen.

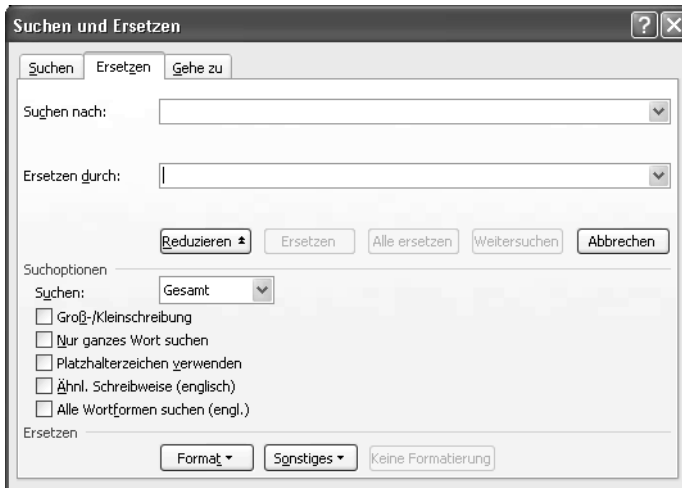
### HINWEIS

*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 noch nicht vertraut sind, lohnt es sich, einen Blick darauf zu werfen, da überraschend viele Aufgaben ohne zusätzliche Programmierarbeit 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 5.32. Mit nur wenigen Ergänzungen kann das Resultat problemlos als Automatisierungscode eingesetzt werden. Viele der Argumente stimmen mit den englischen Bezeichnungen der Dialogfeld-Steuererelemente überein, so dass es relativ einfach ist, die Handlung nachzuvollziehen. Die deutsche Version des Dialogfelds sehen Sie in Abbildung 5.13, die Tabelle 5.11 bietet eine Übersicht der entsprechenden englischen Begriffe.

**Abbildg. 5.13** Das Dialogfeld veranschaulicht einen Großteil der Funktionalität, die auch dem Entwickler zur Verfügung steht



**Tabelle 5.11** 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>
MatchPrefix	<i>*Präfix beachten</i>
MatchSuffix	<i>*Suffix beachten</i>
IgnorePunct	<i>*Interpunktionszeichen ignorieren</i>
IgnoreSpace	<i>*Leerzeichen ignorieren</i>
HitHighlight bzw. ClearHighlight	<i>*Lesehervorhebung</i>
* Neu in Word 2007	

Listing 5.32 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 5.32 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 Dropdownfeld *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 5.32 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

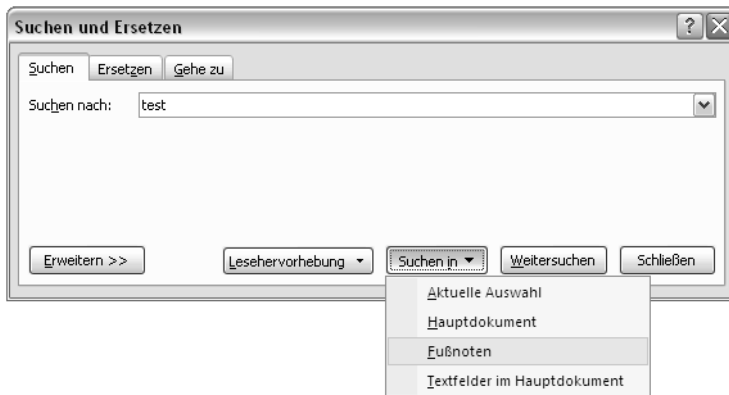
Gehen Sie mit `wdFindContinue` sorgfältig um. Wenn die Suche für eine Zwischenhandlung unterbrochen wird, kann diese Einstellung zu einer Endlosschleife führen, die nur mit `[Strg] + [Pause]` abgebrochen werden kann.



2007

In Word 2007 werden alle zur Verfügung stehenden Dokumentbereiche im Dialogfeld *Suchen und Ersetzen* (Registerkarte *Start*, Gruppe *Bearbeiten*, Schaltfläche *Suchen*) im Dropdownmenü zur Schaltfläche *Suchen in* aufgelistet (Abbildung 5.14).

Abbildg. 5.14 Das Word 2007 Dialogfeld Suchen und Ersetzen



Wird ein bestimmter Dokumentbereich zur Suche angegeben, z.B. *Kopf- und Fußzeilen*, so wird dieser Bereich erst in einem neuen Fensterausschnitt (Panee-Auflistung) angezeigt und aktiviert, bevor die Suche in diesem Bereich ausgeführt wird:

```
ActiveWindow.Panes(2).Activate.
```

Der Index des Fensterbereiches (Panee(Item)) steht in keinem Zusammenhang mit dem Dokumentbereich, sondern wird nach Anzahl der sichtbaren Dokumentbereiche vergeben. Er kann nur mittels Durchlaufen aller Panee und Vergleich ermittelt werden.

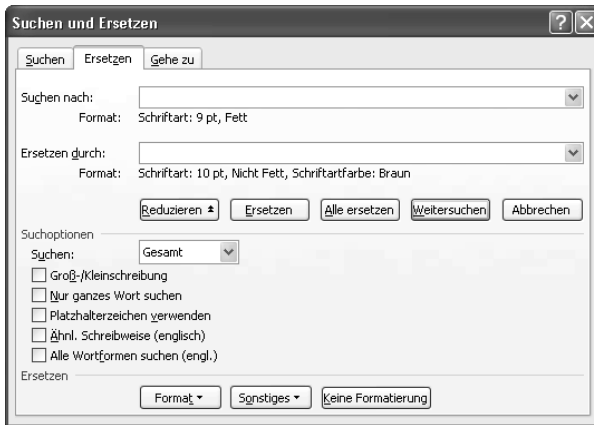
#### HINWEIS

Im Objektmodell von Word 2002/2003 gibt es kein Pendant zum Kontrollkästchen *Gefundene Elemente markieren in*, das sich im Dialogfeld *Bearbeiten/Suchen* befindet. In Word 2007 wurde das Kontrollkästchen durch die Schaltfläche *Leseervorhebung* ersetzt, die den gesuchten Begriff mit der Formatierung »Hervorheben« formatiert bzw. diese Hervorhebung wieder entfernt. Diesen Hervorhebungen entsprechen im Objektmodell die Methoden HitHighlight bzw. ClearHighlight. HitHighlight wird als Ersatz für die Methode Execute verwendet und hat viele der gleichen Parameter.

## Formatierungen suchen und ersetzen

Es gibt einen Bereich, wo der Makrorekorder in Word-Versionen vor 2007 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 5.15 mit dem Code in Listing 5.33. Die fünf Zeilen mit dem Vermerk »hinzugefügt« wurden vom Makrorekorder nicht aufgezeichnet. Sie müssen die Anweisungen .Font.Bold, .Font.Size usw. selbst eingeben.

**Abbildg. 5.15** Diese Kriterien für die Schriftformatierung werden vom Makrorekorder nicht aufgezeichnet



**Listing 5.33** 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
```



2007

Der Makrorekorder in Word 2007 zeichnet deutlich mehr auf, als der in Word 2002 oder Word 2003. Zudem ist der Makrocode strukturierter.

Ein Vergleich von Listing 5.33 mit Listing 5.34 zeigt, dass in Word 2007 die vorher manuell hinzugefügten Kriterien mit aufgezeichnet werden.

**Listing 5.34** Vom Makrorekorder in Word 2007 aufgezeichnete *Suchen und Ersetzen*-Kriterien

```
Sub Word2007Fett9PunktSuchenMitRot10PunktErsetzen()
    Selection.Find.ClearFormatting
    With Selection.Find.Font
```

**Listing 5.34** Vom Makrorekorder in Word 2007 aufgezeichnete *Suchen und Ersetzen*-Kriterien (Fortsetzung)

```

        .Size = 9
        .Bold = True
    End With
    Selection.Find.Replacement.ClearFormatting
    With Selection.Find.Replacement.Font
        .Size = 10
        .Bold = False
        .Color = 192
    End With
    With Selection.Find
        .Text = ""
        .Replacement.Text = ""
        .Forward = True
        .Wrap = wdFindContinue
        .Format = True
        .MatchCase = False
        .MatchWholeWord = False
        .MatchWildcards = False
        .MatchSoundsLike = False
        .MatchAllWordForms = False
    End With
    Selection.Find.Execute Replace:=wdReplaceAll
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 auch 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.

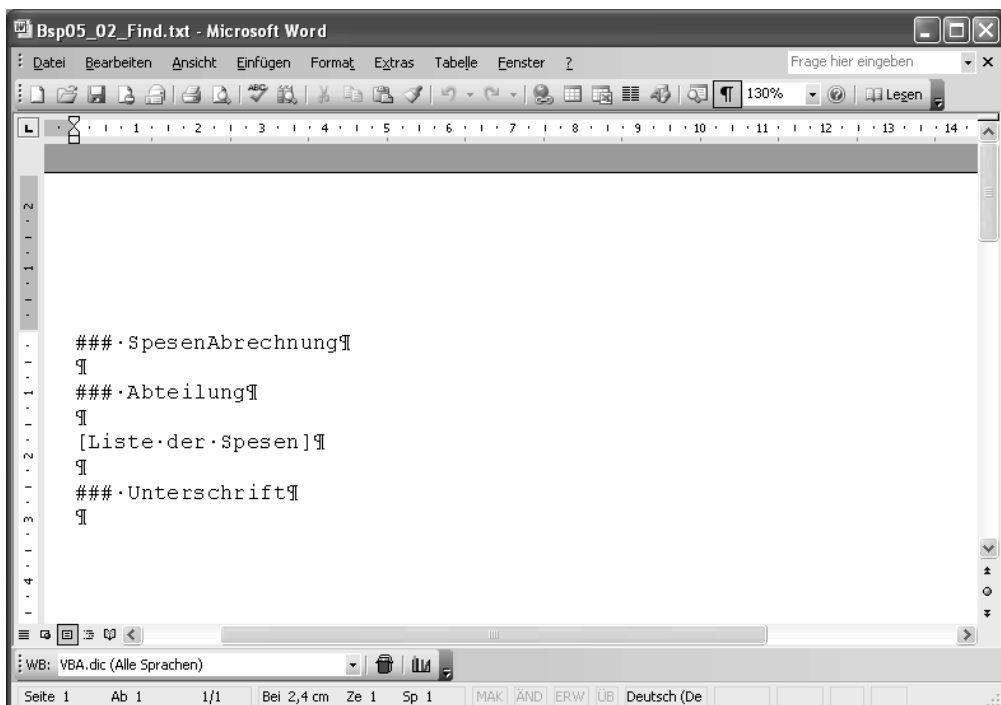
Ähnliches gilt auch für die Suche mit dem Range-Objekt, aber

- die Suche wird im angegebenen Bereich durchgeführt,
- der Bildschirm bleibt ruhig,
- bei erfolgreicher Suche verkleinert sich der Range auf den gefundenen Bereich.

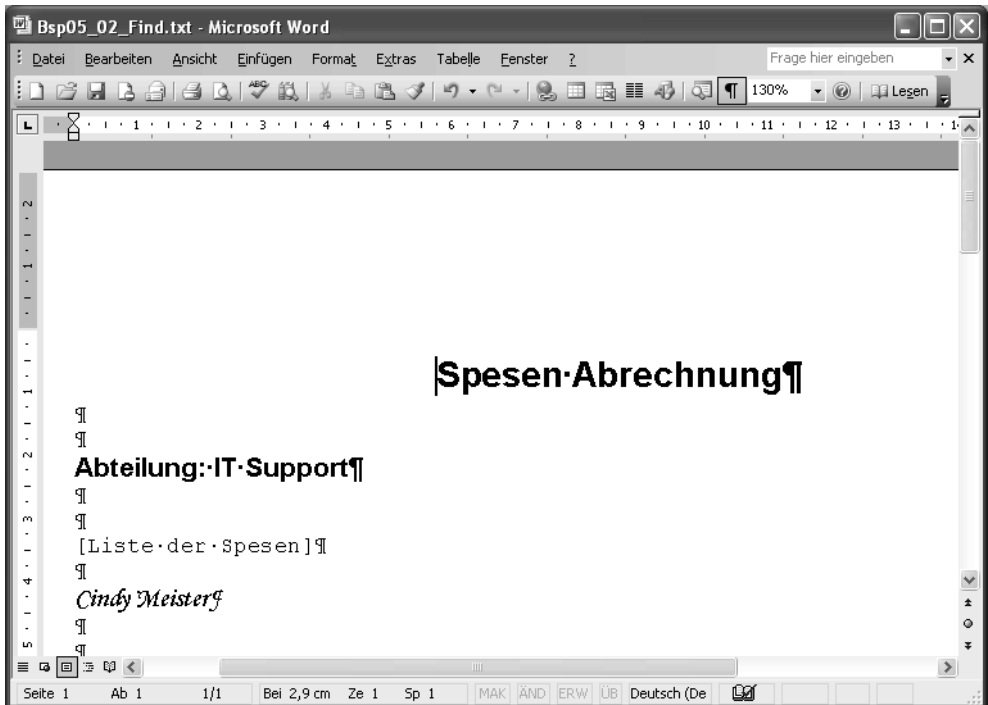
Der Hauptunterschied hier ist, dass die Weitersuche sich auf den *gefundenen Bereich* beschränkt, was meistens nicht erwünscht ist. Folglich muss vor Weiterführung der Suche der Bereich neu gesetzt werden.

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 5.16) und mit der Prozedur eines globalen Add-Ins – *SpesenrechnungVervollstaendigen* in Listing 5.35 bzw. Listing 5.36 – 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 5.17 dargestellt.

Abbildg. 5.16 Eine einfache Spesenabrechnung. Den zu ersetzenden Begriffen stehen die Zeichen ### voran.



Abbildg. 5.17 Das Resultat von Listing 5.35. 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.Duplicate** 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.

**Found.** 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 5.35 auszuführen, kopieren Sie *Bsp05\_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 *Bsp05\_02\_Find.txt* in Word (befindet sich im Ordner *\Beispiele\Kap05*) und führen Sie das Makro *SpesenrechnungVervollstaendigen* aus, das Sie in *Extras/Makros* finden werden. (Die Datei *Bsp05\_01\_Find.dot*, die die AutoText-Einträge enthält, muss sich im gleichen Ordner wie die Datei *Bsp05\_02\_Find.txt* befinden).

Listing 5.35 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 & "\Bsp05_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
        'Bzw. für Word 2007
        .Format = wdFormatXMLDocument
        .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 tmpAutoTextBehaelter As Word.Template
    Dim bFound As Boolean

    Set rngSuchBereich = doc.Content
    Set rngErgebnis = rngSuchBereich.Duplicate
    Set tmpAutoTextBehaelter = doc.AttachedTemplate

    AllgemeineSuchkriterienFestlegen rngErgebnis
    Do
        With rngErgebnis.Find
            .Text = strSuchBegriff
            .Forward = True
```

**Listing 5.35** Nach jeder erfolgreichen Suche findet eine Handlung statt (*Fortsetzung*)

```

        .Wrap = wdFindStop
        .Execute
        bFound = .Found
    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 = tmplAutoTextBehaelter.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 5.36** In C# müssen alle Argumente der Methode *Find.Execute* angegeben werden


```

private void SpesenrechnungVervollstaendigen()
{
    //Sie müssen Bsp05_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 + "\\Bsp05_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
}

```

Listing 5.36 In C# müssen alle Argumente der Methode *Find.Execute* angegeben werden (Fortsetzung)

```

wd.Dialog dlgSaveAs = wdApp.Dialogs[wd.WdWordDialog.wdDialogFileSaveAs];
object[] parameters = new object[1];
parameters[0] = (object) 0;
//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 tmp1AutoTextBehaelter = (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 fortfahren
            try
            {
                object objAutoTextRange = (object) suchErgebnis;
                object objATName = (object) suchErgebnis.Text;
                wd.AutoTextEntry at = tmp1AutoTextBehaelter.AutoTextEntries.get_Item(
                    ref objATName);
                suchErgebnis = at.Insert(suchErgebnis, ref objTrue);
            }
            catch {}
            suchErgebnis.End = suchBereich.End;
            eintraegeZaehler += 1;
        }
    } while (bFound);
    return eintraegeZaehler;
}

```



Die Beispieldateien *Bsp05\_01\_Find.dot* (benutzerspezifische Add-In-Vorlage; enthält die Auto-Text-Einträge), *Bsp05\_02\_Find.txt* sowie *Bsp05\_02\_Find.dot* (globale Add-In-Vorlage, enthält den Add-In-Code) finden Sie auf der CD-ROM im Ordner *\Beispiele\Kap05*.

Falls der Suchvorgang in einer Tabelle unterbrochen wird, gestaltet sich die Handlung etwas komplizierter, weil die Erweiterung eines Bereichs, der sich in einer Tabellenzeile befindet, automatisch ganze Tabellenreihen einschließt. Abbildung 5.18 und Listing 5.37 veranschaulichen das Problem. Zuerst wird der Bereich *rng2* gleich der dritten Zelle der zweiten Tabellenzeile gesetzt. Dann wird der Bereich bis zum Dokumentende erweitert. Der Bereich erstreckt sich jedoch über die ganze zweite Zeile.

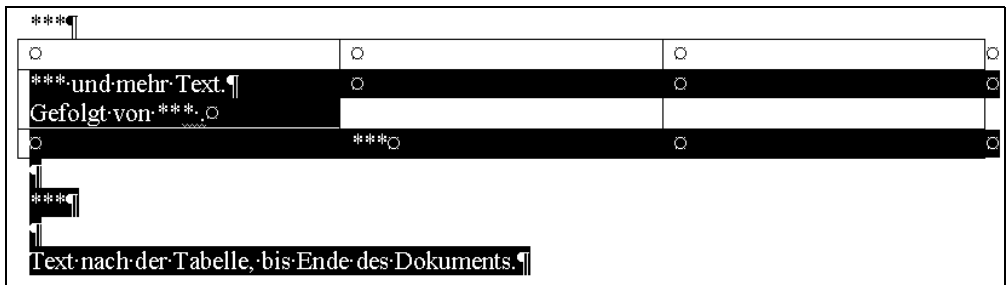
**Listing 5.37** Bereich in einer Tabelle bis zum Dokumentende erweitern

```
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. 5.18** Das Resultat aus Listing 5.37



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 5.38 zeigt.

**Listing 5.38** Ein Dokument mit Tabelle durchsuchen

```
Sub SuchenMitTabelle()
    Dim strSuchBegriff As String
    Dim rngSuchBereich As Word.Range
    Dim rngErgebnisBereich As Word.Range
```

Listing 5.38 Ein Dokument mit Tabelle durchsuchen (Fortsetzung)

```

Dim lngZaehler As Long
Dim tbl As Word.Table

strSuchBegriff = "****"
Set rngSuchBereich = ActiveDocument.Content
Set rngErgebnisBereich = rngSuchBereich.Duplicate

Do While BegriffSuchen(strSuchBegriff, rngErgebnisBereich)
    lngZaehler = lngZaehler + 1
    GefundenerStelleBearbeiten "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
            GefundenerStelleBearbeiten "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)

```

**Listing 5.38** Ein Dokument mit Tabelle durchsuchen (Fortsetzung)

```
rng.InsertAfter strNeuerText
rng.Collapse Direction:=wdCollapseEnd
End Sub
```



Die Beispieldatei *Bsp05\_03\_Find.doc* finden Sie auf der CD-ROM im Ordner *\Beispiele\Kap05*.

## 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 (Dokumententeil). 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 in Kapitel 6 eingehender diskutiert.

In Listing 5.39 bzw. Listing 5.40 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.

Im Beispieldokument sind ein Dokument und eine Grafik verknüpft; die Pfadangabe wird aktualisiert. 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 5.39** 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 & Application.PathSeparator), _
        "\", "\\")
```

Listing 5.39 Alle Dokumentkomponenten durchsuchen (Fortsetzung)

```

For Each sty In doc.StoryRanges
    AlleInstanzenErsetzen sty, strSuchbegriff, strErsatzbegriff
    sty.Fields.Update
    Do While Not (sty.NextStoryRange Is Nothing)
        Set sty = sty.NextStoryRange
        AlleInstanzenErsetzen sty, strSuchbegriff, strErsatzbegriff
        sty.Fields.Update
    Loop
Next sty
End Sub

Sub AlleInstanzenErsetzen(rng As Word.Range, strSuchbegriff As String, _
    strErsatzbegriff As String)
    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
End Sub

```

Listing 5.40 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.FullName + @"\";
    ersatzBegriff = ersatzBegriff.Replace(@"\", @"\");

    foreach (wd.Range sty in doc.StoryRanges)
    {
        AlleInstanzenErsetzen_CS(sty, suchBegriff, ersatzBegriff);
        sty.Fields.Update
        while (sty.NextStoryRange != null)
        {

```

**Listing 5.40** Auch für *Find.Execute* lohnt es sich, »Wrapper«-Funktionen bereitzustellen (Fortsetzung)

```

        wd.Range styFolgender = sty.NextStoryRange;
        AlleInstanzenErsetzen_CS(styFolgender, suchBegriff, ersatzBegriff);
        styl.Fields.Update
        styFolgender = null;
    }
}

private void AlleInstanzenErsetzen_CS(wd.Range rng, string suchBegriff,
    string ersatzBegriff)
{
    rng.TextRetrievalMode.IncludeFieldCodes = true;
    rng.TextRetrievalMode.IncludeHiddenText = true;
    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;
    object objReplaceAll = wd.WdReplace.wdReplaceAll;
    rng.Find.Execute(ref objFindText, 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);
}

```

#### HINWEIS

Leider hat die Methode mit *NextStoryRange* in einigen früheren Word-Versionen 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 eine zusätzliche Prozedur *AlleKopf-UndFußzeilenFuerInhaltTesten* in der Beispieldatei zu diesem Abschnitt dafür, dass Word alle Kopf- und Fußzeilen korrekt erkennt. Führen Sie diese Prozedur vor dem Suchvorgang aus.



Die Beispieldatei *Bsp05\_04\_Find.doc* sowie die verknüpften Dateien *Test.doc* sowie *Seifenblasen.bmp* finden Sie auf der CD-ROM im Ordner *\Beispiele\Kap05*.

## Text in AutoFormen der Kopf- und Fußzeilen bearbeiten

Obwohl die Prozeduren in Listing 5.39 bzw. Listing 5.40 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 5.41 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 5.41 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
    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
            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
    End With
    While rng.Find.Next
        rng.Find.Replacement.Text = strErsatzbegriff
        rng.Find.Execute
    End While
    AlleInstanzenErsetzenMitMustervergleich = lAnzahlVorkommen > 0
End Function

```

**Listing 5.41** Einen Text auch in AutoFormen der Kopf- und Fußzeilen finden und ersetzen

```
.MatchWildcards = True
.MatchSoundsLike = False
.MatchAllWordForms = False
End With
rng.Find.Execute Replace:=wdReplaceAll
AlleInstanzenErsetzenMitMustervergleich = rng.Find.Found
End Function
```

**Listing 5.42** 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()));
}
```

Listing 5.42 Die C#-Version

```
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;
    object objReplaceAll = wd.WdReplace.wdReplaceAll;
    rng.Find.Execute(ref objFindText, ref objFalse, 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 *Bsp05\_05\_Find.doc* finden Sie auf der CD-ROM im Ordner *\Beispiele\Kap05*.

## Bekannte Probleme vermeiden oder beheben

Im Allgemeinen funktioniert *Suchen und Ersetzen* problemlos, wie in den vorangegangenen Abschnitten beschrieben. Es gibt in älteren Word-Versionen jedoch einige Problemfälle, die nur unter bestimmten Umständen auftreten. Die Beschreibung sowie Code-Lösungen befinden sich in einer Beispieldatei auf der Buch-CD.



Die Beispieldatei *Bsp05\_06\_Find.doc* mit Beschreibung und Code finden Sie auf der CD-ROM im Ordner *\Beispiele\Kap05*.

## 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 Laufleiste sind blau statt schwarz gefärbt. Das liegt am *Browserobjekt*, das in Word 97 neu eingeführt wurde.

Wenn Sie auf das runde Symbol 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 *Select*ion-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 5.43 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 5.43** 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
```



Die Beispieldatei *Bsp05\_07\_Find.doc* finden Sie auf der CD-ROM im Ordner *\Beispiele\Kap05*.

## Zusammenfassung

In diesem Kapitel wurde ein Überblick über Grundlagen des Objektmodells von Word vermittelt. Der programmtechnische Umgang mit den Hauptobjekten wurde vermittelt und deren wichtigste Eigenschaften und Methoden näher vorstellt.

- Als Erstes wurde das System-Objekt (Seite 174 ff.) und das Application-Objekt (Seite 177 ff.) vorgestellt.
- Anschließend wurden die übergeordneten Objekte – Selection (Seite 186 ff.), Document (Seite 189 ff.) und Template (Seite 203 ff.) – erläutert.
- Danach wurde das Schlüsselobjekt Range (Seite 207 ff.) behandelt.
- Am Ende des Kapitels wurden die Möglichkeiten der *Find/Raplace*-Eigenschaft des Range- und Selection-Objekts erörtert (Seite 221 ff.).

## Kapitel 6

# Professionelle Dokumente

### In diesem Kapitel:

Abschnitte im Dokument: Das <i>Section</i> -Objekt	242
Bereiche im Dokument: Das <i>StoryRanges</i> -Objekt	244
Die Seite definieren: Das <i>PageSetup</i> -Objekt	248
Die Seite gestalten: Das <i>HeaderFooter</i> -Objekt	256
Lange Dokumente	262
Bausteine: Das <i>BuildingBlocks</i> -Objekt	267
Formatieren mit Stil: Das <i>Style</i> -Objekt	282
Automatische Nummerierung mit Listen	298
Grafiken: Die <i>InlineShape</i> - und <i>Shape</i> -Objekte	309
Zusammenfassung	338

Im vorherigen Kapitel 5 haben wir uns mit den obersten Ebenen des Objektmodells und den am häufigsten verwendeten Objekten befasst. In diesem Kapitel bauen wir auf das bisher vorgestellte Wissen auf und richten unseren Blick auf komplexere, professionellere Dokumente. Dabei denken wir an Szenarien wie die Folgenden:

- Beim Erstellen und der Arbeit mit Dokumenten muss auf die Corporate Identity geachtet werden.
- Es entsteht ein sehr langes Dokument, an dem mehrere Autoren arbeiten werden.
- Ein Dokument enthält wechselnde Seitenausrichtungen, mehrere Textspalten oder Kopf- und Fußzeilen.

In den folgenden Abschnitten werden wir programmtechnische Aspekte behandeln, die mit dem Erstellen und der Arbeit mit solchen Dokumenten verbunden sind.

## 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. (In Word 2007 befindet sich der Befehl *Umbrüche* in der Gruppe *Seite einrichten* der Registerkarte *Seitenlayout*.)

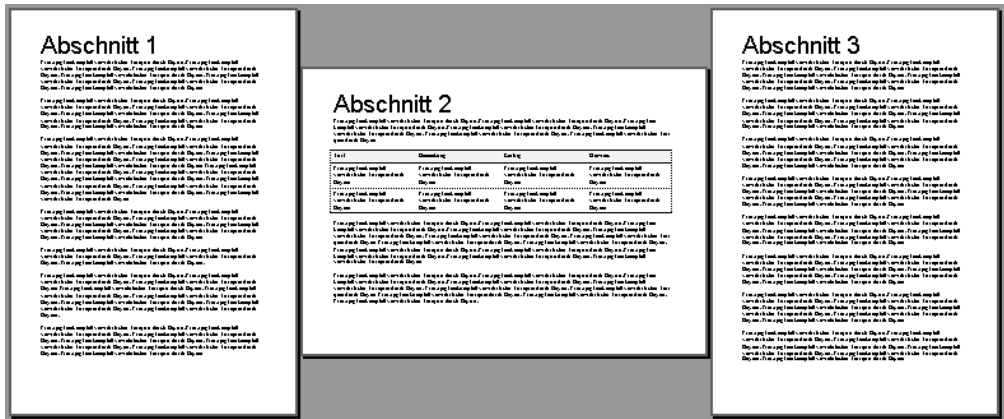
Innerhalb eines Dokuments können vier verschiedene Arten von Abschnittswechsel eingefügt werden. Der eigentliche Abschnittswechsel ist eine Methode eines beliebigen *Range*-Objekts:

```
ActiveDocument.Range.InsertBreak Type:=wdSectionBreakNextPage
```

Eine komplette Liste aller möglichen Umbrüche ist im Abschnitt zum Thema *Range* in Kapitel 5 zusammengefasst.

Wie in Abbildung 6.1 und in Abbildung 6.2 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.

**Abbildung 6.1** 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.

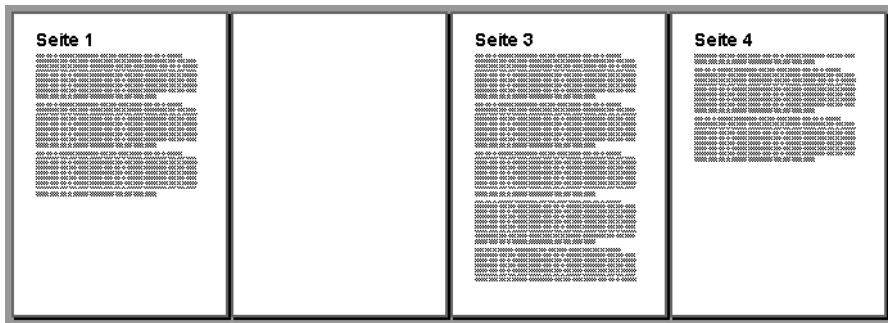
Jeder einzelne Abschnitt wird mittels der Zwischenablage in ein neues Dokument kopiert. Bei diesem Vorgang muss sichergestellt werden, dass der ganze Inhalt eines Abschnitts *ohne* den Abschnittswechsel (Linie mit feinen Doppelpunkten) kopiert wird. Beim letzten Abschnitt muss der ganze Inhalt *ohne* die letzte Absatzmarke kopiert werden. Anschließend kann das Dokument neu aufgebaut werden.



2007

Das Problem des Datenverlusts durch beschädigte Dokumente wurde in Word 2007 erheblich verringert. Im neuen OpenXML-Dateiformat ist der Dokumentinhalt als reiner Text hinterlegt. Die weiterhin vorhandenen, komplexen Strukturen werden durch Verknüpfungen zwischen den verschiedenen XML-Teilen verwaltet. Gerät dennoch etwas in dieser stabileren Dokumentstruktur »durcheinander«, bleibt der Text und meist auch die Formatierung erhalten. Word kann das Dokument trotzdem noch öffnen.

**Abbildg. 6.2** Die Seite 2 wird als leere Seite gedruckt, wenn ein Abschnittswechsel vom Typ `wdSectionBreakOddPage` eingefügt wird



**HeadersFooters.** Grundsätzlich verfügt jeder Abschnitt über eigene Kopf- und Fußzeilen. Dies gilt auch für einen Abschnittswechsel vom Typ `wdSectionBreakContinuous` (vgl. Abbildung 6.4). 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 6.11).

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.

**PageSetup.** Die *PageSetup*-Eigenschaft repräsentiert das Seitenlayout des Abschnitts. Die Eigenschaft gibt ein *PageSetup*-Objekt zurück. Die Eigenschaften und Methoden dieser Auflistung sind im Abschnitt »Die Seite definieren: Das *PageSetup*-Objekt« in diesem Kapitel erläutert.

**ProtectedForForms.** Jedes Dokument kann für die Texteingabe geschützt werden. Dieser Dokumentschutz umfasst 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 in Kapitel 7.

## 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 6.1 Rechnung. Hier werden alle vorhandenen StoryRanges des Dokuments durchforstet und die gefundenen Shape-Objekte aufaddiert.

Listing 6.1

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 6.1 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 6.1 jedoch nicht durchgeführt.

Um festzustellen, ob die aktuelle Dokumentkomponente über weitere Elemente verfügt, steht die *NextStoryRange*-Eigenschaft zur Verfügung. In Tabelle 6.2 wurden die möglichen Werte dieser Eigenschaft zusammengestellt.

Mehrere Programmbeispiele, die alle Dokumentkomponenten und deren zugehörige Elemente berücksichtigen, sind in Kapitel 5 beschrieben. 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 6.1 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 6.1** Die einzelnen Elemente der *StoryRanges*-Auflistung

WdStoryType-Enum	Wert	Bedeutung
<code>wdMainTextStory</code>	1	Haupttext des Dokuments
<code>wdFootnotesStory</code>	2	Fußnoten
<code>wdEndnotesStory</code>	3	Endnoten
<code>wdCommentsStory</code>	4	Kommentare
<code>wdTextFrameStory</code>	5	Textrahmen
<code>wdEvenPagesHeaderStory</code>	6	Kopfzeilen auf den Seiten mit gerader Seitenzahl, sofern <i>Gerade/Ungerade anders</i> aktiviert wurde <sup>a</sup>
<code>wdPrimaryHeaderStory</code>	7	Standardkopfzeile
<code>wdEvenPagesFooterStory</code>	8	Fußzeilen auf den Seiten mit gerader Seitenzahl, sofern <i>Gerade/Ungerade anders</i> aktiviert wurde <sup>a</sup>
<code>wdPrimaryFooterStory</code>	9	Standardfußzeile
<code>wdFirstPageHeaderStory</code>	10	Kopfzeile auf der ersten Seiten des Dokuments, sofern <i>Erste Seite anders</i> aktiviert wurde <sup>a</sup>
<code>wdFirstPageFooterStory</code>	11	Fußzeile auf der ersten Seiten des Dokuments, sofern <i>Erste Seite anders</i> aktiviert wurde <sup>a</sup>
<code>wdFootnoteSeparatorStory</code>	12	Fußnotentrennlinie
<code>wdFootnoteContinuationSeparatorStory</code>	13	Fußnoten-Fortsetzungstrennlinie
<code>wdFootnoteContinuationNoticeStory</code>	14	Fußnoten-Fortsetzungshinweis
<code>wdEndnoteSeparatorStory</code>	15	Endnotentrennlinie
<code>wdEndnoteContinuationSeparatorStory</code>	16	Endnoten-Fortsetzungstrennlinie
<code>wdEndnoteContinuationNoticeStory</code>	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. In Word 2007 klicken Sie auf den Dialoglauncher der Gruppe *Seite einrichten* in der Registerkarte *Seitenlayout*.

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 in Kapitel 5 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 6.1 dargestellt, begrenzt.

**NextStoryRange.** In der Tabelle 6.2 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 6.2 bzw. das Listing 6.3, 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.

**Tabelle 6.2** 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 6.2** 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 6.3** Die C#-Version: *Sections*- sowie *Headers*-Auflistungen werden wie Datenfelder behandelt


```
private void Demo_NextStoryRanges_CS(wd.Document doc)
{
    //Abschnitt Zwei und Drei erzeugen
    object objSectionNextPage = wd.WdBreakType.wdSectionBreakNextPage;
    doc.Content.InsertBreak(ref objSectionNextPage);
    doc.Content.InsertBreak(ref objSectionNextPage);

    //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:

- Festlegen des Papierformats und dessen Ausrichtung .
- Bestimmen der Seitenränder und des Bundstegs.

- Definieren der verschiedenen Kopf- und Fußzeilen sowie deren Abstand zur beschreibbaren Textfläche des Dokuments.
- Zuweisen der Papierzufuhr für den Ausdruck.

Alle diese Einstellungen sollten bereits während des Anlegens der Dokumentvorlage als Standard-einstellungen definiert werden. Dazu steht das Dialogfeld *Seite einrichten* zur Verfügung. Dieses kann durch Aufruf des entsprechenden Menübefehls *Datei/Seite einrichten* (in Word 2007 klicken Sie auf den Dialoglauncher der Gruppe *Seite einrichten* auf der Registerkarte *Seitenlayout*) eingeblendet werden. Somit ist sichergestellt, dass das Grundlayout der einzelnen Dokumente immer gleich ist. Mehr zum Aufbau einer Dokumentvorlage ist in Kapitel 13 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 oder in sehr langen Dokumenten vorkommen und deshalb nicht auf Anhieb reproduziert werden können.

**PaperSize, PageHeight, PageWidth.** Die PaperSize-Eigenschaft dient zum Festlegen des Papierformats. In Tabelle 6.3 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 6.3** Zusammenstellung der wichtigsten Konstanten für das Papierformat

Konstante	Wert	Bedeutung
<code>wdPaperCustom</code>	41	Benutzerdefiniertes Papierformat
<code>wdPaperA5</code>	9	DIN A5
<code>wdPaperA4</code>	7	DIN A4
<code>wdPaperA3</code>	6	DIN A3
<code>wdPaperLetter</code>	2	Letter, amerikanisches Papierformat

**Orientation.** Die `Orientation`-Eigenschaft dient zum Festlegen der Dokumentausrichtung innerhalb des Abschnitts. Das Dokument kann entweder im Hochformat (`wdOrientPortrait`) oder im Querformat (`wdOrientLandscape`) genutzt werden.

**MarginTop, MarginBottom, MarginLeft, MarginRight.** Anhand der Eigenschaften `MarginTop` (Rand oben), `MarginBottom` (unten), `MarginLeft` (links) und `MarginRight` (rechts) wird der effektiv beschreibbare Bereich des Abschnitts festgelegt.

In Listing 6.4 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 6.3 ersichtlich.

**Listing 6.4** 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

    'Vertikalen bzw. horizontalen Einzug berechnen
        sngEinzugV = (PointsToCentimeters(.PageHeight) - sngSEITE) / 2
        sngEinzugH = (PointsToCentimeters(.PageWidth) - sngSEITE) / 2

    'Seitenränder zuweisen
        .TopMargin = CentimetersToPoints(sngEinzugV)
        .BottomMargin = CentimetersToPoints(sngEinzugV)
    End With
End Sub
```

Listing 6.4 Satzspiegel für ein Kärtchen berechnen und zuweisen (Fortsetzung)

```

        .LeftMargin = CentimetersToPoints(sngEinzugH)
        .RightMargin = CentimetersToPoints(sngEinzugH)
        .Gutter = CentimetersToPoints(0)
    End With

    'Textbegrenzungen einblenden
    doc.ActiveWindow.View.ShowTextBoundaries = True
End Sub

```

Listing 6.5 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 6.3).

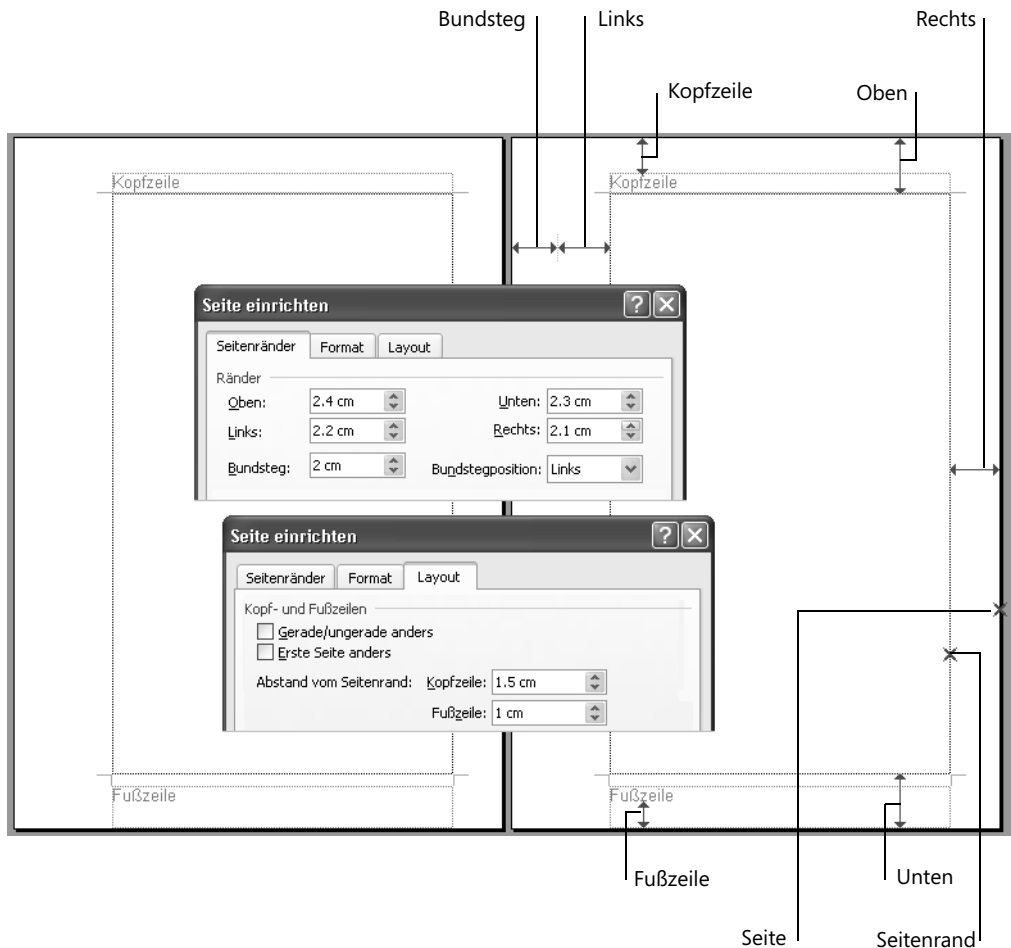
**HeaderDistance, FooterDistance.** Die HeaderDistance- und FooterDistance-Eigenschaften definieren den Abstand von der Seite bis zur Oberkante der Kopfzeile bzw. von der Seite bis Unterkante der Fußzeile.

#### 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.

Abbildg. 6.3 Die Einstellungen des Satzspiegels und deren Auswirkungen



#### HINWEIS

In Abbildung 6.3 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 6.32).

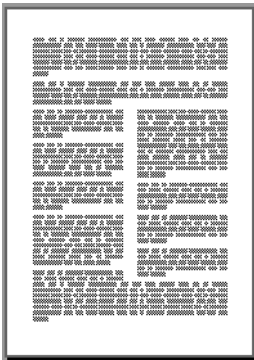
**TextColumns.** Die TextColumns-Eigenschaft gibt eine TextColumns-Auflistung zurück. Diese Auflistung repräsentiert die Textspalten (»Zeitungsspalten«) des Abschnitts. Wie in Abbildung 6.4 ersichtlich, kann innerhalb der gleichen Seite eines Dokuments eine unterschiedliche Spaltenzahl realisiert werden.

#### HINWEIS

Um eine unterschiedliche Anzahl von Spalten auf der gleichen Seite des Dokuments realisieren zu können, muss mindestens ein fortlaufender Abschnittswechsel (wdSection-BreakContinuous) eingefügt werden.



**Abbildg. 6.4** 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. (In Word 2007 finden Sie den Befehl für das Dialogfeld im Dropdownmenü der Schaltfläche *Spalten*, die sich in der Gruppe *Seite einrichten* der Registerkarte *Seitenlayout* befindet.)

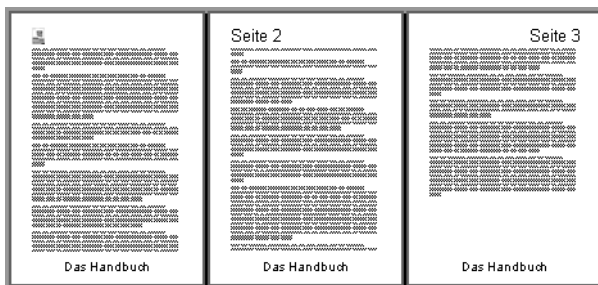
Alle Eigenschaften, die bis jetzt vorgestellt wurden, hatten einen direkten Einfluss auf das Seitenlayout des entsprechenden Abschnitts. Dies ist bei den nachfolgenden Eigenschaften nicht mehr der Fall.

**DifferentFirstPageHeaderFooter, OddAndEvenPagesHeaderFooter.** 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. 6.5** Drei unterschiedliche Kopfzeilen können pro Abschnitt erzeugt werden



**HINWEIS** Damit das Dokument wie in Abbildung 6.5 auf allen Seiten über eine einheitliche Fußzeile verfügt, müssen diese manuell oder mittels eines Makros wie in Listing 6.6 synchronisiert werden.

**Listing 6.6** 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
        .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.

**FirstPageTray, OtherPagesTray.** 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.

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 6.7. Nur die erste Seite des Dokuments soll das spezielle Briefkopfpapier mit aufgedrucktem Logo verwenden. Die restlichen Seiten werden auf normalem weißem Papier ausgegeben.

**Listing 6.7** 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
```

Listing 6.7 Sicherstellen, dass nur die erste Seite des Dokuments auf Briefkopfpapier gedruckt wird (Fortsetzung)

```

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 6.4 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 6.4 Zusammenstellung der Konstanten für die Papierschlachtsteuerung

WdPaperTray-Konstante	Wert	Bedeutung
wdPrinterDefaultBin	0	Standardpapierschlacht gemäß den Einstellungen des Druckertreibers bzw. der Papierschlacht, der in den Optionen festgelegt wurde
wdPrinterOnlyBin	1	Einziger Papierschlacht (entspricht Oberer Papierschlacht)
wdPrinterUpperBin	1	Oberer Papierschlacht
wdPrinterLowerBin	2	Unterer Papierschlacht
wdPrinterMiddleBin	3	Mittlerer Papierschlacht
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	Papierschlacht mit kleinem Papierformat
wdPrinterLargeFormatBin	10	Papierschlacht mit großem Papierformat

**Tabelle 6.4** Zusammenstellung der Konstanten für die Papierschachtsteuerung (Fortsetzung)

WdPaperTray-Konstante	Wert	Bedeutung
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 6.8 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 6.5 kurz zusammengefasst.

Tabelle 6.5 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 Dokumentvariable

Um eine Kopf- bzw. Fußzeile manuell in das bestehende Dokument aufzunehmen, muss der Menübefehl *Ansicht/Kopf- und Fußzeile* aufgerufen werden. Diese müssen in der Ansicht *Seitenlayout* bearbeitet werden. Gleichzeitig wird die Symbolleiste *Kopf- und Fußzeile* eingeblendet. Diese Symbolleiste stellt die wichtigsten Funktionen zum Navigieren innerhalb der Kopf- und Fußzeilen zur Verfügung.



2007

In Word 2007 werden Kopf- und Fußzeilen über Schaltflächen auf der Registerkarte *Einfügen* erstellt. Es werden mehrere vordefinierte Bausteine vorgeschlagen; der Menübefehl, um eine Kopf- oder Fußzeile direkt zu bearbeiten, steht am Ende der Liste. Befindet sich die Einfügemarke in einer Kopf- oder Fußzeile, steht die Registerkarte *Kopf- und Fußzeilentools* zur Verfügung.

**HINWEIS**

Ein Beispiel für das Erstellen und Bearbeiten von Kopf- und Fußzeilen finden Sie in Kapitel IV des Bonusteils auf der Buch-CD.

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 6.6** Zusammenstellung der Konstanten für den Zugriff auf die Kopf- bzw. Fußzeilen

WdHeaderFooter-Enum	Wert	Bedeutung
wdHeaderFooterEvenPages	3	Bei aktivierter <b>OddAndEvenPagesHeaderFooter</b> -Eigenschaft wird die Kopf- bzw. Fußzeile auf allen Seiten mit gerader Seitennummer ausgegeben
wdHeaderFooterFirstPage	2	Bei aktivierter <b>DifferentFirstPageHeaderFooter</b> -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 6.8** 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 Makrorekorder 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 sichergesellt 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 6.9 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 6.9** 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
```

**LinkToPrevious.** 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 6.10 dargestellt, kann beispielsweise die gleiche Fußzeile in allen Abschnitten verwendet werden. Die Kopfzeile wird in jedem Abschnitt anders definiert.

**Listing 6.10** 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.

**PageNumbers.** 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 oder eines Textfelds. Diesen Positionsrahmen und Textfelder mittels VBA zu bearbeiten, ist weit aufwändiger als das Einfügen und Bearbeiten eines Page-Felds.



2007

In Word 2007 wurde das Vorgehen für einfache Seitenzahlen, die im Bereich der Kopf- oder Fußzeile positioniert sind, geändert. Solche Seitenzahlen werden mit Tabstopps in den Textfluss eingefügt.

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 6.11 dargestellt wird.
- Zum Formatieren des *Page*-Felds (Zahlenformat, Kapitelnummer usw.) wie dies in Listing 6.13 dargestellt wird.

In der Benutzerschnittstelle wird das in Abbildung 6.6 abgebildete Dialogfeld durch Anwählen des Menübefehls *Einfügen/Seitenzahlen* und Betätigen der Schaltfläche *Format* erreicht. In Word 2007 befindet sich der Befehl *Seitenzahlen formatieren* am Ende des Dropdownmenüs zur Schaltfläche *Seitenzahl* (Registerkarte *Einfügen*, Gruppe *Kopf- und Fußzeilen*). 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 6.11** 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
```



Listing 6.11 Festlegen des Startwerts für die Seitennummer im zweiten Abschnitt (Fortsetzung)

```

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 6.12 Die C#-Version



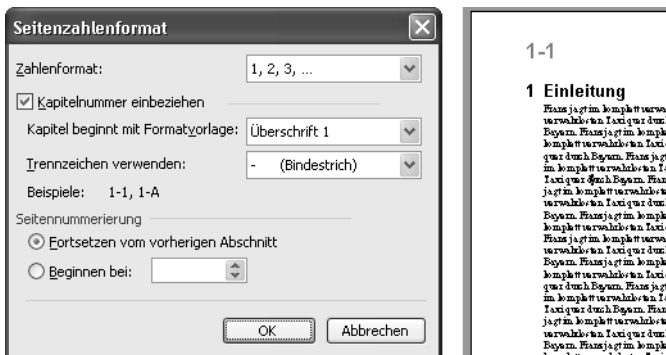
```

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 6.6 sind die verschiedenen Möglichkeiten zum Formatieren der Seitennummer dargestellt. Das Resultat der dargestellten Einstellungen ist ebenfalls ersichtlich. Die gleichen Einstellungen werden anschließend im Listing 6.13 umgesetzt.

Abbildg. 6.6 Legen Sie die Eigenschaften für das Seitenzahlenformat fest



**Listing 6.13** 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 in Kapitel 5 in der Diskussion zum Thema Selection-Objekt erläutert wurde, raten wir Ihnen jedoch vom Arbeiten 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 Handbüchern 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 die Einführung des OpenXML-Dateiformats in Word 2007 als Standardformat verliert die über das Objektmodell automatisierte Erstellung und Verwaltung von langen Dokumenten an Bedeutung. Die Aufgabe ist allgemein schneller mit XML zu erreichen und die Gefahr der Dokumentbeschädigung ist geringer.

**HINWEIS**

Eine Einführung in die XML-Funktionalität in Word finden Sie in Kapitel 22 dieses Buchs. Mehr Informationen zur programmtechnischen Erstellung von Word 2007-Dokumenten finden Sie im Internet unter <http://OpenXMLDeveloper.org>.

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 ein tausend 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 Abschnittswechseln 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).

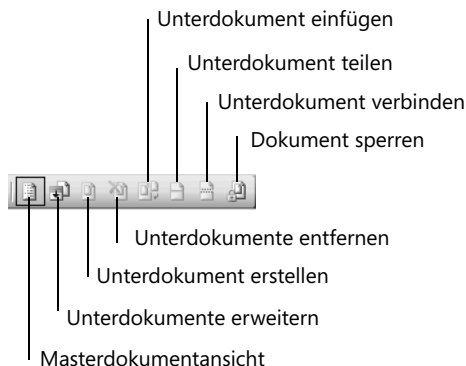
Zentraldokumente werden in der Gliederungsansicht mit den Werkzeugen der »Masterdokumentansicht« der Symbolleiste *Gliederung* (Abbildung 6.7) verwaltet. Filialdokumente können aus dem Text des Zentraldokuments oder durch Einfügen eines gespeicherten Dokuments erstellt werden. Allgemeine Angaben liefert die Word-Hilfedatei. In diesem Abschnitt werden wir lediglich einige wichtige Punkte kurz vorstellen.



In Word 2007 werden Zentraldokumente immer noch in der Gliederungsansicht bearbeitet. Auf der Registerkarte *Gliederung* verfügt die Funktionalität über eine eigene Gruppe *Zentraldokument*. Per Klick auf die Schaltfläche *Dokument anzeigen* werden die weiteren Werkzeuge eingeblendet, die denen der älteren Symbolleiste in Abbildung 6.7 entsprechen. Die Beschriftung *Unterdokument entfernen* lautet neu *Verknüpfung aufheben*, was dem Sinn der Handlung besser entspricht. Die grundlegende Funktionalität wurde nicht geändert.

Ein Filialdokument kann in zwei Dokumente 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).

Abbildg. 6.7 Die Symbolleiste für die Arbeit mit Zentral- und Filialdokumenten



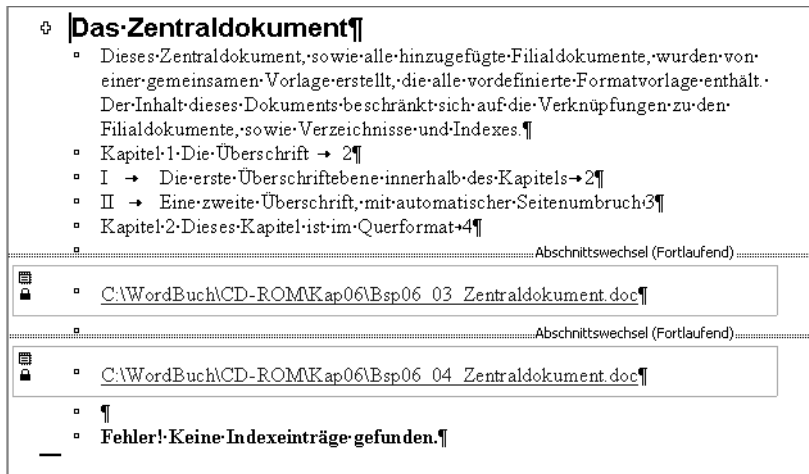
#### HINWEIS

Den Übersetzern der Benutzerschnittstelle der Word-Versionen 2002 und 2003 war es offensichtlich nicht klar, dass die Funktionalität seit Jahren Zentral- und Filialdokumente heißt. Sie haben etwas willkürlich, an die englische Terminologie anlehnend, die Symbolleisten-schaltflächen mit »Masterdokument« und »Unterdokument« beschriftet. Da in der Word-Hilfe immer noch »Zentraldokument« und »Filialdokument« gebraucht werden, behalten wir diese Ausdrücke bei. In Word 2007 wurden die Beschriftungen wieder korrigiert.

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 6.8 ersichtlich. Ab 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. 6.8 Noch nicht erweiterte Filialdokumente in der Gliederungsansicht



Die Beispieldatei *Bsp06\_01\_Zentraldokument.dot* ist eine Vorlage für das Beispiel in Abbildung 6.8. 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* (bzw. in Word 2007 über den gleichnamigen Befehl unter der *Office*-Schaltfläche) unter einem eigenen Dateinamen gespeichert. Wird dies nicht getan, speichert Word beim Schließen des Zentraldokuments das Filialdokument automatisch in den gleichen Ordner. 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.

**SubDocument.** Ist ein Dokument ein Zentraldokument, werden seine Filialdokumente über die *SubDocuments*-Eigenschaft angesprochen. Wichtige Eigenschaften und Methoden sind *AddFromFile* (Filialdokument aus 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 Überschriftenebene 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 6.14 zeigt, wie aus einem gewöhnlichen Dokument durch Einfügen von Dateien als Filialdokumente ein Zentralkument wird. Bitte beachten Sie, dass

- das Dokument in der Zentralkumentansicht sein muss,
- diese Funktionalität auf der gegenwärtigen Markierung im Dokument basiert: das Filialdokument wird an dieser Stelle eingefügt und ersetzt eventuell markierten Text.

Um der ersten Forderung gerecht zu sein, blendet der Code die korrekte Ansicht ein. Es wird dann mit dem Selection-Objekt nach der gewünschten Stelle gesucht, wo die Filialdokumente einzufügen sind. Diese befinden sich alle im gleichen Ordner und deren Dateinamen fangen mit den Zeichen »Test« an. Es wird durch die Dateien in diesem Ordner geschleift, bis keine Dateien mehr vorliegen, die dem festgelegten Muster entsprechen.

In manchen Word-Versionen, falls das Filialdokument auf einer anderen Vorlage als das Zentralkument basiert, blendet Word eine entsprechende Meldung ein, die nicht unterdrückt werden kann.

Beim Speichern des Zentralkuments prüft Word 2007, ob alle Dokumente das gleiche Dateiformat haben und wird Sie bei Unstimmigkeiten fragen, ob das Filialdokument in einem anderen Format gespeichert werden soll.

**Listing 6.14** Alle doc-Dateien eines Ordners, die mit den 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\_Zentralkument.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* (in Word 2007 finden Sie den Befehl *Text aus Datei* unter der Schaltfläche *Objekt* in der Gruppe *Text* der Registerkarte *Einfügen*) 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 Abschnittswechsel 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 den Abschnitt »Feldfunktion« in Kapitel 7).

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.

## Bausteine: Das *BuildingBlocks*-Objekt



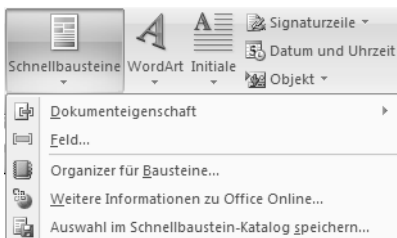
Eine weitere Neuerung neben der neuen Oberfläche in Word 2007 (siehe Kapitel 17) sind die *Schnellbausteine* zum Hinzufügen vorformatierter Inhalte zu Dokumenten. Sie rufen diesen Menübefehl über die Schaltfläche *Schnellbausteine* auf der Registerkarte *Einfügen* in der Gruppe *Text* auf (Abbildung 6.9).

Das Dropdownmenü zur Schaltfläche *Schnellbausteine* umfasst standardmäßig die folgenden Befehle, die z.T. in den nachfolgenden Abschnitten angesprochen werden:

- *Dokumenteigenschaft* (Abschnitt »Die Dokumenteigenschaften«)
- *Feld* (Abschnitt »Felder (Feldfunktionen)«)
- *Organizer für Bausteine* (Abschnitt »Der Organizer für Bausteine«)
- *Auswahl im Schnellbaustein-Katalog speichern* (Abschnitt »Der Schnellbaustein-Katalog«)

Erst über den letzten Menüpunkt können Sie diesem Menü direkt Schnellbausteine (als Menüeinträge) hinzufügen, was im entsprechenden Abschnitt besprochen wird.

Abbildg. 6.9 Schaltfläche in der Gruppe *Text* für den Zugriff auf die Schnellbausteine



In diesem Menü finden Sie somit neben den aus älteren Word-Versionen bekannten Bausteinen (Dokumenteigenschaften und Felder) auch neue, die sich hinter den Menüpunkten *Organizer für Bausteine* und *Auswahl im Schnellbaustein-Katalog speichern* verbergen.

In diesem Kapitel soll nun ein Überblick über die Funktionen dieses Menüpunktes gegeben werden und wie Sie aus VBA darauf zugreifen können.

**HINWEIS** Über den folgenden Link werden Sie zur Microsoft-Webseite weitergeleitet, auf der Sie weitere Bausteine und Informationen zu den Bausteinen finden:

<http://office.microsoft.com/de-de/templates/CT012316111031.aspx>

Mehr zu den Word 2007-Schnellbausteinen erfahren Sie bei Interesse im Microsoft Press-Buch »Office 2007 – Das Profibuch«.

## Schnellbausteine

Schnellbausteine (kurz *Bausteine* = *Building Blocks*) sind wiederverwendbare Inhalte, die Texte, Bilder und Formatierungen enthalten können. Sie sind in Katalogen und Kategorien gespeichert und somit jederzeit wiederverwendbar. Bausteine werden in Dokumentvorlagen gespeichert und können so verteilt werden.

Es wird dabei zwischen den benutzerdefinierten und integrierten Bausteinen unterschieden. Während benutzerdefinierte Bausteine in beliebigen Vorlagen gespeichert werden können, sind die integrierten Bausteine in der Vorlage *Building Blocks.dotx* gespeichert.

**HINWEIS** Die Vorlage *Building Blocks.dotx* liegt standardmäßig in Windows XP im Verzeichnis

*C:\Dokumente und Einstellungen\<Benutzername>\Anwendungsdaten\Microsoft\Document Building Blocks\1031*

und in Windows Vista im Verzeichnis

*C:\Users\<Benutzername>\AppData\Roaming\Microsoft\Document Building Blocks\1031\Building Blocks.dotx*

Während die Bausteine in Word 97 bis Word 2003 nur die AutoTexte und Felder umfassten, beinhalten sie jetzt neben den beiden Typen insgesamt folgende Elemente (Kataloge):

**Tabelle 6.7** Übersicht über die *BuildingBlockTypes* mit Index-Nummer und *wdBuildingBlockTypes*-Konstante

ID	BuildingBlockType	wdBuildingBlockTypes-Konstante
1	Schnellbausteine	wdTypeQuickParts
2	Deckblätter	wdTypeCoverPage
3	Formeln	wdTypeEquations
4	Fußzeilen	wdTypeFooters
5	Kopfzeilen	wdTypeHeaders
6	Seitenzahlen	wdTypePageNumber
7	Tabellen	wdTypeTables
8	Wasserzeichen	wdTypeWatermarks



**Tabelle 6.7** Übersicht über die *BuildingBlockTypes* mit Index-Nummer und *wdBuildingBlockTypes*-Konstante (Fortsetzung)

ID	BuildingBlockType	wdBuildingBlockTypes-Konstante
9	AutoText	wdTypeAutoText
10	Textfelder	wdTypeTextBox
11	Seitenzahlen (oben)	wdTypePageNumberTop
12	Seitenzahlen (unten)	wdTypePageNumberBottom
13	Seitenzahlen (Ränder)	wdTypePageNumberPage
14	Inhaltsverzeichnis	wdTypeTableOfContents
15	Benutzerdefinierte Schnellbausteine	wdTypeCustomQuickParts
16	Benutzerdefinierte Deckblätter	wdTypeCustomCoverPage
17	Benutzerdefinierte Formeln	wdTypeCustomEquations
18	Benutzerdefinierte Fußzeilen	wdTypeCustomFooters
19	Benutzerdefinierte Kopfzeilen	wdTypeCustomHeaders
20	Benutzerdefinierte Seitenzahlen	wdTypeCustomPageNumber
21	Benutzerdefinierte Tabellen	wdTypeCustomTables
22	Benutzerdefinierte Wasserzeichen	wdTypeCustomWatermarks
23	Benutzerdefinierter AutoText	wdTypeCustomAutoText
24	Benutzerdefinierte Textfelder	wdTypeCustomTextBox
25	Benutzerdefinierte Seitenzahlen (oben)	wdTypeCustomPageNumberTop
26	Benutzerdefinierte Seitenzahlen (unten)	wdTypeCustomPageNumberBottom
27	Benutzerdefinierte Seitenzahlen (Ränder)	wdTypeCustomPageNumberPage
28	Benutzerdefiniertes Inhaltsverzeichnis	wdTypeCustomTableOfContents
29	Benutzerdefiniert 1	wdTypeCustom1
30	Benutzerdefiniert 2	wdTypeCustom2
31	Benutzerdefiniert 3	wdTypeCustom3
32	Benutzerdefiniert 4	wdTypeCustom4
33	Benutzerdefiniert 5	wdTypeCustom5
34	Literaturverzeichnisse	wdTypeBibliography
35	Benutzerdefinierte Literaturverzeichnisse	wdTypeCustomBibliography

Sie finden in den nicht benutzerdefinierten Katalogen eine ganze Reihe von verschiedenen vorgefertigten Bausteinen für die jeweiligen Katalogtypen. So z.B. verschiedene Bausteine für Seitenzahlen oder Wasserzeichen.

Wie diese Kataloge im gesamten Kontext der Bausteine eingeordnet sind, lesen Sie im Abschnitt »Der Organizer für Bausteine« in diesem Kapitel.

## Die zentrale Bausteinvorlage *Building Blocks.dotx*

Normalerweise werden in Word 2007 Bausteine geladen, wenn sie zum ersten Mal benötigt werden, z. B. wenn ein Benutzer mithilfe der Multifunktionsleiste einen Katalog aufruft. Da die integrierten Bausteine jedoch in der *Building Blocks.dotx* gespeichert sind, können Sie Word mit der folgenden Anweisung zwingen, diese Bausteine sofort zu laden:

```
Templates.LoadBuildingBlocks
```

Allerdings liefert diese Methode keinen Verweis auf die Vorlage zurück, so dass Sie für den direkten Zugriff auf diese Vorlage erst noch den Index der *Building Blocks.dotx* ermitteln müssen. Wenn keine Add-Ins geladen sind, ist die *Building Blocks.dotx* die erste Vorlage und Sie erhalten über die folgende Definition Zugriff auf diese Vorlage:

```
Templates(1)
```

Wenn hingegen weitere Add-Ins geladen sind, kann diese Zuordnung nicht garantiert werden, so dass erst der korrekte Index in einer Schleife ermittelt werden muss (siehe den Abschnitt zum »Template-Objekt« in Kapitel 5).

### TIPP

Wenn Sie in einem Makro je nach Word-Version automatisch zwischen dem *AutoText*-Dialogfeld (Word 97 bis Word 2003) und dem Dialogfeld *Organizer für Bausteine* (Word 2007) umschalten möchten, können Sie das nachstehende Makro *func\_AutoText\_OR\_BuildingBlocks\_Dialog* verwenden.

Mit diesem Makro wird anhand der Versionsnummer von Word das jeweils korrekte Dialogfeld aufgerufen.

Um einen Kompilierungsfehler zu vermeiden, wird dabei die `wdDialogs`-Konstante `wdDialogBuildingBlockOrganizer` als Konstante deklariert.

**Listing 6.15** Makro zum automatischen Aufruf des verfügbaren AutoText/Bausteine-Dialogfeldes

```
Const wdDialogBuildingBlockOrganizer As Long = 2067
Function func_AutoText_OR_BuildingBlocks_Dialog() As String
If Val(Application.Version) = 12 Then
'Word 2007 - Aufruf des Bausteine-Organizers
Dialogs(wdDialogBuildingBlockOrganizer).Show
func_AutoText_OR_BuildingBlocks_Dialog = "BuildingBlocks"
Else
'Word 2000-2003 - Aufruf des Autotext-Dialogfeldes.
Dialogs(wdDialogEditAutoText).Show
func_AutoText_OR_BuildingBlocks_Dialog = "AutoText"
End If
End Function
```

**HINWEIS** Sollte einmal der Fall auftreten, dass keine integrierten Bausteine/Kopfzeilen/Fußzeilen oder ähnliche sonst verfügbare Auswahllisten angezeigt werden, könnte die *Building Blocks.dotx* defekt sein.

Suchen Sie in diesem Fall diese Datei und benennen Sie sie um. Word erstellt beim nächsten Start automatisch eine neue Datei mit den Standard-Auswahllisten und Bausteinen.

## Die Dokumenteigenschaften

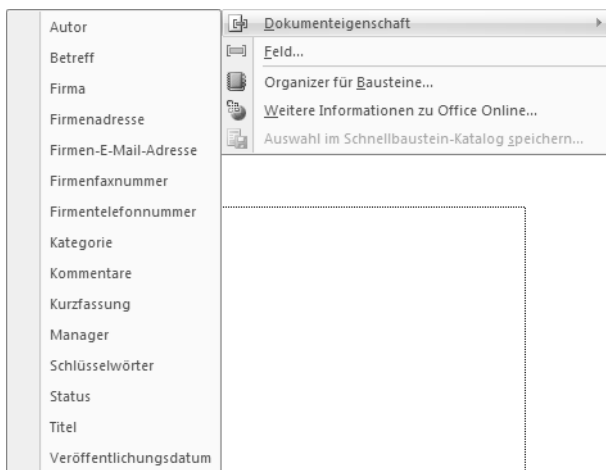
Der erste Menüeintrag in den Schnellbausteinen betrifft die Dokumenteigenschaften.

Bei diesen Dokumenteigenschaften handelt es sich um eine fest definierte Anzahl von Eigenschaften des jeweils aktiven Dokumentes; sie sind also dokumentenspezifisch.

Mit einem Mausklick auf den Menüeintrag *Dokumenteigenschaft* öffnet sich ein Untermenü mit allen verfügbaren Dokumenteigenschaften.

**WICHTIG** Die Dokumenteigenschaften stehen nur für Dateien im Word 2007-Format zur Verfügung: Also *docx*-, *dotx*- (normale Dateien ohne Makros) und *docm*- sowie *dotm*- (Dateien mit Makros) Dateien. Bei älteren Dateien, die im Kompatibilitätsmodus geöffnet werden, ist dieser Menüpunkt nicht verfügbar!

Abbildg. 6.10 Zugriff auf Dokumenteigenschaften über die Schnellbausteine



Über diese aufgeführten Eigenschaften können Sie direkt auf die wichtigsten Dokumenteigenschaften des aktiven Dokumentes zugreifen. Durch Auswahl eines Eintrags wird diese Dokumentinformation als Inhaltssteuerelement an die markierte Stelle in das Dokument eingefügt. Das Inhaltssteuerelement ist fest mit der Dokumenteigenschaft verknüpft: wird das eine geändert, wird der Wert im anderen wiedergegeben.

**HINWEIS** Mehr zum Thema Verknüpfung von Dokumenteigenschaften mit Inhaltssteuerelementen erfahren Sie in Kapitel 7.

**HINWEIS**

Diese Dokumenteigenschaften entsprechen denen, die Sie auch über *Office/Vorbereiten/Eigenschaften* aufrufen und ändern können.

Mit der folgenden Anweisung können Sie diese Eigenschaften aufrufen und auch wieder ausblenden:

```
Application.DisplayDocumentInformationPanel = False
```

Weitere Möglichkeiten, auf diese Felder oder die Eigenschaften aus VBA zuzugreifen, hat Microsoft leider nicht vorgesehen. Sie können nur auf normalem Weg über die Dokumenteigenschaften (ActiveDocument.BuiltInDocumentProperties-Eigenschaft und ActiveDocument.CustomDocumentProperties-Eigenschaft) auf die verschiedenen Felder zugreifen.

## Felder (Feldfunktionen)

Der nächste Menüeintrag ist der Eintrag *Feld* (Abbildung 6.10). Über diesen rufen Sie das Dialogfeld zum Einfügen von Feldern und zur Erstellung von Feldfunktionsausdrücken auf.

Dieses Dialogfeld ist identisch mit dem aus älteren Word-Versionen, das Sie über den Menüpunkt *Einfügen/Feld* aufrufen.

Aus VBA heraus rufen Sie dieses Dialogfeld wie bisher über die entsprechende Dialogs-Konstante wdDialogInsertField auf. Das nachstehende Listing 6.16 ruft das Dialogfeld zum Einfügen eines Feldes auf und zeigt den Code der ausgewählten Feldfunktion an.

**Listing 6.16** Aufruf des Dialogfeldes *Feld*

```
Sub BspFeldDialog()
' Zeigt das Feld-Dialogfeld an
' wird der Dialog mit "OK" bestätigt, _
  wird der Code der ausgewählten Feldfunktion angezeigt
Dim dlg As Dialog
Set dlg = Dialogs(wdDialogInsertField)
With dlg
  If .Display = -1 Then MsgBox dlg.Field
End With
End Sub
```

Weitere Informationen zum Umgang mit Feldern und Dialogfeldern finden Sie in Kapitel 7 bzw. Kapitel 15. Daher soll an dieser Stelle nicht weiter auf die Feldfunktionen eingegangen werden.

Auch für diesen Schnellbaustein hat Microsoft keine besonderen Befehle zur Verfügung gestellt, so dass Sie ganz normal über die Fields-Eigenschaft Felder erstellen und darauf zugreifen können.

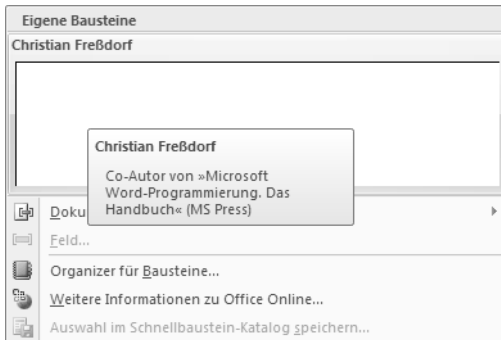
## Der Schnellbaustein-Katalog

Wie am Anfang des Kapitels angemerkt, können Sie auch Bausteine direkt im Schnellbaustein-Menü für den direkten Zugriff ablegen (daher auch der Name des Menüpunkts).

Dazu müssen Sie den gewünschten Bereich (Text und ggf. weitere Elemente) markieren, bevor Sie ihn über den Befehl *Auswahl im Schnellbaustein-Katalog speichern* in das Schnellbaustein-Menü

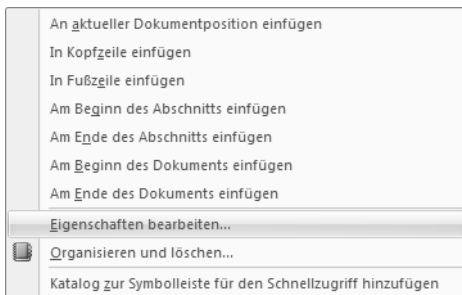
übernehmen können. Der ausgewählte Text (ggf. mit weiteren Elementen) wird dann im Menü mit einer Voransicht angezeigt.

Abbildg. 6.11 Festlegen eines markierten Textes als Schnellbaustein im Schnellbaustein-Menü



Jeder Baustein wird durch verschiedene Eigenschaften beschrieben. Sie können diese Eigenschaften anzeigen, wenn Sie zunächst mit der rechten Maustaste einen Schnellbausteineintrag anklicken und im daraufhin geöffneten Kontextmenü den Eintrag *Eigenschaften bearbeiten* wählen.

Abbildg. 6.12 Anzeigen und bearbeiten der Bausteineigenschaften



Daraufhin wird das Dialogfeld zum Bearbeiten der Bausteineigenschaften geöffnet.

Abbildg. 6.13 Bearbeiten der Bausteineigenschaften



Mit diesen Eigenschaften wird jeder (Schnell-)Baustein beschrieben. Es handelt sich dabei um die Eigenschaften mit Informationen, die nur für den jeweiligen Baustein gelten: Name, Description (Beschreibung), Type (Katalog), Category (Kategorie) und Value (Inhalt) .

Wenn Sie einen neuen Schnellbaustein per VBA erzeugen, erfolgt dies über die `Template.BuildingBlockEntries`-Eigenschaft und dabei über die `Add`-Methode.

Dabei muss es sich bei dem `Template`-Objekt entweder um eine geladene Dokumentvorlage (*dotx*) oder um die spezielle Vorlage *Building Blocks.dotx* (siehe den Abschnitt »Die zentrale Bausteinvorlage *Building Blocks.dotx*«) handeln. Über die Parameter der `Add`-Methode wird dann der Baustein entsprechend der Abbildung 6.13 festgelegt.

**Listing 6.17** Markierten Text per Makro als Schnellbaustein speichern

```
Sub subSchnellbausteinErstellen()
Dim objTemplate As Template
Dim objBB As BuildingBlock
Templates.LoadBuildingBlocks , Lädt die Building Blocks.dotx
Set objTemplate = Templates(1)
Set objBB = objTemplate.BuildingBlockEntries.Add( _
    Name:="Christian Freßdorf", _
    Type:=wdTypeQuickParts, _
    Category:="Eigene Bausteine", _
    Range:=Selection.Range, _
    Description:="Co-Autor von »Microsoft Word-Programmierung. Das Handbuch« (MS Press)", _
    InsertOptions:=wdInsertContent)
End Sub
```

**ACHTUNG** Leider lassen sich in Word 2007 mit Einführung der Multifunktionsleiste nicht mehr alle Funktionsaufrufe und Funktionen mit dem Makrorekorder aufzeichnen. So zeichnet beispielsweise in diesem Fall der Makrorekorder für das Erstellen eines Schnellbausteines den folgenden VBA-Code auf:

```
Sub SchnellbausteinErstellenRekorder()
WordBasic.CreateCommonFieldBlockFromSel Description:="Autor"
End Sub
```

Abgesehen davon, dass die `WordBasic`-Befehle schon seit Einführung von VBA in Office 97 nur noch aus »Kompatibilitätsgründen« weitergeführt wurden, ist dieser Befehl auch unvollständig, wie ein Vergleich mit Abbildung 6.13 zeigt, da nur die Beschreibung angegeben werden kann. Leider lassen sich die fehlenden Eigenschaften nicht durch zusätzliche Parameter im Aufruf hinzufügen, da es dann zu einem Syntaxfehler kommt.

Wenn Sie beispielsweise eine Grafik auf dem markierten Absatz verankern und die Grafik dann hinter den Text legen und positionieren, können Sie so den Text mit der Grafik als Schnellbaustein speichern (Abbildung 6.14). Sie müssen dabei nur beachten, dass der Anker der Grafik mit im markierten Bereich liegt.

Der Makroaufruf bleibt in diesem Fall gleich.

Sobald Sie einen Schnellbaustein angelegt haben, wird dieser auch im Organizer für Bausteine aufgeführt (siehe den Abschnitt »Der Organizer für Bausteine« in diesem Kapitel).

Abbildg. 6.14 Markierten Text mit Bild im Hintergrund als Schnellbaustein festlegen



Jeden erstellten und angezeigten Schnellbaustein können Sie an verschiedene Stellen im aktuellen Dokument einfügen. Klicken Sie dazu mit der rechten Maustaste auf den jeweiligen Baustein, um das Kontextmenü mit den Einfügeoptionen (im oberen Bereich der Abbildung 6.12) aufzurufen. Diese Einfügeoptionen sind soweit selbst erklärend, so dass darauf nicht weiter eingegangen wird.

Die Schnellbausteine (also die Einträge im Schnellbaustein-Katalog) in diesem Menü sind Bestandteil der neuen Bausteine: der BuildingBlocks-Auflistung. Dabei stellen sie einen besondern Typen dieser BuildingBlocks dar, auf den über die `wdBuildingBlockTypes`-Konstante `wdTypeQuickParts` zugegriffen werden kann.

Da die Schnellbausteine an verschiedenen Stellen (Vorlagen) gespeichert werden können, werden im Listing 6.18 alle geladenen Vorlagen durchlaufen und für jede Vorlage der Schnellbausteintyp `wdTypeQuickParts` auf Einträge (BuildingBlocks) überprüft.

Da die Baustein-Einträge im Baustein-Katalog über die Reihenfolge

1. Vorlage (Eigenschaft: *Speichern in*)
2. Schnellbausteine (Eigenschaft: *Katalog=Schnellbaustein*)
3. Kategorie (Eigenschaft: *Kategorie*)

gespeichert werden, muss auch der Zugriff auf die Schnellbausteine in dieser Reihenfolge erfolgen. Dazu wird für jede Vorlage zuerst überprüft, ob im Schnellbaustein-Katalog (`BuildingBlockTypes(wdTypeQuickParts)`) Kategorien (`Categories.Count`) vorhanden sind. Ist dies der Fall, wird für jede vorhandene Kategorie geprüft, ob Einträge (`BuildingBlocks.Count`) vorhanden sind und wie sie eingefügt werden sollen (`InsertOptions`). Diese Informationen werden dabei gesammelt und am Ende angezeigt (Abbildung 6.15).

Listing 6.18 Übersicht über die Schnellbausteine

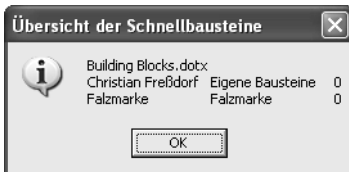
```
Sub subSchnellbausteineAuflisten()
    Dim objTemplate As Template
    Dim objBBT As BuildingBlockType
    Dim objBB As BuildingBlock
    Dim objCat As Category
    Dim intCount As Integer
    Dim intCountCat As Integer
    Dim strList As String
    Templates.LoadBuildingBlocks
    strList = "Keine Schnellbausteine gefunden"
    For Each objTemplate In Templates
        Set objBBT = objTemplate.BuildingBlockTypes(wdTypeQuickParts)
        If objBBT.Categories.Count > 0 Then
            strList = objTemplate & vbCrLf
            For intCount = 1 To objBBT.Categories.Count
                Set objCat = objBBT.Categories(intCount)
```

**Listing 6.18** Übersicht über die Schnellbausteine (Fortsetzung)

```

        For intCountCat = 1 To objCat.BuildingBlocks.Count
            Set objBB = objCat.BuildingBlocks(intCountCat)
            strList = strList & objBB.Name & vbTab & objCat.Name & _
                vbTab & objBB.InsertOptions & vbCrLf
        Next intCountCat
    Next intCount
End If
Next objTemplate
MsgBox strList, vbInformation, "Übersicht der Schnellbausteine"
End Sub

```

**Abbildg. 6.15** Anzeige verfügbarer Schnellbausteine


Für die InsertOptions-Eigenschaft stehen folgende wdDocPartInsertOptions-Konstanten zur Verfügung (siehe auch Abbildung 6.13, Feld *Optionen*).

**Tabelle 6.8** wdDocPartInsertOptions-Parameter zum Einfügen von (Schnell-)Bausteinen

Name	Wert	Beschreibung
wdInsertContent	0	Inlinebaustein Fügt nur den Inhalt des Schnellbausteins in den Fließtext ein
wdInsertPage	2	Seitenebenen-Baustein Fügt den Inhalt des Schnellbausteins auf einer eigenen Seite ein
wdInsertParagraph	1	Absatzebenen-Baustein Fügt den Inhalt des Schnellbausteins in einen eigenen Absatz ein

Möchten Sie einen Schnellbaustein in das Dokument einfügen, kennen jedoch nur die Kategorie, nicht aber den Speicherort, müssen Sie alle Schnellbausteine in allen Vorlagen durchlaufen und die Einträge auf Namen und Kategorie überprüfen. Da es leider keine For Each...Next-Anweisung für diese neuen Objekte gibt, müssen Sie die Einträge mit For...Next-Anweisungen durchlaufen.

**Listing 6.19** Einfügen eines Schnellbausteins ohne Kenntnis des Speicherortes

```

Sub subSchnellbausteinInTextEinfügen()
    Dim objTemplate As Template
    Dim objBBT As BuildingBlockType
    Dim objBB As BuildingBlock
    Dim intCount As Integer, intBBCount As Integer
    Templates.LoadBuildingBlocks
    For Each objTemplate In Templates
        Set objBBT = objTemplate.BuildingBlockTypes(wdTypeQuickParts)
        For intCount = 1 To objBBT.Categories.Count
            If objBBT.Categories(intCount).Name = "Eigene Bausteine" Then

```



Listing 6.19 Einfügen eines Schnellbausteins ohne Kenntnis des Speicherortes (Fortsetzung)

```

    For intBBCount = 1 To objBBT.Categories(intCount).BuildingBlocks.Count
        Set objBB = objBBT.Categories("Eigene Bausteine").BuildingBlocks(intCount)
        If objBB.Name = "Christian Freßdorf" Then
            objBB.Insert Selection.Range, True
        Exit For
    End If
    Next intBBCount
End If
Next intCount
Next objTemplate
End Sub

```

Wenn Sie den Schnellbaustein an einem der im Kontextmenü des Bausteins vorgeschlagenen Dokumentort einfügen möchten, müssen Sie dies bei Verwendung der `BuildingBlock.Insert`-Methode explizit angeben: Entweder durch Angabe des entsprechenden `Range`-Objektes oder indem Sie vorher an die Stelle im Dokument springen und dann das `Selection.Range`-Objekt verwenden.

**ACHTUNG** Wenn Sie das Einfügen eines Schnellbausteins an eine Dokumentstelle aus dem Kontextmenü mit dem Makrorekorder aufzeichnen, wird der Wechsel zur ausgewählten Dokumentstelle **nicht** mit aufgezeichnet.

Es wird auch nicht berücksichtigt, ob der Baustein aus den Schnellbausteinen oder einem anderen Katalog eingefügt wird, wenn mehrere Bausteine mit gleichem Namen existieren. Der Rekorder verwendet immer den ersten Eintrag mit dem angegebenen Baustein-Namen, den er im Organizer findet. Dies unabhängig von der Kategorie und davon, ob es der ausgewählte Baustein ist.

## Die Bausteine (*BuldingBlockTypes*)

Wie schon angedeutet, stellen die Schnellbausteine einen besonderen Typen innerhalb der Bausteine (des `BuildingBlockTypes`-Objektes) dar. Insgesamt stehen die 32 in Tabelle 6.7 aufgeführten Bausteintypen zur Verfügung.

Zur besseren Organisation können Sie Bausteine in Kategorien innerhalb eines Bausteintyps ordnen.

Wie im Abschnitt »Der Schnellbaustein-Katalog« beschrieben, wird ein Baustein durch die Eigenschaften *Vorlage*, *Katalog*, *Kategorie* und *Name* eindeutig festgelegt.

Zugriff auf einen bestimmten Baustein erhalten Sie dabei über das `BuildingBlock`-Objekt, das genau einen Baustein eines bestimmten Typen (`BuildingBlockTypes`) und einer bestimmten Kategorie (`Category`) darstellt. Im Listing 6.20 werden für einen Baustein aus der *Building Blocks.dotx* die verschiedenen Informationen (*Index*, *Vorlage*, *Katalog*, *Kategorie*) ausgegeben (Abbildung 6.16).

Listing 6.20 Ausgabe der Bausteininformationen

```

Sub ShowBuildingBlockInformation()
    Const c_Titel As String = "Bausteininformationen"
    Dim objTemplate As Template
    Dim i As Variant
    Dim objBB As BuildingBlock
    Templates.LoadBuildingBlocks

```

**Listing 6.20** Ausgabe der Bausteininformationen (Fortsetzung)

```

Set objTemplate = FindBuildingBlocksDOTX
i = InputBox("Bitte Bausteinnummer angeben ( _  
1-" & objTemplate.BuildingBlockEntries.Count & ")", "Bausteininformationen", "10")
If IsNumeric(i) = False Then
    MsgBox "Keine Zahl eingegeben oder Baustein nicht vorhanden", vbCritical, c_Titel
    Exit Sub
End If
On Error Resume Next
Set objBB = objTemplate.BuildingBlockEntries(CInt(i))
If objBB Is Nothing Then
    MsgBox "Baustein nicht vorhanden", vbCritical, c_Titel
    Exit Sub
End If
On Error GoTo 0
MsgBox "Name: " & vbTab & objBB.Name _  
    & vbCrLf & "Index: " & vbTab & objBB.Index _  
    & vbCrLf & "Kategorie: " & vbTab & objBB.Category.Name _  
    & vbCrLf & "Katalog: " & vbTab & objBB.Type.Name _  
    & vbCrLf & "Vorlage: " & vbTab & objTemplate.Name, vbInformation, c_Titel
End Sub

```



Die Funktion *FindBuildingBlocksDOTX* finden Sie auf der CD-ROM zum Buch in der Datei *\Beispiele\Kap06\Kap06-BuildingBlocks.docm* im Modul *modBuildingBlocks*.

Als Ergebnis werden zu dem ausgewählten Baustein die Informationen, die diesen Baustein festlegen, ausgegeben.

**Abbildg. 6.16** Anzeige von Bausteininformationen


Zum Erstellen eines benutzerdefinierten Bausteinkatalogs stehen Ihnen zusätzlich eigene Kataloge zur Verfügung. Auf diese können Sie über die entsprechenden *wdBuildingBlockTypes*-Konstanten *wdTypeCustom<Typ/Nummer>* zugreifen.

Das folgende Beispiel *Bsp\_FalzmarkenBausteinErstellen* erstellt einen neuen Baustein *Falzmarke* im benutzerdefinierten Kopfzeilenkatalog und dort in der Kategorie *CHF Bausteine*. Dazu werden in einem neuen Dokument Falzmarken in die Kopfzeile eingefügt und anschließend diese Kopfzeile als Baustein abgespeichert.

**Listing 6.21** Erstellen von Falzmarken und Speichern als benutzerdefinierten Kopfzeilen-Baustein

```

Sub Bsp_FalzmarkenBausteinErstellen()
    Dim oDoc As Document
    Set oDoc = funcBsp06_Falzmarke

```

Listing 6.21 Erstellen von Falzmarken und Speichern als benutzerdefinierten Kopfzeilen-Baustein (Fortsetzung)

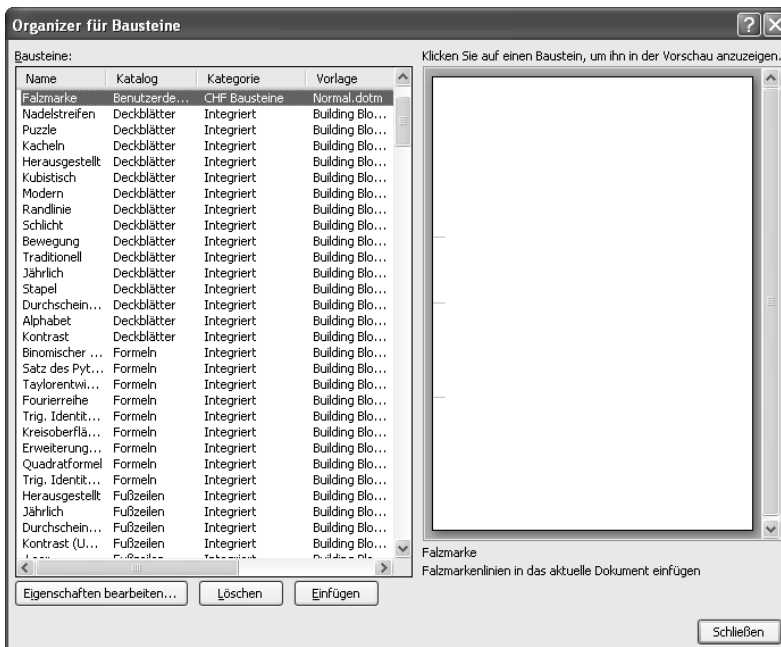
```

Dim objTemplate As Template
Dim conType As WdBuildingBlockTypes
Dim objRange As Range
Dim intHdr As Integer
Dim hdr As HeaderFooter
Set objTemplate = ActiveDocument.AttachedTemplate
conType = wdTypeCustomHeaders
If oDoc.Sections(1).PageSetup.DifferentFirstPageHeaderFooter Then
    intHdr = wdHeaderFooterFirstPage
Else
    intHdr = wdHeaderFooterPrimary
End If
Set hdr = oDoc.Sections(1).Headers(intHdr)
Set objRange = hdr.Range
objTemplate.BuildingBlockEntries.Add _
    Name:="Falzmarke", _
    Type:=conType, _
    Category:="CHF Bausteine", _
    Range:=objRange, _
    Description:="Falzmarkenlinien in das aktuelle Dokument einfügen"
oDoc.ActiveWindow.View = wdPrintView
End Sub

```

Im Organizer (siehe den Abschnitt »Der Organizer für Bausteine«) finden Sie daraufhin den neu erstellten Baustein *Falzmarke* im Katalog *Benutzerdefinierte Kopfzeile* unter der Kategorie *CHF Bausteine*. Gespeichert ist dieser Baustein in der *Normal.dotm*.

Abbildg. 6.17 Organizer mit neu erstelltem, benutzerdefiniertem Kopfzeilen-Baustein





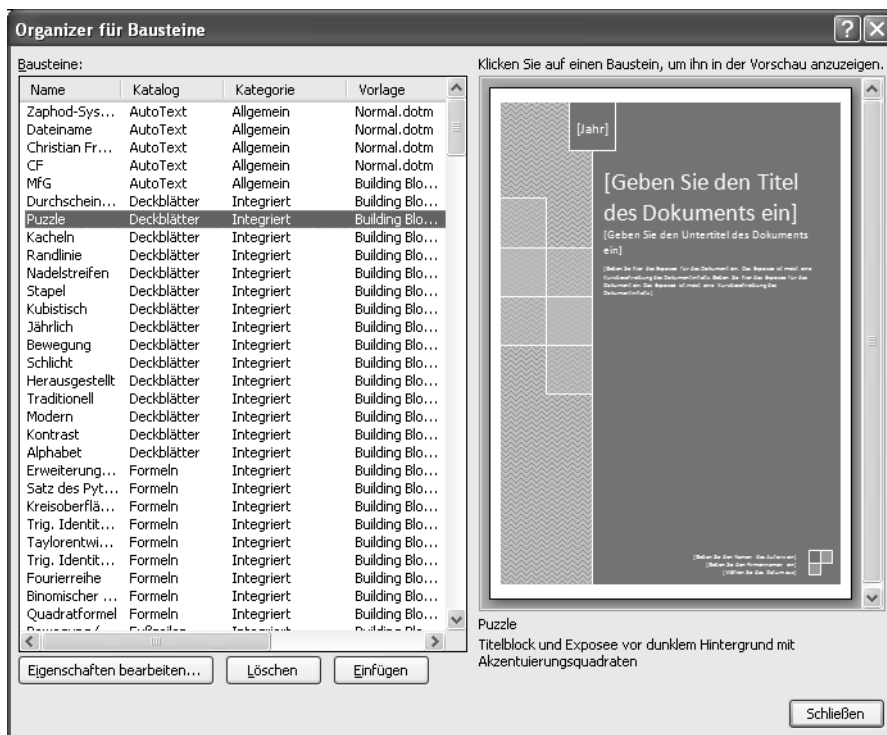
Das vollständige Beispiel mit allen benötigten Funktionen finden Sie auf der CD-ROM zum Buch in der Datei `\Beispiele\Kap06\Kap06-BuildingBlocks.docm` im Modul `modFalzmarkeErstellen`.

## Der Organizer für Bausteine

Der Organizer für alle Bausteine, inkl. der Schnellbausteine, ist die zentrale Verwaltungsstelle für die Bausteine. Über den Organizer können Sie die Bausteine verwalten, in andere Vorlagen verschieben, umbenennen und löschen.

Den Organizer rufen Sie über den Menüpunkt *Organizer für Bausteine* auf:

Abbildg. 6.18 Organizer-Dialog für die Bausteine



Im Objektmodell erreichen Sie den entsprechenden Dialog über den `Dialogs`-Typen `wdDialogBuildingBlockOrganizer`:

```
Dialogs(wdDialogBuildingBlockOrganizer).Show
```

Leider lässt sich dieses Dialogfeld nicht einschränken oder per Parameter vorsortieren, da dieses `Dialogs`-Objekt keine Parameter besitzt.

Über die drei Schaltflächen *Eigenschaften bearbeiten*, *Löschen* und *Einfügen* rufen Sie das Eigenschaftendialogfeld auf (Abbildung 6.12), über das Sie durch Zuweisen eines anderen Speicherortes den markierten Baustein verschieben (organisieren) können, löschen den markierten Baustein oder fügen ihn an der Einfügemarke in den Text ein.

## Ein Organizer mit erweiterter Funktionalität

In der Beispieldatei *Kap06-BuildingBlocks.docm* finden Sie eine neue Registerkarte *BuildingBlocks*. Wenn Sie auf diese Registerkarte klicken, sehen Sie eine neue Gruppe mit einer Reihe von Symbolen angezeigt, über die Sie die verschiedenen Beispiele zu diesem Kapitel aufrufen können.



Über dieses Symbol mit dem Namen *Userform anzeigen* rufen Sie eine UserForm auf, auf der Sie alle verfügbaren Bausteine (in allen geladenen Vorlagen) angezeigt bekommen. Über die Auswahllisten (*Vorlage*, *Katalog*, *Kategorie*) können Sie dann die Bausteine, die im Organizer ja nicht eingeschränkt werden können, auf einzelne Vorlagen, Kataloge oder Kategorien einschränken.

Abbildg. 6.19 Übersicht über alle Bausteine mit der Möglichkeit der Anzeigeeinschränkung

Name	Katalog	Kategorie	Vorlage
Christian Freßdorf	Schnellbausteine	Eigene Bausteine	Building Bl...
Falzmarke	Schnellbausteine	Falzmarke	Building Bl...
Traditionell	Deckblätter	Integriert	Building Bl...
Randlinie	Deckblätter	Integriert	Building Bl...
Stapel	Deckblätter	Integriert	Building Bl...
Schlicht	Deckblätter	Integriert	Building Bl...
Alphabet	Deckblätter	Integriert	Building Bl...
Jährlich	Deckblätter	Integriert	Building Bl...
Kubistisch	Deckblätter	Integriert	Building Bl...
Modern	Deckblätter	Integriert	Building Bl...
Nadelstreifen	Deckblätter	Integriert	Building Bl...
Durchscheinend	Deckblätter	Integriert	Building Bl...
Herausgestellt	Deckblätter	Integriert	Building Bl...
Puzzle	Deckblätter	Integriert	Building Bl...
Bewegung	Deckblätter	Integriert	Building Bl...
Kacheln	Deckblätter	Integriert	Building Bl...
Kontrast	Deckblätter	Integriert	Building Bl...
Kreisoberfläche	Formeln	Integriert	Building Bl...
Binomischer Lehrsatz	Formeln	Integriert	Building Bl...
Erweiterung einer Su...	Formeln	Integriert	Building Bl...
Fourierreihe	Formeln	Integriert	Building Bl...
Satz des Pythagoras	Formeln	Integriert	Building Bl...

Abbildg. 6.20 Einschränkung der Anzeige auf bestimmte Bausteine (Vorlagen, Kataloge, Kategorien)

Name	Katalog	Kategorie	Vorlage
Christian Freßdorf	Schnellbausteine	Eigene Bausteine	Building Blocks
Falzmarke	Schnellbausteine	Falzmarke	Building Blocks

Mit einem Doppelklick auf einen Eintrag wird dieser Baustein an der Einfügemarke in das Dokument eingefügt.

Leider gibt es keinen dokumentierten Befehl, um das Dialogfeld »Baustein bearbeiten« (siehe Abbildung 6.13) aufzurufen. Wenn Sie einen Baustein ändern möchten, müssen Sie dies über den Organizer vornehmen.



Die Beispieldatei *Kap06-BuildingBlocks.docm* befindet sich im Ordner `\Beispiele\Kap06` auf der CD-ROM zum Buch.

## Formatieren mit Stil: Das *Style*-Objekt

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.
- Eine standardisierte Formatierung (»Corporate Identity«) kann besser durchgesetzt werden.

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 die Texterstellung durch 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

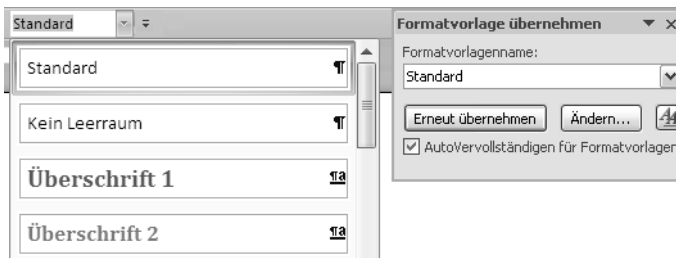


Ab Word 2002 gibt es vier wichtige Benutzerschnittstellen für die Arbeit mit Formatvorlagen in Word: die Dropdownliste in der *Format*-Symbolleiste (links in Abbildung 6.21), den Aufgabenbereich *Formatvorlagen und Formatierungen* (der über die Symbolschaltfläche links neben der Liste oder über *Format/Formatvorlagen und Formatierungen* eingeblendet wird) sowie benutzerdefinierte Symbolschaltflächen und Tastenkombinationen, denen über *Extras/Anpassen* auf der Registerkarte *Befehle* Formatvorlagen zugewiesen wurden. (Alle diese stehen früheren Word-Versionen auch zur Verfügung mit Ausnahme des Aufgabenbereichs.)



Davon kennt Word 2007 noch den Aufgabenbereich *Formatvorlagen*; die Dropdownliste kann der Schnellzugriffsleiste zugefügt werden (Abbildung 6.21), und Formatvorlagen können noch Tastenkombinationen zugewiesen werden. Neu gesellt sich dazu die Liste *Schnellformatvorlage* in der Registerkarte *Start* und der Aufgabenbereich *Formatvorlage übernehmen*. Letzterer ist als Ersatz für die alte Dropdownliste gedacht und wird mit der Tastaturkombination  $\text{Alt} + \text{Strg} + \text{S}$  eingeblendet. Sein größter Nachteil ist, dass er entweder über dem Text (und daher im Weg) liegt, oder, am Rand gedockt, zu viel Platz beansprucht.

**Abbildg. 6.21** Die alte Dropdownliste in der Symbolleiste für den Schnellzugriff sowie sein Ersatz, der neue Aufgabenbereich


**HINWEIS**

Dieser Aufgabenbereich wird in Kapitel 19 näher vorgestellt.

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 19) 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 in Word 2003 und früher auch Symbolleisten mit Symbolschaltflächen für Formatvorlagen. In Abbildung 6.22 sehen Sie eine solche, mit deren Hilfe dieses Buch erfasst wurde. Das Objektmodell bietet die Möglichkeit in diesen Versionen, Symbolleisten zu erstellen, zu positionieren und zu schützen. Mehr darüber lesen Sie in Kapitel 16. Hierfür gibt es in Word 2007 keinen gleichwertigen Ersatz, der mit dem Word-Objektmodell realisierbar ist. Der Entwickler eines COM-Add-Ins könnte dazu ein Custom Task Pane (siehe Kapitel 10) einsetzen.

Ihnen ist sicher aufgefallen, dass sich auf jeder Symbolschaltfläche der Abbildung 6.22 ein Tastaturkürzel befindet. Auch diese können programmtechnisch erstellt werden, über die KeyBindings-Auflistung, die in Kapitel 18 vorgestellt wird.

**Abbildg. 6.22** 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.



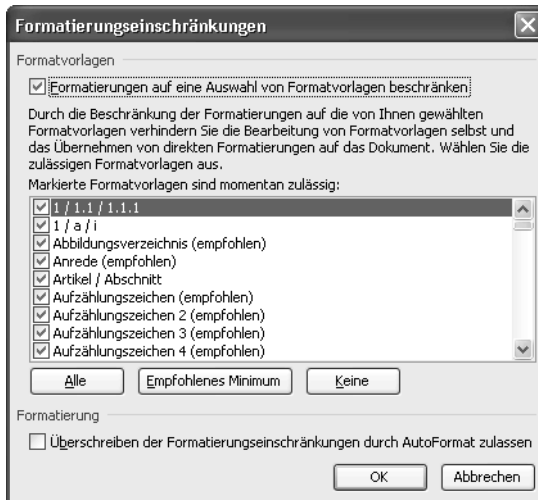
Ab Word 2003 dagegen steht eine Dokumentschutzoption zur Verfügung, um die im Dokument erlaubten Formatierungen festzulegen. In der Benutzerschnittstelle befindet sie sich im Aufgabenbereich *Dokument schützen* (*Extras/Dokument schützen* bzw. in Word 2007 die Schaltfläche *Dokument schützen* auf der Registerkarte *Entwicklertools* in der Gruppe *Schützen*). Durch Anklicken des Links *Einstellungen* wird das in Abbildung 6.23 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.



In Word 2007 ist die Funktionalität auch über den Aufgabenbereich *Formatvorlagen* zugänglich. Im Dialogfeld *Formatvorlage verwalten* befindet sich eine Registerkarte *Einschränken*, die eine erweiterte Auswahl an Möglichkeiten anbietet. Mehr zu diesem Thema lesen Sie im Buch »Office 2007 – das Profibuch«.

Erst nach Aktivierung des Dokumentschutzes im Aufgabenbereich *Dokument schützen* wird die Einschränkung wirksam. Der Schutz kann mit einem Kennwort versehen werden.

**Abbildg. 6.23** Hier wird festlegt, 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

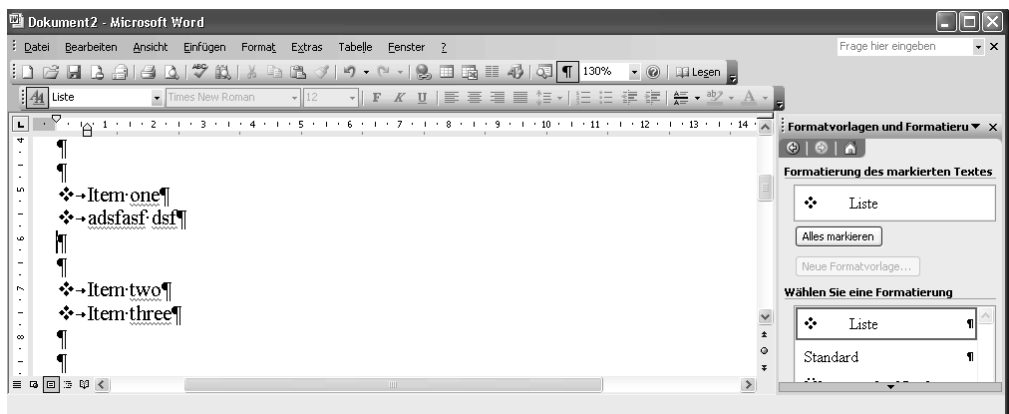
mit der Formatvorlage arbeiten. »wahr« bedeutet, dass die Formatvorlage in diesem Kontext gesperrt ist.

Das untere Kontrollkästchen steht für die Dokumentoption `AutoFormatOverride`. Ist diese Einstellung aktiviert, dürfen die Optionen *Format/AutoFormat* und *AutoFormat während der Eingabe* ausgeführt werden. (In Word 2007 befindet sich *AutoFormat* nicht in der Multifunktionsleiste; der Befehl kann jedoch der Symbolleiste für den Schnellzugriff zugewiesen 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 6.24 veranschaulicht. Obwohl die Formatvorlagen gesperrt sind, hat AutoFormat nach zweimaligem Drücken der `[↵]`-Taste das Symbol der Formatvorlage *Liste* aus dem Text entfernt. Der Dropdownliste *Formatvorlage* sowie dem Aufgabenbereich ist zu entnehmen, dass die Formatvorlage der Markierung noch *Liste* und nicht *Standard* ist.

Abbildg. 6.24 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 6.22 veranschaulicht.

Zudem entfernt diese Prozedur alle im Dokument vorhandenen unerwünschten 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 6.22** 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 6.23** 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)
        {

```

**Listing 6.23** Die C#-Version (Fortsetzung)

```

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 13. Ein Beispiel für dieses Vorgehen ist im Abschnitt »Feldfunktion« in Kapitel 7 beschrieben.

**NameLocal.** 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 Word-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 Word-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 6.24** 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
```

**Listing 6.24** Alle nicht Word-internen Formatvorlagen in einem neuen Dokument auflisten (Fortsetzung)

```
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 & vbCrLf
    End If
Next
End Sub
```

**Listing 6.25** 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 zum Buch 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 6.26 führt die Angaben im Code auf. Die `AutoOpen`-Prozedur wird beim Öffnen des Dokuments automatisch ausgeführt.

**Listing 6.26** Die Namen von Formatvorlagen der Umgebungssprache anpassen

```
Private Const m_LANZFV As Long = 2
Private Const m_LANZSPRACHEN As Long = 3
Private Const INDEX_DEUTSCH As Long = 0
Private Const INDEX_ENGLISCH As Long = 1
```

Listing 6.26 Die Namen von Formatvorlagen der Umgebungssprache anpassen (Fortsetzung)

```

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_LANZFV - 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"
    aFVNamen(INDEX_FRANZ, 0) = "Test1"
    aFVNamen(INDEX_FRANZ, 1) = "Test2"
End Sub

```

**Listing 6.26** Die Namen von Formatvorlagen der Umgebungssprache anpassen (*Fortsetzung*)

```
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 zum Buch 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.** 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*-Konstantenwerte (Word 2000 erkennt nur die beiden ersten): *wdStyleTypeParagraph* (Absatz), *wdStyleTypeCharacter* (Zeichen), *wdStyleTypeList* (Nummerierung) oder *wdStyleTypeTable* (Tabelle).



2007

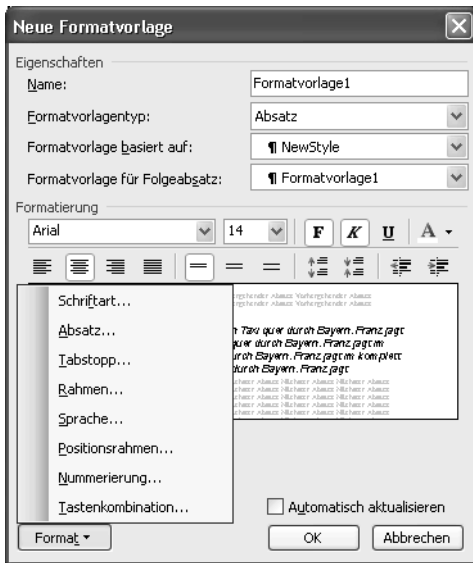
In Word 2007 kommen dazu die Typen *wdStyleTypeLinked* (verknüpft = kann als Zeichen sowohl wie Absatzformatvorlage dienen) und *wdStyleTypeParagraphOnly* (nur Absatz = nicht verknüpft). Diese werden eingehender in Kapitel 19 erklärt. Das Prinzip der verknüpften Formatvorlage entnehmen Sie bitte dem Abschnitt »Wilde Formatvorlagen« in diesem Kapitel.


In der Benutzerschnittstelle 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 in der Benutzerschnittstelle findet im Dialogfeld *Formatvorlage erstellen* (Abbildung 6.25) mit den Menüs unter der Schaltfläche *Format* statt. Da diese den Dialogfeldern des Menüs *Format* (Word 2003 und früher) in der Menüleiste entsprechen, ist das automatische Erstellen von Formatvorlagen ähnlich der Formatierung von Text, wie auch das Listing 6.27 zeigt.



Abbildg. 6.25 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 (NextParagraph-Style-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 6.25 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 6.26 zu sehen ist.

Listing 6.27 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 6.27** 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 6.28** 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 6.28 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;
}

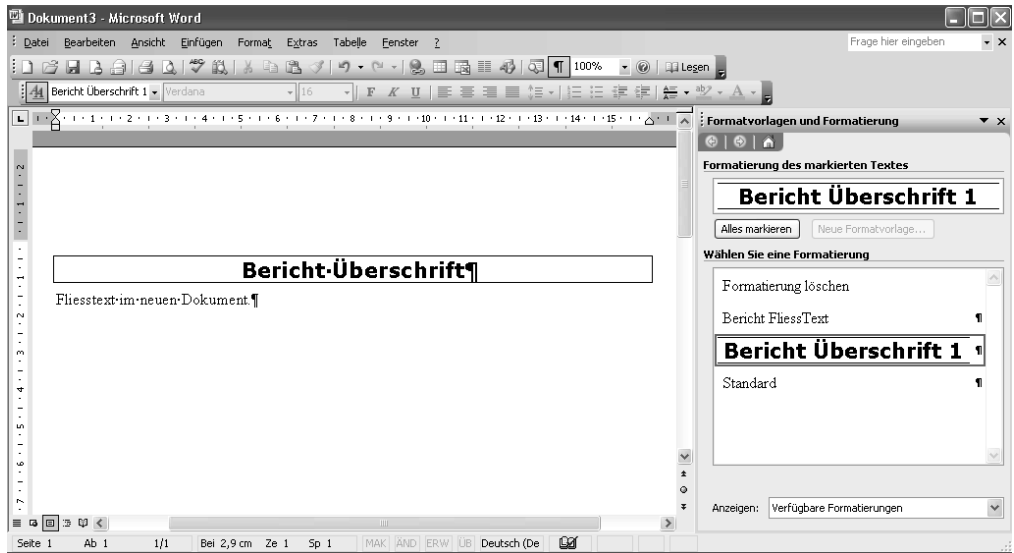
private void BerichtSchreiben_CS(wd.Document doc)
{
    wd.Range rng = doc.Content;
    rng.Text = "Bericht Überschrift\n";
    object stylÜberschrift1 = (object) "Bericht Überschrift 1";
    rng.set_Style(ref stylÜberschrift1);
    object objCollapseEnd = (object) wd.WdCollapseDirection.wdCollapseEnd;
    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 zum Buch im Ordner *\Beispiele\Kap06*. Bitte beachten Sie, dass das Sperren von Formatvorlagen (die Locked-Eigenschaft sowie der Dokumentschutz für Formatvorlagen) nur ab Word 2003 funktioniert. Das Erstellen und die Anwendung von Formatvorlagen bleibt in allen Versionen von Word gleich.

Abbildg. 6.26 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

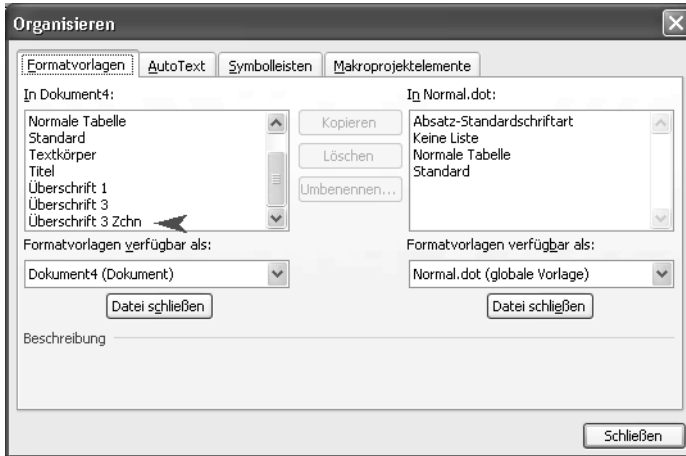
In Word 2002 führte Microsoft einige Änderungen der Formatvorlagenverwaltung ein mit der Absicht, die Anzahl verschiedener Formatierungen im Dokument zu reduzieren. Eine dieser Maßnahmen stellt eine Verbindung zwischen einer Formatvorlage und einer direkten Formatierung her, und zwar dann, wenn der Benutzer 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 verknüpft ist. Seit einem Service Pack für Word 2002 sieht man diese zwar nicht mehr im Aufgabenbereich, wohl aber unter *Organisieren* (Abbildung 6.27). Sie erscheint auch, wenn ein Dokument in Word 2000 oder Word 97 geöffnet und gespeichert wird.



Aus diesem Grund wurden die neuen *wdStyleType*-Typen in Word 2007 eingeführt. Mit dem Typ *wdStyleTypeParagraphOnly* erstellt Word keine verknüpfte Zeichenformatvorlage. Es ist auch möglich, im Aufgabenbereich mittels eines Kontrollkästchens, die automatische Verknüpfung in der Anwendung gänzlich auszuschalten. Im Objektmodell entspricht dies der Eigenschaft *Application.RestrictLinkedStyles*.

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.

**Abbildg. 6.27** 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.

**LinkStyle.** 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.

## 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* (automatische in statische Nummerierung umwandeln) und *RemoveNumbers* (Nummerierung entfernen).

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 6.29 und in Abbildung 6.28 veranschaulicht.

**Listing 6.29** 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

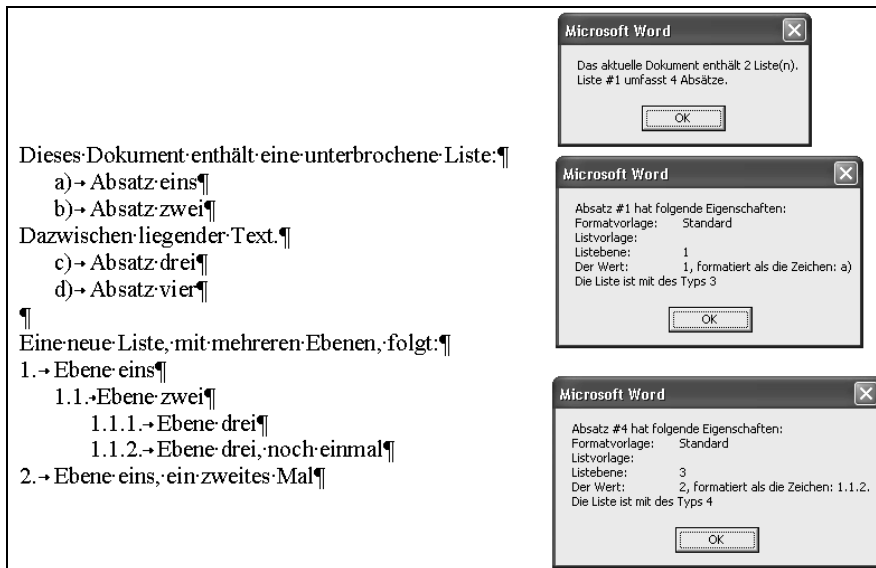
    Set doc = ActiveDocument
    lAnzListen = doc.Lists.Count
    For lListZaehler = 1 To lAnzListen
        Set lst = doc.Lists.Item(lListZaehler)
        lAnzParas = lst.ListParagraphs.Count
        MsgBox "Das aktuelle Dokument enthält " & CStr(lAnzListen) & " Liste(n)." & _
            vbCrLf & "Liste #" & 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:" & _
                & vbCrLf & "Formatvorlage:" & vbCrLf & sFormatvorlage & vbCrLf & _
                "Listvorlage: " & vbCrLf & sListvorlage & vbCrLf & "Listebene:" & vbCrLf & _
                sListEbene & vbCrLf & "Der Wert:" & vbCrLf & sOrdinalZahl & vbCrLf & _
                ", formatiert als die Zeichen: " & sVisuelleZiffer & vbCrLf & _
                "Die Liste ist vom Typ " & sListTyp
        Next
    Next
End Sub
```



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.

Abbildg. 6.28 Eigenschaften von Listen in einem Dokument ermitteln



## 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 auf eine Vorschau in *Format/Nummerierung und Aufzählungszeichen* klickt, 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 Hunderter von 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.

Der Benutzer hat 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 und Word 2007) 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 in Versionen vor 2007 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 Auskünfte 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 4 wurde die Vorbereitung der Dokumentvorlage *Normal.dot* besprochen, die allen Word-Dokumenten zu Grunde liegt.

---

**Name.** 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 6.30 veranschaulicht. Falls eine Listvorlage mit der angegebenen Bezeichnung bereits vorhanden ist, wird diese zurückgegeben. Sonst wird eine neue erstellt und zurückgegeben.

**Listing 6.30** 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

**ListLevel.** 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.

**OutlineNumbered.** 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 6.31 bzw. das Listing 6.32. 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 6.29 ersichtlich. Bitte beachten Sie, dass, um so wie in diesem Beispiel vorgehen zu können, die Absatz-Formatvorlage bereits im Dokument vorhanden sein muss, um eine Listvorlage damit zu verbinden.

---

**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 6.9 beschrieben.

**Listing 6.31** 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
```

Listing 6.31 Eine Listvorlage definieren (Fortsetzung)

```

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, _
    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 6.32 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",
        0f, 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.
}

```

**Listing 6.32** Die C#-Version (Fortsetzung)

```
//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);
}
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.LinkedStyle = LL_Formatvorlage;
    }
    catch (System.Exception ex)
    { System.Windows.Forms.MessageBox.Show(ex.Message); }
}
```

Abbildg. 6.29 Das Aufzählungsformat wurde programmtechnisch erstellt und mit der Formatvorlage verbunden

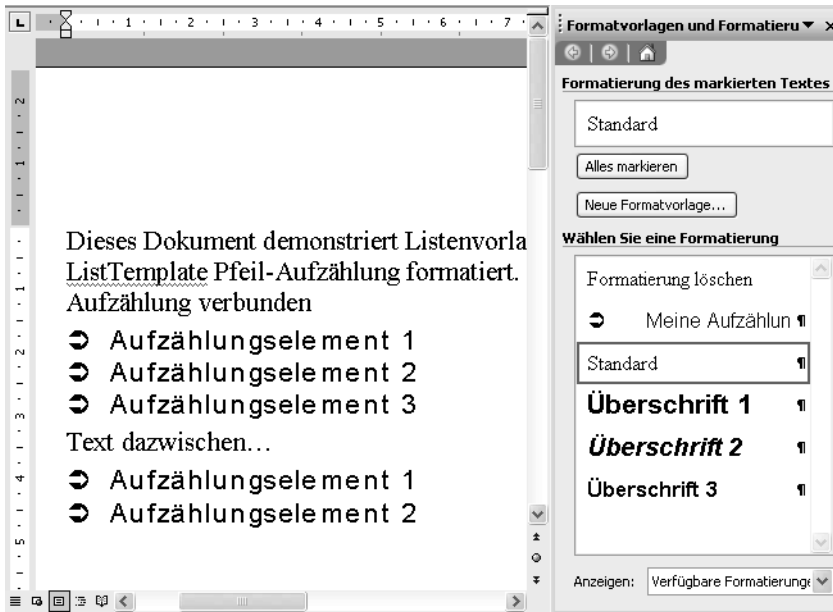


Tabelle 6.9 Die Eigenschaften, die die Formatierung einer jeden Ebene einer Aufzählung bestimmen

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 Schrifteigenschaften 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 Single, angegeben in Points. Stellt die Position der Aufzählung relativ zum linken Rand dar. Ein negativer Wert positioniert die Aufzählung am linken Textrand.

**Tabelle 6.9** Die Eigenschaften, die die Formatierung einer jeden Ebene einer Aufzählung bestimmen (Fortsetzung)

ListLevel-Eigenschaft	Mögliche Werte/Bemerkungen
NumberStyle	WdListNumberStyle-Konstantwert. Die üblichsten in westlichen Länder sind: wdListNumberStyleNone wdListNumberStyleArabic wdListNumberStyleBullet wdListNumberStyleLegal wdListNumberStyleLowercaseLetter wdListNumberStyleLowercaseRoman wdListNumberStyleUppercaseLetter wdListNumberStyleUppercaseLetter Legt die Art des Aufzählungszeichens fest (arabische, römische, Buchstaben usw.)
RestartOnHigher	Legt fest, ob die Nummerierung neu beginnt, wenn sie auf eine 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.
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 6.30 ersichtlich.

Die Listvorlage wurde mit dem Code in Listing 6.33 erstellt. Im Gegensatz zu Listing 6.31 wird eine Listvorlage des Typs Gliederung festgelegt, die neun Listebenen zur Verfügung stellt. Deren fünf werden in dieser Prozedur definiert.

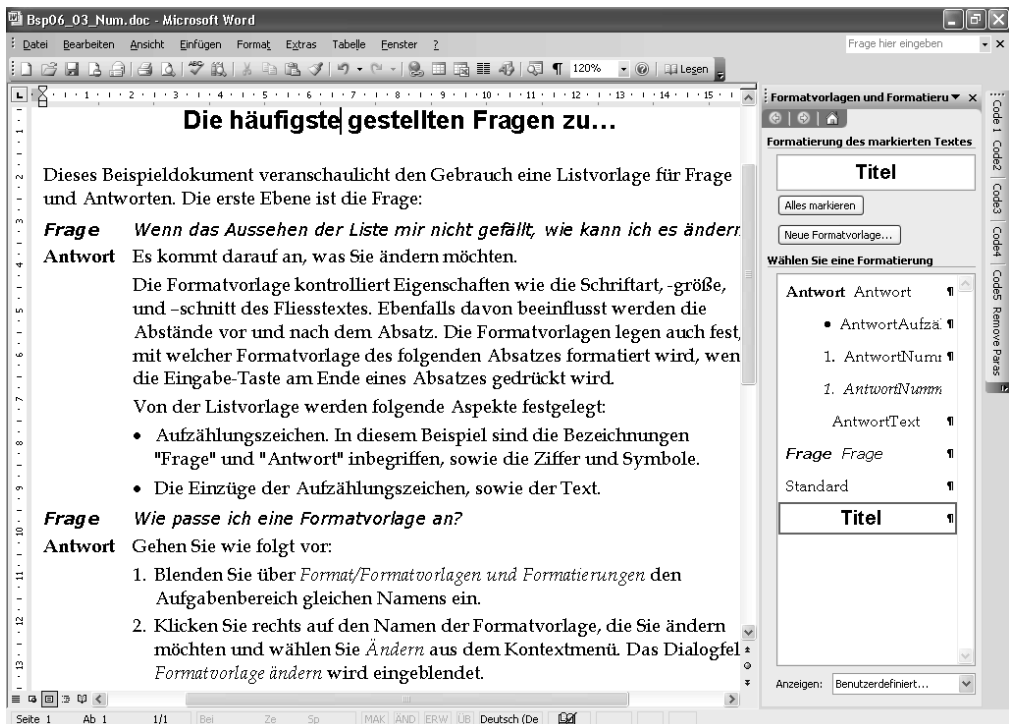
#### HINWEIS

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  $\square + \text{Alt} + \rightarrow$  die *AntwortNummeriert*-Formatierung angewendet werden. Ein nochmaliges Drücken formatiert den Absatz mit *AntwortAufzählung*. Die Tastenkombination  $\square + \text{Alt} + \leftarrow$  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. 6.30 Die ersten fünf Ebenen der Listvorlage definieren die Elemente eines Fragen-und-Antwort-Dokuments



**Listing 6.33** 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)
    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 6.34 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 6.34** 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 zusätzliche Absatz am

```



**Listing 6.34** Die Fragen und erster Absatz jeder Antwort als Zusammenfassung in ein neues Dokument übernehmen (*Fortsetzung*)

```
'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`.

## 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.



### HINWEIS

Für Office 2007 wurde die Grafik-Funktionalität neu entwickelt, um den erhöhten Ansprüchen der heutigen Welt zu entsprechen. Die neuen Möglichkeiten wurden vollständig in PowerPoint, aber auch in Excel integriert. Da in Word die Zusammenarbeit mit Grafiken komplexer ist, wurde hier die neue Funktionalität nur teilweise eingeführt. Gewisse Funktionalitäten, die der Anwender aus einer früheren Version erwartet, stehen ihm nicht mehr zur Verfügung, während sie im Objektmodell noch vorhanden sind. Andersherum kann der Programmierer auf einige der neuen Eigenschaften zugreifen, die dem Benutzer noch nicht zur Verfügung stehen. Eine verwirrende Situation, die sich hoffentlich bis zur nächsten Version klärt... Wir werden in diesem Kapitel aus Platzgründen diesen Eigenarten nicht nachgehen, sondern uns hauptsächlich auf den Umgang mit dem Objektmodell beschränken.

### InlineShape 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. Ab Word 2002 darf der Benutzer sogar die Methode selber bestimmen. (Bis Word 2007 über den Menübefehl *Extras/Optionen*, Registerkarte *Bearbeiten*, Option *Bild einfügen als*. Ab Word 2007 befindet sich die Option in der Kategorie *Erweitert*, im Abschnitt *Ausschneiden, Kopieren und Einfügen* der Word-Optionen.) Folglich muss der Code, wollen Sie alle Grafiken eines Dokuments bearbeiten, beide Auflistungen berücksichtigen.

## Grafiken einfügen

Bis und mit Word 2003 sind Einsichten in das Objektmodell für beide Auflistungen durch den Makrorekorder möglich. Es gibt aber einige Tricks, um eine Grafik nach deren Einfügung zu markieren, um sie weiter zu manipulieren. In Listing 6.35 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 Textflussformatierung.





Word 2007 zeichnet das Einfügen der Grafik korrekt auf. Im Gegensatz zu früheren Versionen sind die eingefügten Objekte markiert. Das nutzt uns herzlich wenig, da über die Multifunktionsleiste vorgenommene Formatierungen vom Makrorekorder nicht aufgezeichnet werden. Dies liegt wohl an der oben beschriebenen, teilweisen Integrierung der neuen Grafik-Funktionalität. Wem keine ältere Version von Word zur Verfügung steht, um ungewisse Punkte im Objektmodell nachzuforschen, wird es hart haben.

Während in früheren Word-Versionen ein `InlineShape` von einem `Shape` durch Farbe der Anfasser schnell erkennbar war, ist dies in Word 2007 nur im Kompatibilitäts-Format möglich. Grafiken, die in ein Word 2007-Dokument (*.docx*) eingefügt wurden, sind visuell nur durch Prüfen der Textumbruch-Formatierung erkennbar.

### Aus Datei mit `AddPicture`

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 in Word 2003 und früher 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 in Word 2003 und früher 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 6.35** 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
```

Listing 6.35 Dieser aufgezeichnete Code gibt Aufschluss über die *InlineShape*- bzw. *Shape*-Objekte (Fortsetzung)

```

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
    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 in Kapitel 5 erwähnt, soll Code mit dem Selection-Objekt möglichst vermieden werden. Listing 6.36 bzw. Listing 6.37 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 in Word 2003 über die Schaltfläche *Weitere der Registerkarte Layout* im Dialogfeld *Grafik formatieren* erreicht wird (Abbildung 6.32) und in Word 2007 über den Eintrag *Weitere Layoutoptionen* der Dropdownmenüs zu den Schaltflächen *Position* und *Textumbruch* auf der Registerkarte *Bildtools/Format*.)

**Listing 6.36** 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:\Beispiele\Soap Bubbles.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:\Beispiele\Zapotec.bmp", LinkToFile:=False, SaveWithDocument:=True)
    shp.WrapFormat.Type = wdWrapSquare
    shp.LockAspectRatio = msoTrue
    shp.Height = 113.4
    shp.RelativeHorizontalPosition = _
        wdRelativeHorizontalPositionColumn
    shp.RelativeVerticalPosition = _
        wdRelativeVerticalPositionParagraph
    shp.Left = CentimetersToPoints(0)
    shp.Top = CentimetersToPoints(0)
End Sub
```

**Listing 6.37** Die C#-Version



```
private void InlineShapeImportierenUndBearbeiten_CS()
{
    wd.Selection sel = wdApp.Selection;
    object objRange = (object) sel.Range;
    wd.InlineShape ils = sel.InlineShapes.AddPicture(@"C:\Beispiele\Soap Bubbles.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)
{
}
```

Listing 6.37 Die C#-Version (Fortsetzung)

```

wd.Selection sel = wdApp.Selection;
object objRange = (object) sel.Range;
wd.Shape shp = doc.Shapes.AddPicture(@"C:\Beispiele\Zapotec.bmp", ref objFalse,
    ref objTrue, 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 Zwischenablage

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 6.38. 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*; ab Word 2007 im Dropdownmenü zur Schaltfläche *Einfügen* auf der Registerkarte *Start*) 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 zuzuweisen. 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 6.38** 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
    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 eindeutigem 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
End Sub
```

Listing 6.38 Prozeduren, um das über die Zwischenablage eingefügte Objekt im Code zu erfassen (Fortsetzung)

```

    End If
Next
'Da AlternativeText bei eingeblendeten nicht druckbaren Zeichen
'über die Grafik angezeigt wird, alle "Autogeneriert" löschen
For Each shp In rng.Paragraphs(1).Range.ShapeRange
    If Left(shp.AlternativeText, 13) = "Autogeneriert" 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

```

### Dialogfeld *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 V des Bonusteils auf der Buch-CD.

Es ist auch möglich, ab Word 2002, das `FileDialog`-Objekt einzusetzen, falls Sie über die Benutzerschnittstelle mehr Kontrolle brauchen (beispielsweise nur bestimmte Dateierweiterungen anbieten wollen). Das `FileDialog`-Objekt wird im Kapitel 15 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 6.36 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 6.10 aufgelistet.

**Tabelle 6.10** Eigenschaften für die Verwaltung von Verknüpfungen

LinkFormat-Eigenschaft oder -Methode	Datentyp	Beschreibung
<code>BreakLink</code>		Löst die Verknüpfung auf
<code>SavePictureWithDocument</code>	Boolean	Wenn <b>False</b> , wird nur die Verknüpfung zur Grafik im Dokument gespeichert (*)
<code>SourceFullName</code> <code>SourceName</code> <code>SourcePath</code>	Zeichenkette	Geben die vollen Pfadangaben, den Dateinamen sowie den Dateipfad zurück
<code>Update</code>		Aktualisiert die Grafik (*)
<i>Gilt nur für <code>InlineShapes</code></i>		
<code>Locked</code>	Boolean	Wenn <b>True</b> , kann die Grafik nicht aktualisiert werden
(*) Fügt ab Word 2002 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 6.31 abgebildet). Neben dem Feldnamen besteht der Feldcode aus der Pfadangabe zur

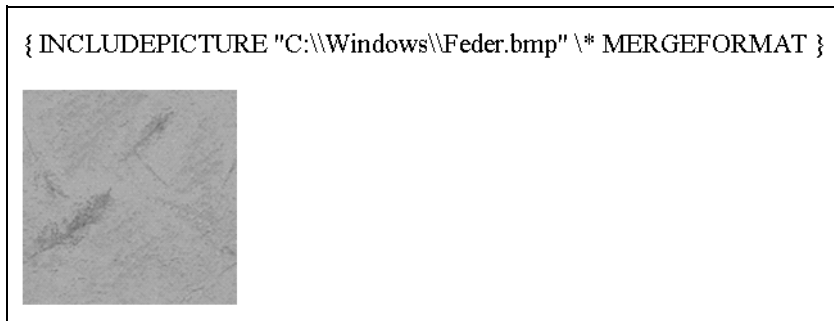


Ursprungsdatei; der Pfad darf relativ zum Speicherort des Dokuments sein oder absolut. Zusätzlich stehen der Feldfunktion zwei wichtige Schalter zur Verfügung:

- \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 diejenigen für InlineShape-Objekte sehen, da Shapes außerhalb des Textes in der Zeichnungsebene stehen.

Abbildg. 6.31 In Wirklichkeit werden in Word verknüpfte Grafiken durch *IncludePicture*-Feldcodes verwaltet



#### 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 Ordnebene 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 ab Word 2002 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 6.39 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 das Objekt wieder hergestellt werden. Am Ende der Prozedur *GrafikAlsFeldEinfuegen* wird die Verknüpfung aufgelöst, so dass nur eine gewöhnliche eingebettete Grafik im Dokument zurückbleibt.

**Listing 6.39** 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
    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*.

#### HINWEIS

Das Dialogfeld für die Verwaltung verknüpfter Objekte befand sich in früheren Word-Versionen in der Menüfolge *Bearbeiten/Verknüpfungen*. In Word 2007 müssen Sie unter der *Office*-Schaltfläche den Befehl *Vorbereiten* und dann (ganz unten im Untermenü) *Verknüpfungen mit Dateien bearbeiten* anwählen.

## 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 Gliederungs- und Normalen- (in Word 2007 Entwurfs-) 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 vor Word 2007 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 und 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 V des Bonusteils auf der Buch-CD.

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 6.40.

**Listing 6.40** Grafik in einen Positionsrahmen mit fester Breite einfügen

```
Private Const strMSGITITEL 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. Er soll aber nur eine Absatzmarke enthalten.
    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, strMSGITITEL
    Case Else
        MsgBox "Unerwarteter Fehler: " & CStr(Err.Number) & vbCr & vbCr & _
            Err.Description, vbOKOnly + vbCritical, strMSGITITEL
End Select
End Sub
```



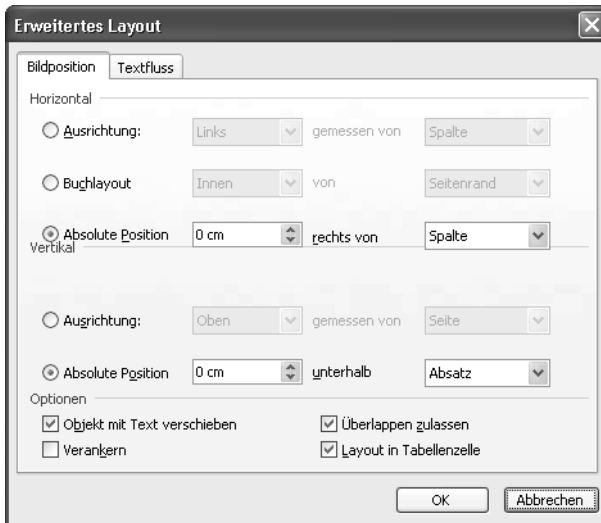
Die Beispieldatei *Bsp06\_03\_Graf.doc* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap06*.

## 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 Word 2007 auf der Registerkarte *Bildtools/Format* der Eintrag *Weitere Layoutoptionen* im Dropdownmenü der Schaltflächen *Position* oder *Textumbruch*), in der Registerkarte *Bildposition* (Abbildung 6.32) 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. 6.32 Die Optionen, die die Positionierung von *Shape*-Objekten regeln



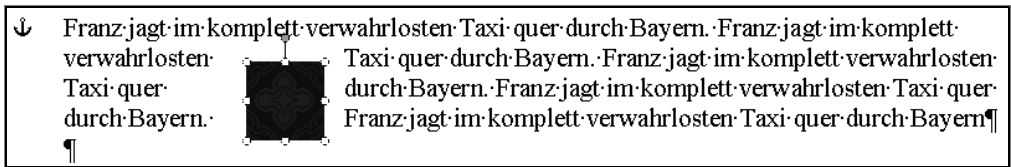
Die erste und wichtigste Regel für die Positionierung ist, dass ein grafisches Objekt (Shape oder Positionsrahmen (Frame)) *immer* mit einem Absatz verankert ist, und sich *immer* auf der gleichen Seite befindet wie dieser Absatz. In Abbildung 6.33 sehen Sie einen Objektanker. Falls das entsprechende Kontrollkästchen über *Extras/Optionen* auf der Registerkarte *Ansicht* aktiviert ist, soll er sichtbar sein, wenn sein grafisches Objekt markiert ist. (In Word 2007 befindet sich diese Option in der Kategorie *Anzeigen* der Word-Optionen.)

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. 6.33** Der Anker eines grafischen Objekts befindet sich immer links des Absatzes, mit dem das Objekt verankert ist



### Verankern = LockAnchor

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. 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 des allgemeinen Dokumentschutzes).

### HINWEIS

Es ist möglich, mit Code ein grafisches Objekt mit einer Seite zu assoziieren, so dass es immer wieder auf die angegebene Seite zurückgesetzt werden kann. Die Lösung finden Sie in Kapitel V im Bonusteil auf der Buch-CD und enthält auch Beispiele für die in diesem Abschnitt vorgestellten Eigenschaften.

### Objekt mit Text verschieben

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 6.11 sowie in Tabelle 6.12 zusammengefasst.

**Tabelle 6.11** 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

Tabelle 6.11 Werte für die Eigenschaft *RelativeVerticalPosition* (Fortsetzung)

Enumeration	Wert	Beschreibung
wdRelativeVerticalPositionPage	1	Der Abstand wird vom oberen Seitenrand gemessen
wdRelativeVerticalPositionParagraph	2	Der Abstand wird relativ zum verankernden Absatz gemessen (mit Text verschieben)

Tabelle 6.12 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 6.13.

Tabelle 6.13 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 ( <i>Gerade/ungerade anders</i> in der Menüfolge <i>Datei/Seiten-einrichten/Layout</i> )
wdShapeLeft	–999998	links
wdShapeOutside	–999993	Die Grafik wird am Außenrand eines Buches ausgerichtet ( <i>Gerade/ungerade anders</i> in der Menüfolge <i>Datei/Seiten-einrichten/Layout</i> )
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 6.41 vorgegangen.

**Listing 6.41** 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.


**Neue Positionierungs- und Vergrößerungsmöglichkeiten in Office 2007**

Zusätzlich zu den oben beschriebenen Positionierungsmöglichkeiten wurden für Office 2007-Grafiken weitere eingeführt. Diese stehen in Word hauptsächlich für AutoFormen zur Verfügung. Der Laufzeitfehler in Abbildung 6.34 erscheint, wenn die neuen Positionierungsbefehle mit einem Shape-Objekt des falschen Typs verwendet werden.

**Abbildg. 6.34** Fehlermeldung, wenn ein *Shape*-Objekt mit den neuen Grafikeigenschaften nicht kompatibel ist



Die Tabelle 6.14 bis Tabelle 6.16 bieten einen Überblick der neuen Enumerationen.



Das Office-Objektmodell wurde entsprechend um die Eigenschaften `LeftRelative`, `WidthRelative`, `TopRelative` sowie `HeightRelative` erweitert, um die Position bzw. Größe des grafischen Objekts festzulegen.

Diese neue Funktionalität unterstützt die Positionierung und Größenänderung relativ zu den Seitenrändern. Angaben werden in Prozent der Breite bzw. Höhe des angegebenen Seitenrands festgelegt. Ein Beispiel für deren Verwendung sehen Sie in Listing 6.42.

**Tabelle 6.14** Zusätzliche Werte für die Eigenschaft *RelativeVerticalPosition* in Office 2007

Enumeration	Wert	Beschreibung
<code>wdRelativeVerticalPositionTopMarginArea</code>	4	Positioniert an den oberen Seitenrand
<code>wdRelativeVerticalPositionBottomMarginArea</code>	5	Positioniert an den unteren Seitenrand
<code>wdRelativeVerticalPositionInnerMarginArea</code>	6	Bei gespiegelten Seitenrändern, positioniert an den inneren Seitenrand
<code>wdRelativeVerticalPositionOuterMarginArea</code>	7	Bei gespiegelten Seitenrändern, positioniert an den äußeren Seitenrand
<code>wdShapePositionRelativeNone</code>	-999999	Prozentuelle Positionierungen werden ignoriert

**Tabelle 6.15** Zusätzliche Werte für die Eigenschaft *RelativeHorizontalPosition* in Office 2007

Enumeration	Wert	Beschreibung
<code>wdRelativeHorizontalPositionLeftMarginArea</code>	4	Positioniert an den linken Seitenrand
<code>wdRelativeHorizontalPositionRightMarginArea</code>	5	Positioniert an den rechten Rand
<code>wdRelativeHorizontalPositionInnerMarginArea</code>	6	Bei gespiegelten Seitenrändern, positioniert an den inneren Seitenrand
<code>wdRelativeHorizontalPositionOuterMarginArea</code>	7	Bei gespiegelten Seitenrändern, positioniert an den äußeren Seitenrand

**Tabelle 6.16** Werte für die Eigenschaften *RelativeHorizontalSize* sowie *RelativeVerticalSize* in Office 2007

Enumeration	Wert	Beschreibung
<code>wdRelativeHorizontalSizeInnerMarginArea</code>	4	Die Breite ist relativ zu der Größe des inneren Rands: bei ungeraden Seiten relativ zu der Größe des linken Seitenrands, bei geraden Seiten relativ zu der Größe des rechten Seitenrands
<code>wdRelativeHorizontalSizeLeftMarginArea</code>	2	Die Breite ist relativ zu der Größe des linken Seitenrands
<code>wdRelativeHorizontalSizeMargin</code>	0	Die Breite ist relativ zu dem Abstand zwischen dem linken und dem rechten Seitenrand

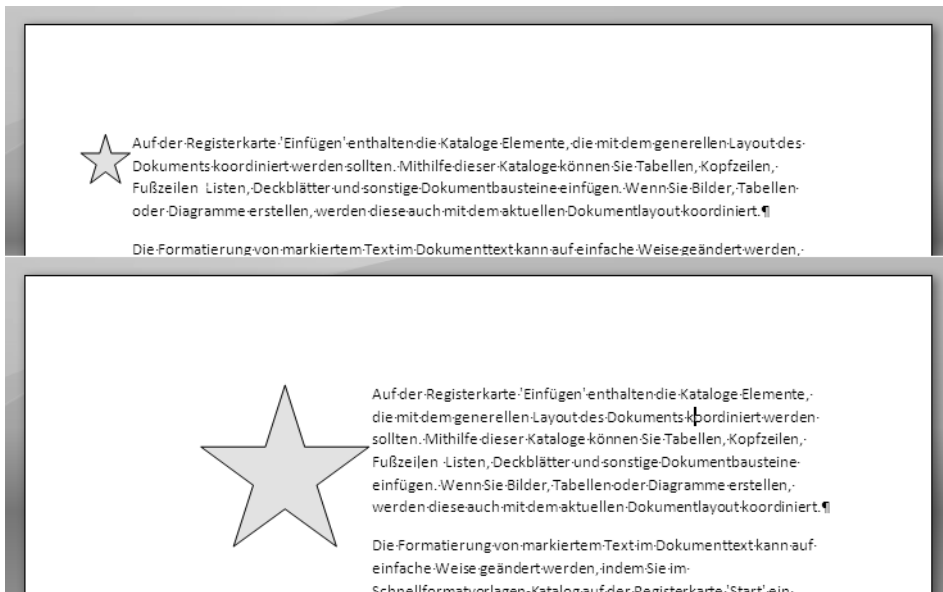
**Tabelle 6.16** Werte für die Eigenschaften *RelativeHorizontalSize* sowie *RelativeVerticalSize* in Office 2007 (Fortsetzung)

Enumeration	Wert	Beschreibung
<code>wdRelativeHorizontalSizeOuterMarginArea</code>	5	Die Breite ist relativ zu der Größe des äußeren Rands: bei ungeraden Seiten relativ zu der Größe des rechten Seitenrands, bei geraden Seiten relativ zu der Größe des linken Seitenrands
<code>wdRelativeHorizontalSizePage</code>	1	Die Breite ist relativ zu der Breite der Seite
<code>wdRelativeVerticalSizeBottomMarginArea</code>	3	Die Höhe ist relativ zu der Größe des unteren Seitenrands
<code>wdRelativeVerticalSizeInnerMarginArea</code>	4	Die Höhe ist relativ zu der Größe des inneren Rands: bei ungeraden Seiten relativ zu der Größe des oberen Seitenrands, bei geraden Seiten relativ zu der Größe des unteren Seitenrands
<code>wdRelativeVerticalSizeMargin</code>	0	Die Höhe ist relativ zu dem Abstand zwischen dem linken und dem rechten Seitenrand
<code>wdRelativeVerticalSizeOuterMarginArea</code>	5	Die Höhe ist relativ zu der Größe des äußeren Rands: bei ungeraden Seiten relativ zu der Größe des unteren Seitenrands, bei geraden Seiten relativ zu der Größe des oberen Seitenrands
<code>wdRelativeVerticalSizePage</code>	1	Die Höhe ist relativ zu der Höhe der Seite
<code>wdRelativeVerticalSizeTopMarginArea</code>	2	Die Höhe ist relativ zu der Größe des oberen Seitenrands
<code>wdShapeSizeRelativeNone</code>	-999999	Prozentuelle Größenänderungen werden ignoriert

Die Prozedur `FormRelativ` arbeitet mit einem Zeichnungsobjekt, das im Dokument vorgängig benannt wurde (dessen Name-Eigenschaft auf »Stern« festgelegt wurde). Die Größe wird auf 50% der Breite des linken Seitenrands festgelegt. In Abbildung 6.35 ist ersichtlich, wie die Änderung der Seitenrandbreite sich auf die Breite des Sterns auswirkt. Zudem wird der linke Rand relativ zum linken Seitenrand auf eine Weite von 50% der Breite des Seitenrands festgelegt.

Die senkrechte Position ist einfach zum oberen Rand; die Höhe ist gleich die absolute Breite des Objekts festgelegt. Dies hat zur Folge, dass sie sich bei einer Änderung des linken Rands nicht anpasst. Um das Ergebnis in Abbildung 6.35 zu erzielen, wurde die Prozedur nochmals durchgeführt.

Abbildg. 6.35 Die relative Positionierung sowie Größenänderung einer AutoForm in Word 2007



Listing 6.42 Eine AutoForm mit den neuen Möglichkeiten in Word 2007 formatieren

```

Sub FormRelativ()
    Dim shp As Word.Shape

    'Die AutoForm wurde vorgängig explizit benannt
    Set shp = ThisDocument.Shapes("Stern")
    shp.LockAspectRatio = msoFalse
    shp.RelativeHorizontalSize = wdRelativeHorizontalSizeLeftMarginArea
    shp.WidthRelative = 50
    shp.Height = shp.Width
    shp.RelativeHorizontalPosition = wdRelativeHorizontalPositionLeftMarginArea
    shp.LeftRelative = 50
    shp.RelativeVerticalPosition = wdRelativeVerticalPositionMargin
    shp.Top = 0
    shp.LockAspectRatio = msoTrue
End Sub

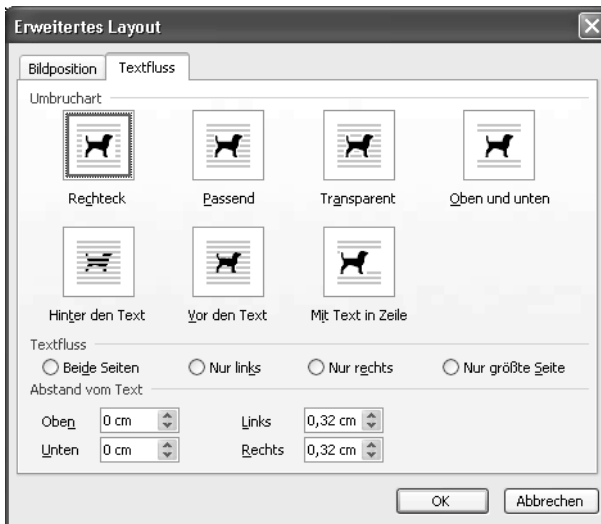
```



Das Word 2007-Beispieldokument *Bsp06\_06\_Graf.docm* finden Sie im Ordner *Beispiele/Kap06* auf der CD-ROM zum Buch.

## Textfluss-Formatierung

Neben der Registerkarte *Bildposition* enthält das Dialogfeld *Erweitertes Layout* die Registerkarte *Textfluss* (Abbildung 6.36), die einige zusätzliche Optionen zur Registerkarte *Layout* im Dialogfeld *Grafik formatieren* anbietet.

**Abbildg. 6.36** 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 Wirkungen sind in Tabelle 6.17 aufgelistet.

**Tabelle 6.17** Die *WrapFormat*-Eigenschaften, die den Textfluss festlegen

Eigenschaft	Enumeration	Wert	Beschreibung oder Dialogfeldoption
Type	<code>wdWrapInline</code>	7	Gilt nur für Office-AutoFormen, die nicht in <code>InlineShape</code> -Objekte umgewandelt werden können. Die AutoForm verhält sich wie ein <code>InlineShape</code> , ist aber keines, was an den weißen Anfassern zu erkennen ist.
	<code>wdWrapNone</code>	3	Stellt das grafische Objekt vor den Text. Wird zurückgegeben, wenn die Grafik vor oder hinter dem Text steht.
	<code>wdWrapSquare</code>	0	Entspricht der Option <i>Rechteck</i>
	<code>wdWrapThrough</code>	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.)
	<code>wdWrapTight</code>	1	Entspricht der Option <i>Passend</i> . Der Textfluss folgt seitlich dem Umriss des Hauptteils des Bildes und ignoriert den Hintergrund.
	<code>wdWrapTopBottom</code>	4	Entspricht der Option <i>Oben und unten</i> . Allgemein ist ein <code>InlineShape</code> , allein stehend in einem Absatz, dieser Einstellung vorzuziehen.

Tabelle 6.17 Die WrapFormat-Eigenschaften, die den Textfluss festlegen (Fortsetzung)

Eigenschaft	Enumeration	Wert	Beschreibung oder Dialogfeldoption
Side	wdWrapBoth	0	Textfluss/Beide Seiten
	wdWrapLargest	3	Textfluss/Nur größte Seite
	wdWrapLeft	1	Textfluss/Nur links
	wdWrapRight	2	Textfluss/Nur rechts
DistanceBottom			Abstand vom Text/Unten
DistanceLeft			Abstand vom Text/Links
DistanceRight			Abstand vom Text/Rechts
DistanceTop			Abstand vom Text/Oben

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 6.37 klar ersichtlich. Von links nach rechts: hinter dem Text, die Textebene und vor dem Text.

Abbildg. 6.37 Die drei grafischen Ebenen eines Word-Dokuments

**HINWEIS**

In der Lösung »Tischkarte mit WordArt« (Kapitel VIII im Bonusteil auf der Buch-CD) finden Sie Beispielcode für die Festlegung und Bestimmung der meisten Eigenschaften des Shape-Objekts, die die Positionierung und Formatierung beeinflussen.



2007

Eine neue Art Umbruch wurde in Word 2007 eingeführt: TextboxTightWrap. Es ermöglicht den Umbruch von Text *um den Text* in einem Textfeld, wie in Abbildung 6.38.

Abbildg. 6.38 Der Text im Dokument fließt um den Text im Textfeld, statt um den (unsichtbaren) Rahmen

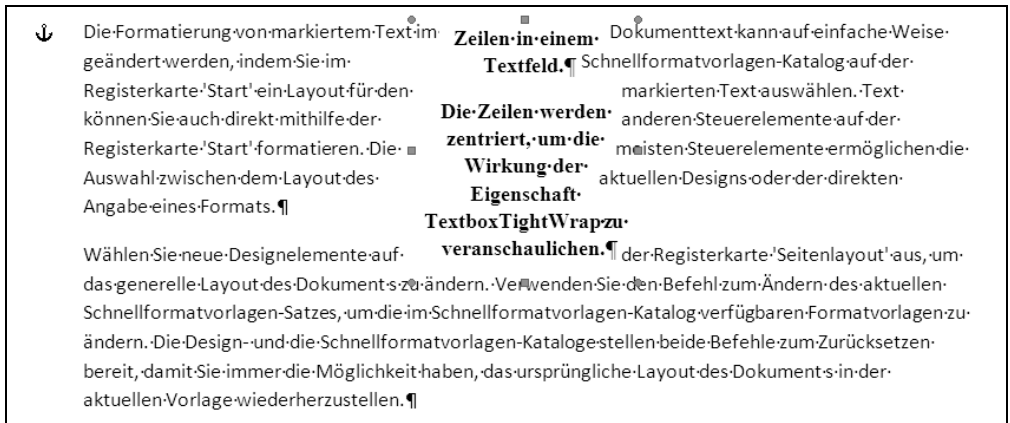


Tabelle 6.18 Die möglichen Werte für die Eigenschaft *TextboxTightWrap*

Enumeration	Wert	Beschreibung
wdTightAll	1	Text umfließt den Inhalt von Textfeldern auf allen Zeilen
wdTightFirstAndLastLines	2	Text umfließt nur die erste und letzte Zeile
wdTightFirstLineOnly	3	Text umfließt nur die erste Zeile
wdTightLastLineOnly	4	Text umfließt nur die letzte Zeile
wdTightNone	0	Text umfließt den Inhalt eines Textfelds nicht

Um dies zu realisieren, muss das Textfeld die folgenden Bedingungen erfüllen:

- Es darf weder einen Rahmen noch eine Schattierung haben.
- Der Textumbruch für das Textfeld muss auf *Passend* festgelegt werden.

Das Listing 6.43 zeigt, wie das Textfeld in Abbildung 6.38 erstellt wurde.

Listing 6.43 Ein Textfeld mit Textumbruch *um den Text* im Textfeld formatieren

```
Sub TextFeldMitUmbruch()
    Dim shp As Word.Shape
    Dim rng As Word.Range

    Set rng = ActiveDocument.Paragraphs(2).Range
    Set shp = ActiveDocument.Shapes.AddTextbox(Orientation:=msoTextOrientationHorizontal, _
        Left:=0, Top:=0, Width:=120, Height:=140, Anchor:=rng)

    shp.RelativeHorizontalPosition = wdRelativeHorizontalPositionColumn
    shp.Left = wdShapeCenter
    shp.RelativeVerticalPosition = wdRelativeVerticalPositionParagraph
    shp.Top = wdShapeCenter
    shp.Fill.Visible = msoFalse
    shp.Line.Visible = msoFalse
    shp.WrapFormat.Type = wdWrapTight
End Sub
```

Listing 6.43 Ein Textfeld mit Textumbruch *um den Text* im Textfeld formatieren (Fortsetzung)

```

With shp.TextFrame
    .TextRange = str1 & vbCrLf & str2
    .TextRange.Bold = True
    .TextRange.Font.Name = "Times New Roman"
    .TextRange.ParagraphFormat.TextboxTightWrap = wdTightAll
    .TextRange.ParagraphFormat.Alignment = wdAlignParagraphCenter
End With
End Sub

```



Die Beispieldatei *Bsp06\_07\_Graf.docm* finden Sie auf der CD-ROM im Ordner *\Beispiele\Kap06*.

**Zeichnen/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. (In Word 2007 befinden sich die Befehle in der Gruppe *Anordnen* der Registerkarte *Zeichentools/Format*.) Im Objektmodell entspricht diesem Befehl die Methode *ZOrder*, deren Werte in Tabelle 6.19 aufgelistet sind.

Tabelle 6.19 Die Werte der Methode *ZOrder*

Enumeration	Menübefehl
<code>msoBringForward</code>	<i>Eine Ebene nach vorne</i>
<code>msoBringInFrontOfText</code>	<i>Vor den Text bringen</i>
<code>msoBringToFront</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 lässt sich Folgendes sagen:

- 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.

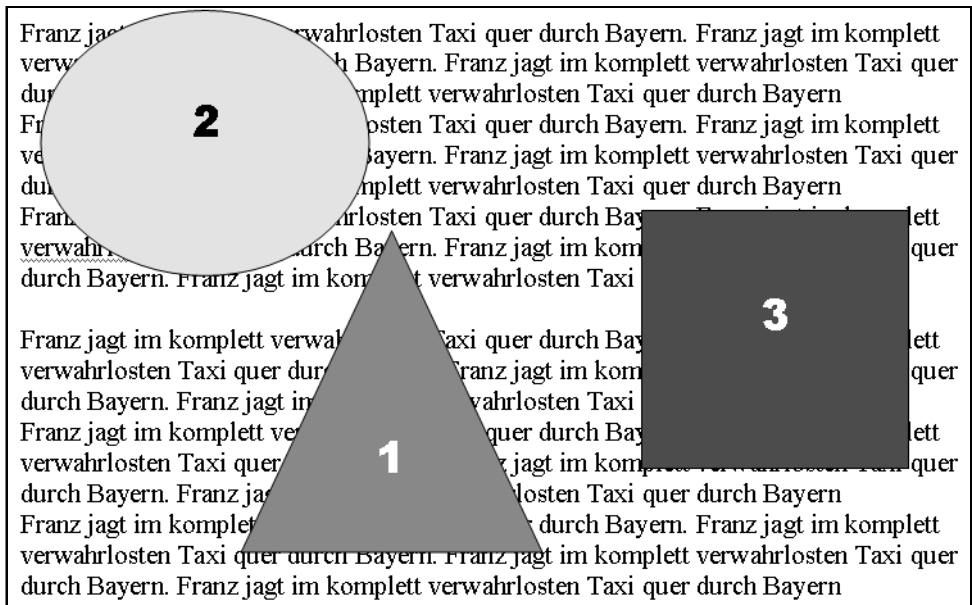
In Abbildung 6.39 sehen Sie drei AutoFormen (alle in der gleichen Ebene vor dem Text stehend), die in der Reihenfolge nummeriert sind, wie sie in das Dokument eingefügt wurden. Die Prozedur *GrafikenAusrichtenUndEinordnen* in Listing 6.44 richtet sie zuerst waagrecht sowie senkrecht zentriert auf der Seite (also über einander) aus. Wie in Abbildung 6.40 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 6.41, 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

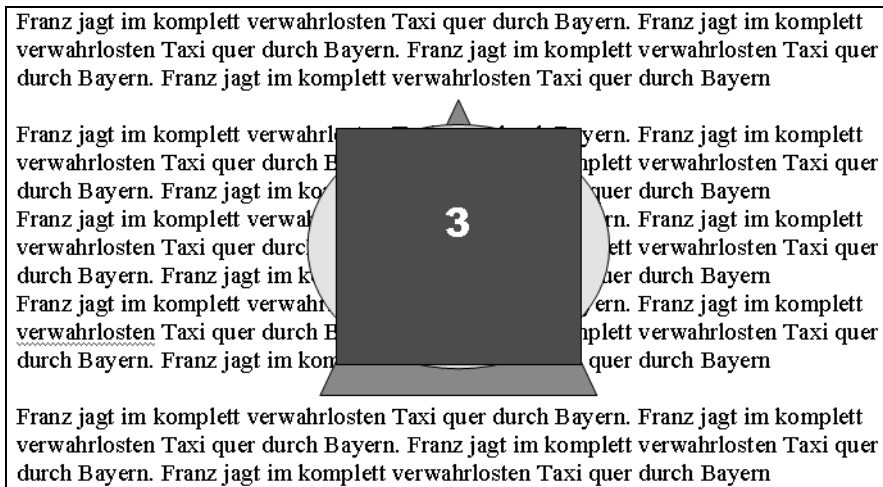
Bitte beachten Sie, wenn Sie mit einer *For Each...Next*-Schleife alle Shape-Objekte in einem Dokument oder Bereich durchlaufen, dass dies in der Reihenfolge ihrer Verankerung geschieht. Mit dieser Reihenfolge hat die *ZOrderPosition* nichts zu tun.

Abbildg. 6.39 Die Grafiken sind in der Reihenfolge nummeriert, in welcher sie eingefügt wurden

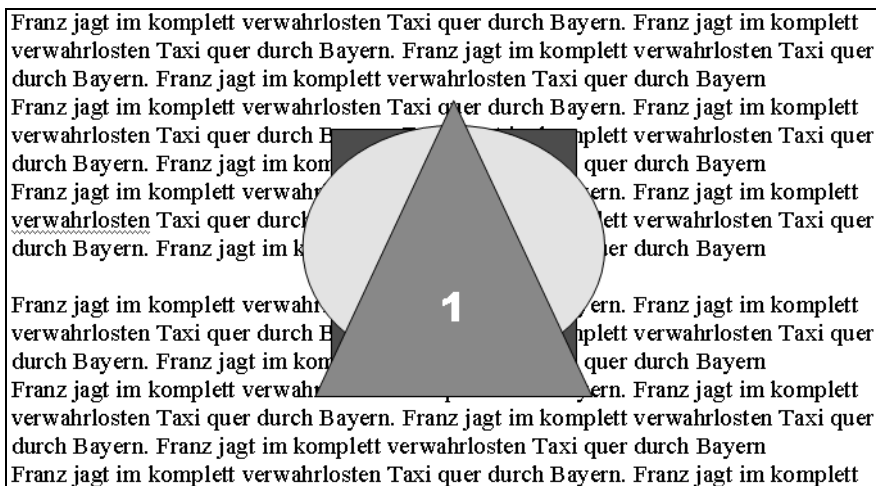




Abbildg. 6.40 Zieht man die Grafiken übereinander, liegt die zuletzt eingefügte zu oberst



Abbildg. 6.41 Die Prozedur in Listing 6.44 hat die ZOrder umgekehrt



Listing 6.44 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 6.44** 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.
' Um diese Wirkung zu sehen, die folgenden Zeilen auskommentieren.
' 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.

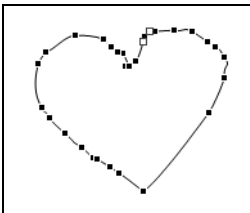
### HINWEIS

Angaben zu WordArt, das ein Objektmodell besitzt, finden Sie in Kapitel 12.

### Freihandformen bearbeiten

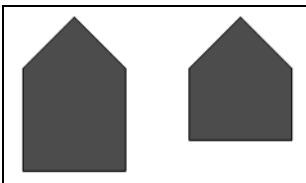
Etwas, das immer wieder für Unklarheit sorgt, ist die Points-Eigenschaft der ShapeNodes-Auflistung. Ein ShapeNode ist entweder ein Eckpunkt oder bei Kurven ein Punkt, der die Rundung bestimmt. Durch Änderung der Koordinaten (Points) wird das Aussehen einer *von Hand gezeichneten* AutoForm geändert. In der Benutzerschnittstelle werden sie über den Menüpunkt *Punkte bearbeiten* der *Zeichnen*-Symbolleiste in der gleichnamigen Symbolleiste aktiviert und sehen wie in Abbildung 6.42 aus. (In Word 2007 befindet sich der Befehl im Dropdownmenü zur Schaltfläche *Form bearbeiten* der Gruppe *Formen einfügen* in der Registerkarte *Zeichentools/Format*.)

**Abbildg. 6.42** 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 6.45 veranschaulicht das Problem und die Lösung. Das Makro erstellt ein Fünfeck als Vieleck (Polygon), dann werden die zwei senkrechten Seiten verkürzt, wie in Abbildung 6.43 ersichtlich.

**Abbildg. 6.43** Ein Fünfeck mit VBA erstellen und bearbeiten



Zuerst wird eine Objekt-Variable – `shp` – für die `AutoForm` als `Shape` deklariert. In der Datenfeld-Variable `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 ist.

`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 waagrechte Einstellung gleich, die senkrechte wird um 15 Punkte (typographisches Maß) verkürzt (höher gestellt).

**Listing 6.45** 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
```

Listing 6.45 Ein Polygon erstellen und anschließend die Eckpunkte ändern (Fortsetzung)

```

aEckPunkte(6, 2) = 25
Set shp = ActiveDocument.Shapes.AddPolyline(aEckPunkte)
shp.RelativeHorizontalPosition = wdRelativeHorizontalPositionMargin
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 jedes 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 6.46 finden Sie Beispielcode, der alle Textfelder (und ausschließlich diese) mit Textinhalt im Dokumenttext gleich formatiert.

Listing 6.46 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"
                End If
            End With
        End If
    Next shp
End Sub

```

**Listing 6.46**    Den Text in allen Textfeldern des Dokumentkörpers mit Arial 10, fett, formatieren (*Fortsetzung*)

```
        .TextRange.Font.Size = 10
        .TextRange.Bold = True
    End If
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*.

---

## Zusammenfassung

In diesem Kapitel wurde ein Überblick über die professionelle Arbeit mit Word-Dokumenten und die entsprechenden Teile des Objektmodells vermittelt. Im Brennpunkt lagen:

- Das Wissen zum Einrichten eines komplexen Dokuments basiert auf dem Section- (Seite 242 ff.), StoryRange- (Seite 244 ff.), PageSetup- (Seite 248 ff.) und HeaderFooter-Objekt (Seite 256 ff.).
- Vertiefte Informationen zu den Themen »Lange Dokumente« (Seite 262 ff.) und »Bausteine« (Seite 267 ff.).
- In den weiteren Abschnitten standen Formatvorlagen (Seite 242) die Nummerierungen (Seite 298 ff.) und die Grafik-Objekte (Seite 309 ff.) im Vordergrund des Interesses.

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

# Word und Datenstrukturen

### In diesem Kapitel:

Zielscheibe Textmarke: Das <i>Bookmark</i> -Objekt	340
Inhalt mit Tabellen strukturiert darstellen	349
Feldfunktionen	378
Formulare: Das <i>FormField</i> -Objekt	391
Die Alternative zu Formularfeldern: <i>ContentControls</i> -Objekt	399
Der Seriendruck: Das <i>MailMerge</i> -Objekt	427
Zusammenfassung	446


Von Word wird seit jeher verlangt, dass es ein breites Spektrum von Diensten erfüllt. Unter anderem möchten Benutzer und Entwickler es für das Sammeln und Ausgabe von Daten einsetzen. Dieses Kapitel widmet sich den Word-eigenen Werkzeugen, dieser Aufgabe gerecht zu werden.

## Zielscheibe Textmarke: Das *Bookmark*-Objekt

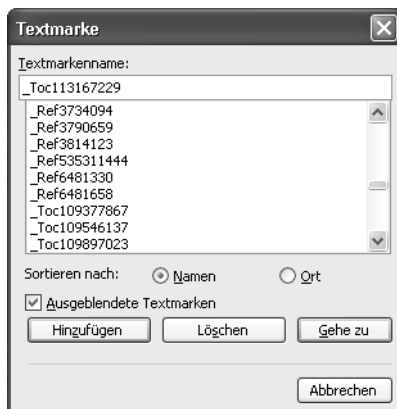
Textmarken in Word-Dokumenten erfüllen zwei wichtige Aufgaben. Dem Anwender dienen sie hauptsächlich als Quellenbezeichner für Verweise. Der Entwickler benutzt sie, um Zielbereiche zu bezeichnen.

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« weiter hinten in diesem Kapitel). Solche Textmarken bleiben unter normalen Umständen dem Benutzer verborgen, da ihre Namen mit einem Unterstrich beginnen. Wenn Sie jedoch das Kontrollkästchen *Ausgeblendete Textmarken* im Dialogfeld *Textmarke* ein-, aus- und nochmals einschalten (vor Word 2007 erreichen Sie es über die Befehlsfolge *Einfügen/Textmarke*; in Word 2007 befindet sich die Schaltfläche in der Gruppe *Hyperlinks* der Registerkarte *Einfügen*), erscheinen diese in der Liste wie in Abbildung 7.1. 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.

Abbildg. 7.1 In diesem Dialogfeld werden die Textmarken verwaltet





Das Dialogfeld in Abbildung 7.1 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 oder Unterstrich 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, das geht jedoch über die Programmierschnittstelle.

**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 7.2 ersichtlich. Die erste markiert eine bestimmte 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. Vor Word 2003 befindet sich diese in der Menüfolge *Extras/Optionen/Ansicht*; in Word 2007 in der Kategorie *Erweitert* der *Word-Optionen* im Abschnitt *Dokumentinhalt anzeigen*.

Abbildg. 7.2 Die zwei Arten von Textmarken

Falls die entsprechende Option in *Extras/Optionen/Ansicht* aktiviert ist, sehen Sie anfangs dieses Absatzes eine Textmarke, die eine Position markiert. ¶  
In diesem Absatz ist das dritte Wort mit einer Textmarke versehen. ¶

### Textmarken 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 7.1. 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 7.1** 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 & vbCrLf & 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 Beispielpjekt auf der Buch-CD.

Listing 7.2 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 sind
    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="";
}

```



Die Beispieldatei *Bsp07\_01\_Bookmark.doc* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap07*.

### Daten schreiben

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: `Selection.GoTo What:=wdGoToBookmark, Name:="Test"`. 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, erfolgt eine Meldung, dass die Textmarke nicht gefunden werden kann.

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 7.3).

Umfasst die Textmarke Text, wie die Textmarke »TM3« in Abbildung 7.3, ersetzen die Daten diesen. Zudem wird dadurch die Textmarke gelöscht.

Eine Alternativmethode besteht darin, den neuen Text vor dem Textmarkenbereich einzufügen mit der Methode `Range.InsertBefore`. 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 7.3.

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 7.3. (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 7.3 bzw. Listing 7.4 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. 7.3** Daten werden den drei Textmarken mit verschiedenen Methoden hinzugefügt. In jedem Fall bleibt die Textmarke bestehen.

<p>TM1 mit Inhalt TM2 als Position TM3 mit Inhalt</p> <p>NEUER TEXT TM1 mit Inhalt NEUER TEXT TM2 als Position NEUER TEXT</p>
---

Listing 7.3 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 7.4 Die C#-Version



```

private void DatenEingeben_CS()
{
    object objMissing = System.Reflection.Missing.Value;
    string text = "NEUER TEXT";
    string dateiName = System.IO.Directory.GetCurrentDirectory();
    System.IO.DirectoryInfo di = new System.IO.DirectoryInfo(dateiName);
    object objDateiName = (object)
        (di.Parent.Parent.Parent.Parent.FullName + "\\Bsp07_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, 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;
}

```

**Listing 7.4** Die C#-Version (Fortsetzung)

```

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 *Bsp07\_01\_Bookmark.doc* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap07*.

**Daten aus Textmarken 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 11 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 7.1. Die meisten davon haben gleichwertige VBA-Anweisungen ersetzt, interessant sind vor allem `\Page`, `\Line` sowie `\HeadingLevel`.

**Tabelle 7.1** 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>
<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>

**Tabelle 7.1** Liste der vordefinierten Textmarken, mit gleichwertiger Anweisung des Objektmodells, wo vorhanden (Fortsetzung)

Vordefinierte Textmarke	Beschreibung	Objektmodell-Anweisung
<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 7.5 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 7.5** 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
```



## Listing 7.5

Da mit vordefinierten Textmarken gearbeitet wird, wird ausnahmsweise mit dem *Selection*- statt dem *Range*-Objekt gesucht (Fortsetzung)

```
.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 *Bsp07\_01\_Bookmark.doc* finden Sie auf der CD-ROM zum Buch im Ordner \Beispiele\Kap07.

## 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 Datenaufstellungen 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, versionsspezifischen 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.** 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 in 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.

**ConvertTextToTable.** 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 7.6 bzw. dem Listing 7.7 entnommen werden; das Resultat ist in Abbildung 7.4 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. 7.4** 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 7.6** 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
```

**Listing 7.6** Drei mögliche Methoden, um mit Daten gefüllte Tabellen zu erstellen. Die dritte ist mit Abstand die schnellste, wenn die Tabellen lang werden. *(Fortsetzung)*

```

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)
        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

```

**Listing 7.6** Drei mögliche Methoden, um mit Daten gefüllte Tabellen zu erstellen. Die dritte ist mit Abstand die schnellste, wenn die Tabellen lang werden. (*Fortsetzung*)

```
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 7.7 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 7.7** Die C#-Version

```
private void Listing7_7_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)
{
    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;
```

**Listing 7.7** Die C#-Version (Fortsetzung)

```

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);
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))
        {

```

Listing 7.7 Die C#-Version (Fortsetzung)

```

        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,
    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 (in Word 2003 oder 2007) XML-Datei zu erstellen und in Word zu importieren bzw. einzufügen. 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-Datei-format finden Sie in Teil V dieses Buches. Weitere Informationen zu allen Dateiformaten befinden sich auf der MSDN-Webseite bei Microsoft.com.



Die Beispieldatei *Bsp07\_01\_Table.doc* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap07*.

## Tabellen formatieren

Die Tabellenformatierung umfasst zwei Aspekte: das Aussehen und das grafische Layout. Unter Aussehen verstehen wir sowohl Schrift- und Absatzigenschaften 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:

#### Schriftformatierung

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;
```

#### Absatzformatierung

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 Cells-Auflistung wie ein Array:

```
row.Cells[row.Cells.Count].Range.ParagraphFormat.FirstLineIndent =  
wdApp.CentimetersToPoints(0.3f);
```



## Rahmenformatierung

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
```

## Schattierung

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
```



2007

### ACHTUNG

Eigentlich enthält jede Tabellenzelle zwei »verborgene« Zeichen, wie im Abschnitt »Informationen aus Tabellen holen« beschrieben. In Word 2007 ist das letzte anscheinend nicht mehr gleich »sichtbar«, wie bisher. Somit müssen alle Prozeduren, die auf diesem Prinzip aufbauen, für Word 2007 geprüft und wenn nötig angepasst werden. Für das obige Beispiel:

```
Set rng = cel.Range.Words(cel.Range.Words.Count - 1)
```

### Formatvorlage

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.

### Tabellenformatierung

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* (In Word 2007 erreichen Sie das Dialogfeld über *Tabellentools/Layout* in der Gruppe *Tabelle*). Die Abbildung 7.5 veranschaulicht die Wirkung der vorgestellten Anweisungen.

Abbildg. 7.5 Auswirkung der Befehle, die den Textumbruch und seine Position in der Tabellenzelle festlegen

Spalte 1	Spalte 2
Zeile 2	
	Franz jagt im komplett verwahrlosten Taxi quer durch Bayern. Franz jagt im
	komplett verwahrlosten
Mehr Text als Platz in der Zelle ist. Noch eine Zeile, die umbricht.	

Spalte 1	Spalte 2
Zeile 2	
	Franz jagt im komplett verwahrlosten Taxi quer durch Bayern. Franz jagt im komplett verwahrlosten
Spalte 1	Spalte 2
Zeile 2	
Mehr Text als Platz in der Zelle ist. Noch eine Zeile, die umb	

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

### Tabelleneinzug / Einzug von links

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. (In Word 2007 befindet sich die Schaltfläche in *Tabellentools/Layout*, in der Gruppe *Tabelle*) Das Gegenstück im Word-Objektmodell ist:

```
ActiveWindow.View.TableGridlines = False '( bzw. True)
```

### Zellenwechsel

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 Beispieldcode:

```
tbl.Rows.AllowBreakAcrossPages = False
```



Auch diese Eigenschaft verlangt in C# eine Ganzzahl statt eines booleschen Werts.

```
tbl.Rows.AllowBreakAcrossPages = 0;
```

### Kopfzeilen wiederholen

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* bzw. in Word 2007 in der Gruppe *Daten* der Registerkarte *Tabellentools/Layout* mit dem Befehl *Überschriften 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.

**Senkrechte Ausrichtung**

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* (in Word 2007 befindet sich der Befehl *Textrichtung* in der Gruppe *Ausrichtung* der Registerkarte *Tabellentools/Layout*). 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.

**Zellenbegrenzung**

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)
```

## Zellenumbruch

### 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;
```



Die Beispieldatei *Bsp07\_02\_Table.doc*, woraus die obigen Codeschnipsel stammen, finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap07*.

## Tabellenformatvorlage

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.

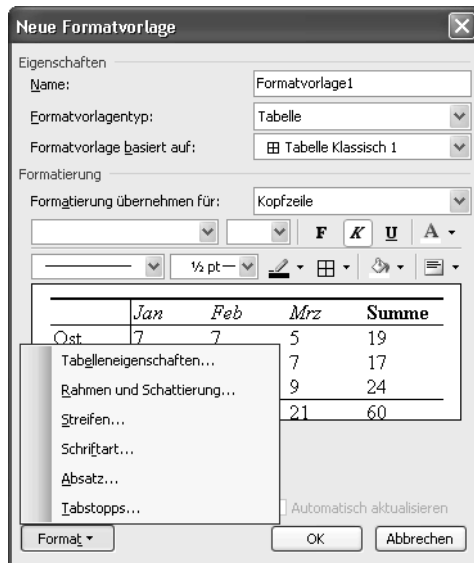
In der Benutzerschnittstelle wird eine Tabellenformatvorlage wie folgt erstellt:

1. Blenden Sie den Aufgabenbereich *Formatvorlagen und Formatierung* ein (in Word 2007 heißt er *Formatvorlagen*). Klicken Sie auf die Schaltfläche *Neue Formatvorlage*. Das Dialogfeld in Abbildung 7.6 wird eingeblendet.

2. Geben Sie im obersten Feld des Dialogfelds einen Namen ein.
  3. Legen Sie als *Formatvorlagentyp* »Tabelle« fest.
  4. 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*.
  5. 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-Konstantenwerte der Condition-Methode. Sie werden beim Erstellen einer Tabellenformatvorlage gebraucht.)
6. Legen Sie die gewünschte Formatierung fest. Zur Verfügung stehen alle Einträge hinter der Schaltfläche *Format* (Abbildung 7.6): *Tabelleneigenschaften*, *Rahmen und Schattierungen*, *Streifen*, *Schriftart*, *Absatz* und *Tabstopps*.

Abbildg. 7.6

Wählen Sie den Formatvorlagentyp *Tabelle*, um eine Tabellenformatvorlage zu definieren



#### HINWEIS

Ein Codebeispiel für das Erstellen einer Tabellenformatvorlage finden Sie im Bonuskapitel I »Kalenderblatt« auf der CD-ROM zum Buch.

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/Verbund

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* (in Word 2007 *Verbund*) 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 7.7 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. 7.7

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 Eckzellenfor-

matierungen 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 Zellausrichtung 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 Dropdownfeld *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.

### 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 in Word 2003 im Aufgabenbereich *Formatvorlagen und Formatierung* rechts auf den entsprechenden Eintrag und wählen Sie *Als Standard-Tabellenformatvorlage festlegen*. In Word 2007 klicken Sie in der Gruppe *Tabellenformatvorlagen* der Registerkarte *Tabellentools/Entwurf* mit der rechten Maustaste auf die Formatvorlage und wählen im Kontextmenü den Eintrag *Als Standard festlegen* aus.

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 weder AutoFormat oder Tabellenformatvorlage für die neuen Tabellen gewählt haben, 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- oder Baustein-Eintrag zu speichern. Wir machen aber darauf aufmerksam, dass beim Einfügen eines solchen 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 *Positionierung* zu erreichen ist, sobald der *Textumbruch* auf *Umgebend* festgelegt wurde. (Diese Einstellungen sind jenen für Grafiken sehr ähnlich. Mehr Informationen finden Sie im Abschnitt »Grafiken: Die InlineShape- und Shape-Objekte« in Kapitel 6.)

Abbildg. 7.8 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 *RelativeVertical*-

Position, festgelegt. Die beiden letzten bestimmen, ob die Position relativ zu Zeichen, Zeile, Absatz, Spalte, Rand oder Seite sein soll, und sind in den Tabellen des Abschnitts »Grafiken: Die Inline-Shape- und Shape-Objekte« im Kapitel 6 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 7.2** *WdTablePosition*-Konstantwerte

<b>WdTablePosition-Enum</b>	<b>Wert</b>	<b>Beschreibung</b>
<code>wdTableBottom</code>	-999997	Die Tabelle wird bündig zum unteren Text- oder Seitenrand positioniert
<code>wdTableCenter</code>	-999995	Die Tabelle wird senkrecht oder waagerecht 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
<code>wdTableTop</code>	-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* (in Word 2007 in der Kategorie *Erweitert* der *Word-Optionen* im Abschnitt *Kompatibilitätsoptionen für*) 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 Kompatibilitätsoptionen finden Sie bei <http://support.microsoft.com/kb/288792/de>. Dieser Artikel umfasst Word 2000 bis 2003. Ein Artikel für 2007 war zum Zeitpunkt, als dieses Buch geschrieben wurde, nicht verfügbar.

**Spaltenbreite**

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
```

**Zeilenhöhe**

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 der Registerkarte *Zeile* der Tabelleneigenschaften. 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 7.8 veranschaulicht. Bitte beachten Sie, dass das Festlegen einer Zeilenhöhe für eine Zeile mit automatischer Zeilenhöhe die `HeightRule` auf `wdRowHeightAtLeast` festlegt.

**Listing 7.8** Die Auswirkungen der *Row*-Eigenschaften *Height* und *HeightRule*

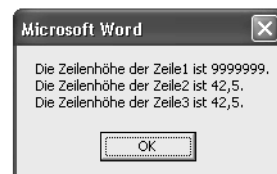
```
Sub TabellenZeilenHöhen()
    Dim strZeilenhöhen As String
    Dim tbl As Word.Table
    Dim row As Word.Row

    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 7.9. 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. 7.9** 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 7.9 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 7.9** 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(2)
    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.InsideLineWidth _
        + celEnde.Borders.OutsideLineWidth + celEnde.Borders.InsideLineWidth
    MsgBox PointsToCentimeters(sngHöhe) & " Zentimeter"
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 werden. Das Objektmodell bietet die Methode *Cell.Split* an. Die folgende Codezeile teilt die erste Zelle der dritten Zeile in zwei untereinander stehende Zellen.

```
tbl.Cell(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 *Bsp07\_03\_Table.doc* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap07*.

## 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 4 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 Speicherung 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 Tabellenzelle 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 7.10 sorgt.

**Listing 7.10** 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
```

Listing 7.10 Daten aus einer Tabelle Zelle für Zelle auslesen und in ein Array aufnehmen (Fortsetzung)

```

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

'Gibt das letzte Element des Arrays aus
Debug.Print aDaten(lAnzZeilen, lAnzSpalten)
End Sub

'Am Ende jedes Zellenbereichs steht eine Absatzmarke (Chr(13))
'sowie "Ende-der-Zelle"-Zeichen (Chr(7))
'Verhält sich gleich in Word 2007
Function TrimZellenInhalt(str As String)
    str = Left(str, Len(str) - 2)
    TrimZellenInhalt = str
End Function

```



Die Beispieldatei *Bsp07\_04\_Table.doc* finden Sie auf der CD-ROM im Ordner *\Beispiele\Kap07*.

**ConvertToText.** 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 7.11 veranschaulicht. Hierzu wird die ConvertToText-Methode (entspricht dem Menübefehl *Tabelle/Umwandeln/Tabelle in Text* bzw. in Word 2007 der Schaltfläche *In Text konvertieren* in der Gruppe *Daten* der Registerkarte *Tabellentools/Layout*) 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 7.10 – das Resultat von Listing 7.11 – trennt ein Tabulatorzeichen die Zellinhalte. Ein »|«-Zeichen wurde durch den Code anstelle einer Absatzmarke innerhalb einer Tabellenzelle eingefügt, während die drei Leerzeichen ein Tabulatorzeichen ersetzt haben. Diese werden später wieder zurückverwandelt.

**Abbildg. 7.10** 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 7.11** 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
    strPara = vbCr
    strTabErsatz = " "
    strParaErsatz = "|"

    Set tbl = ActiveDocument.Tables(1)
    'Die Trennzeichen für die umwandelte 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()

```



**Listing 7.11** Eine Tabelle in eine zeichengetrennte Zeichenkette umwandeln und diese in ein Array schreiben (*Fortsetzung*)

```
'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)
        rng.Find.Execute FindText:=aTrennzeichen(lZaehler, 0),
            replacewith:=aTrennzeichen(lZaehler, 1), Replace:=wdReplaceAll
    Next lZaehler
    rng.Find.Execute
End Sub
```



Die Beispieldatei *Bsp07\_04\_Table.doc* finden Sie auf der CD-ROM im Ordner *\Beispiele\Kap07*.



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 und 2007 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 7.12 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 7.12 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 7.11 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 7.12** 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

    '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"
```

Listing 7.12 Verschachtelte Tabellen erstellen (Fortsetzung)

```

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. 7.11 Das Ergebnis von Listing 7.12

	Spalten- Überschrift			
		Hallo		



Die Beispieldatei *Bsp07\_05\_Table.doc* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap07*.

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 7.13. 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 7.11 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 7.13** 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

```

### Verbundene 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 aufweist.«.

Die beste Methode, um die 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 7.12 ersichtlich, nicht nur die »Rowspan«- und »Colspan«-Informationen, sondern auch Informationen über die Zellenausrichtung und Spaltenbreite liegen bereit.

**Abbildg. 7.12** Ausschnitt aus einer als gefiltertes HTML gespeicherte 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 oder 2007 vorliegt, ist die XML-Eigenschaft den Tabellenbereich abzufragen und deren Resultat zu analysieren (mehr über das XML-Format von Word erfahren Sie in Kapitel 22).

Muss es wirklich mit den Bordmitteln des Objektmodells gehen, bereiten Sie sich auf etwas Kopfzerbrechen vor. Das Beispieldokument *Bsp07\_04\_Table.doc* enthält eine Prozedur *VerbundeneTabellenZellen*, 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 7.13 sehen Sie eine Beispieltabelle sowie das Resultat der Prozedur.

**Abbildg. 7.13** 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.



Die Beispieldatei *Bsp07\_04\_Table.doc* finden Sie auf der CD-ROM zum Buch im Ordner `\Beispiele\Kap07`.

#### HINWEIS

Mit Tabellen aus anderen Anwendungen wie Excel oder Access befassen sich der Abschnitt »Feldfunktionen« in diesem Kapitel sowie die Kapitel 11 und 12.

# 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 Word 2007 befindet sich der Befehl *Feld* im Dropdownmenü zur Schaltfläche *Schnellbausteine* in der Gruppe *Text* der Registerkarte *Einfügen*. Leider wurde die Hilfe zu den einzelnen Feldfunktionen stark dezimiert. Es sind lediglich Angaben für diejenigen Funktionen vorhanden, die in Verbindung mit dem Seriendruck eingesetzt werden. Wir können nur hoffen, dass diese Informationen mit der Zeit nachgeliefert werden. Aber um sicherzugehen, lohnt es sich, eine Sicherheitskopie der Hilfedatei (\*.chm) für eine ältere Version von Word zu erstellen, um die benötigten Informationen nachschlagen zu können.

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.** Word aktualisiert Feldfunktionen nicht laufend; das würde zu viele Ressourcen benötigen und Word würde extrem langsamer 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
'für ein Inhaltsverzeichnis
doc.TablesOfContents(1).Update
```

## HINWEIS

Welche Auslöser welche Feldfunktionen aktualisieren, ist nicht gut dokumentiert. Die einzige uns bekannte Quelle ist der englische Knowledge Base-Artikel »Which fields are updated when you open, repaginate, or print document«, zu finden unter <http://support.microsoft.com/kb/211629/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 7.3 aufgelistet sind. Ein Beispiel für ihren Einsatz befindet sich in Listing 7.15.

Tabelle 7.3 Die `WdFieldKind`-Konstantwerte

<code>WdFieldKind-Enum</code>	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. INCLUDEPICTURE oder FORMDROPDOWN)
<code>wdFieldKindNone</code>	0	Ein ungültiges Feld (z. B. zwei Feldzeichen ohne Inhalt)
<code>wdFieldKindWarm</code>	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).

**Lock, Unlink.** 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 umwandelt werden.

```
doc.Fields.Locked = True
doc.Sections(1).Headers(wdHeaderFooterPrimary).Range.Fields.Unlink
```

Mehr über Abschnitte sowie Kopf- und Fußzeilen erfahren Sie im Abschnitt in Kapitel 6.

#### HINWEIS

Wie in der Diskussion »Das ganze Dokument mit VBA durchsuchen« des Abschnitts »Die Nadel im Heuhaufen: *Find/Replace* einsetzen« in Kapitel 5 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 einen 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 7.14 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 (in Word 2007 befindet sich der Befehl *Eintrag festlegen* in der Gruppe *Index* der Registerkarte *Verweise*). 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. 7.14** Eine Feldfunktion, um einen Index zu generieren. Die graue Schattierung wird über *Extras/Optionen/Ansicht* geregelt.

```
{ INDEX \e " " \c "2" \z "1031" }
```

**Add.** 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 Zeichen- und Tabellenformatierungen, 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 in Kapitel 6), um die Grafikgröße festzuhalten, und in Feldern, deren Ergebnis eine Tabelle ist (*Link* und *Database*).

Das *Index*-Feld in Abbildung 7.14 kann auf eine der zwei beschriebenen Arten, mit oder ohne Angabe des `Type`-Arguments, eingefügt werden:



```

Const strQuote as String = "" 'Anführungszeichen
Set rng = Selection.Range
'Mit Angabe des Typs
Set fld = rng.Fields.Add(Range:=rng, Type:=wdFieldType, 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.wdFieldTypeIndex;
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 vor der Version 2007 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.

Im Abschnitt über Grafiken in Kapitel 6 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 7.14** 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:\\Beispiele\\"
    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 *Bsp07\_01\_Field.doc* finden Sie auf der CD-ROM zum Buch im Ordner `\Beispiele\Kap07`.

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 7.15 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 7.15** 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
End If
```

**Listing 7.15** Ermitteln, ob sich die Markierung in einer Feldfunktion befindet (Fortsetzung)

```
'Liegt die Markierung am Feldanfang, wird der Startpunkt des Feldcodebereichs
'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 *Bsp07\_01\_Field.doc* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap07*.

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.

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 »0« (Zero) festgelegt werden. Bitte beachten Sie, dass sich diese Option auf alle Dokumentfenster der laufenden Word-Sitzung auswirkt.



Die Prozeduren *ButtonTestAusführen* und *ButtonTest*, die die beschriebene Funktionalität veranschaulichen, finden Sie in der Beispieldatei *Bsp07\_01\_Field.doc* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap07*.

## 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 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.

Um die benötigte Syntax zu ermitteln, fügen Sie die Datei, Grafik oder Tabelle über die Benutzerschnittstelle mit Verknüpfung ein. Stellen Sie sicher, dass das Objekt ohne Textflussformatierung formatiert ist. 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 umwandelt werden, so dass diese über mehrere Seiten umbrochen wird (siehe Abbildung 7.15).

**HINWEIS** Eingebettete Excel-Tabellenobjekte werden in Kapitel 12 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 7.15 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 2007 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$(34) & _
"C:\\WordBuch\\Beispiele\\Kap07\\Bsp07_02 Field.xls" & Chr$(34) & " " & Chr$(34) & _
"Tabelle1!wdFieldType" & Chr$(34) & " \a \f 5 \h \* MERGEFORMAT"
```

**HINWEIS** *Link*-Feldfunktionen unterstützen keine relativen Pfadangaben.

**Abbildg. 7.15** 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	Advance	Versetzen
wdFieldExpression	=(Ausdruck)	=(Formula)	Ask	Frage
wdFieldSection	Abschnitt	Section	Author	Autoren
wdFieldDate	AktualDate	Date	AutoNum	AutoNr
wdFieldQuote	Angebot	Quote	AutoNumLg	AutoNrDez
wdFieldNumPages	AnzSeiten	NumPages	AutoNumOut	AutoNrGlo
wdFieldNumWords	AnzWörter	NumWords	AutoText	AutoText
wdFieldNumChars	AnzZeichen	NumChars	AutoTextList	AutoTextListe
wdFieldAutoNum	AutoNr	AutoNum	BarCode	(nur gültig für US-Version)

WdFieldType	Deutscher-Ausdruck	Englischer-Ausdr.	Englischer-Ausdr.	Deutscher-Ausdruck
wdFieldAutoNumLegal	AutoNrDez	AutoNumLg	Comments	Kommentar
wdFieldAutoNumOutline	AutoNrGlo	AutoNumOut	Compare	Vergleich
wdFieldAuthor	Autoren	Author	CreateDate	ErstellDate
wdFieldAutoText	AutoText	AutoText	Databases	Datenbank
wdFieldAutoTextList	AutoTextListe	AutoTextList	Date	AktualDate
wdFieldUserAddress	BenutzerAdre	UserAddress	DocProperty	DokEigenschaft

LINK.Excel.Sheet.8-C:\\WordBuch\\Beispiele\\Kap07\\Bsp07\_02\_Field.xls  
Tabelle1!wdFieldType\\a-f 5-h-\*\\\*MERGEFORMAT



Die Beispieldateien *Bsp07\_02\_Field.doc* und *Bsp07\_02\_Field.xls* finden Sie auf der CD-ROM zum Buch im Ordner *\\Beispiele\\Kap07*.

### 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:\\Beispiele\\FeatherTexture.bmp", PreserveFormatting:=True)
fld.Unlink
```

## Berechnungen

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 aller Versionen vor Word 2007, wenn Sie nach dem Begriff »= (Formula)-Feld« (ohne Anführungszeichen) suchen. Wir befassen uns hier mit zwei Aspekten.

### Auf Tabelleninhalt 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: { = (Tabelle1 C1) }. Um den Wert einer einzelnen Tabellenzelle abzufragen, kann die Funktion *Sum* verwendet werden: { = Sum(Tabelle2 C1) }.



Die Beispieldatei *Bsp07\_01\_Field.doc* befindet sich auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap07*. Leser ohne Zugriff auf eine ältere Hilfedatei können dieser die Grundsätze der Berechnungen in Tabellen entnehmen.

### Verschachtelte Feldfunktionen

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 eingblendeten 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 7.16 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. 7.16** 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 }" }

zehnmillionendrehundertsechsfünfzigtausendzweihundertachtunddreißig
```

Die Prozedur *CardTextFunktionAufbauen* ruft die Funktion *FeldCodeEinfuegen* in Listing 7.16 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 den 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 die Feldfunktion enthaltenden Bereich an die rufende Prozedur *CardTextFunktionAufbauen* zurück, wo die Feldfunktion im Bereich aktualisiert wird.

**ACHTUNG** Bitte beachten Sie, dass die Funktion nicht kontrolliert, ob die Feldfunktion gültig ist oder ob der Zielbereich eine Feldfunktion akzeptieren kann.

**Listing 7.16** 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 = "~"
    Dim lIndex As Long
    Dim bEscapeCharFound As Boolean
    Dim sBuffer As String
    Dim sChar As String
    Dim rngStart As Word.Range
```



Listing 7.16 Eine Zeichenkette in eine Feldfunktion umwandeln (Fortsetzung)

```

bEscapeCharFound = False
sBuffer = ""
Set rngZielBereich = 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

Sub CardTextFunktionAufbauen()
    Dim strTeststrFeldCode As String
    Dim rngTarget As Word.Range

```

**Listing 7.16** Eine Zeichenkette in eine Feldfunktion umwandeln (*Fortsetzung*)

```

Set rngTarget = Selection.Range
strTeststrFeldCode = rngTarget.Text
rngTarget.InsertAfter vbCrLf
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 7.17. Somit bleibt Ihnen erspart, eine komplexe Feldfunktion wie diejenige in Abbildung 7.16 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 7.17** 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

    'Das Resultat in die Zwischenablage übernehmen, sodass der Benutzer

```

Listing 7.17 Diese Prozedur wandelt die markierte Feldfunktion in einfachen Text um

```
'es einfügen kann, wo er will.
Set oData = New DataObject
oData.SetText strNeu
oData.PutInClipboard
End Sub
```



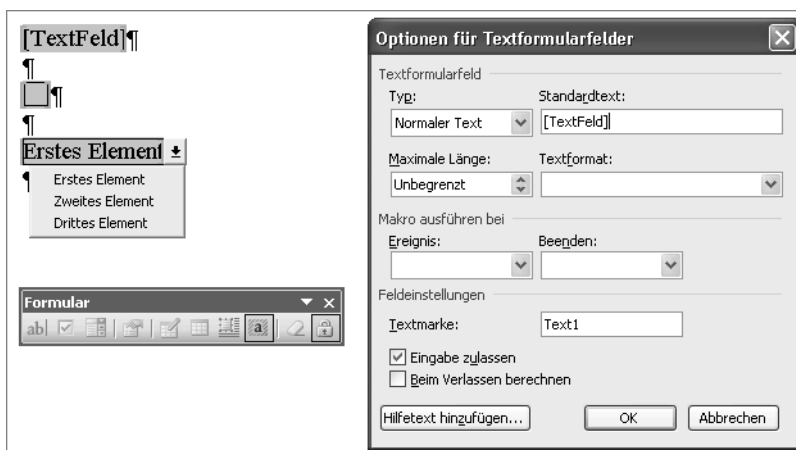
Die Beispieldatei *Bsp07\_02\_Field.doc* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap07*.


## 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*.

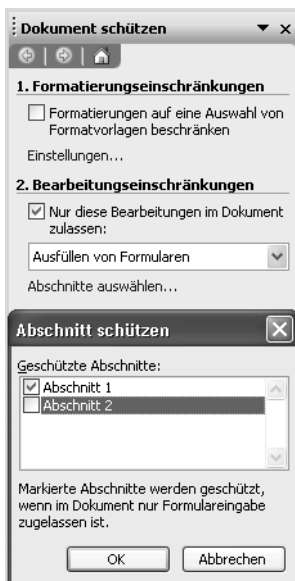
Es gibt drei Arten von Formularfeldern: Text, Kontrollkästchen und Dropdownliste (siehe Abbildung 7.17). Ihnen können Namen zugewiesen werden, die gleichzeitig als Textmarken dienen. Es ist auch möglich, dafür einen Hilfetext zu definieren, der in der Statusleiste oder durch Drücken von **[F1]** eingeblendet 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. 7.17 Formfelder, 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 7.18).

**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. 7.18** Der Aufgabenbereich für den Dokumentschutz ab Word 2002. 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 12. Ab Word 2007 sind Inhaltssteuerelemente zu empfehlen, die im nächsten Abschnitt vorgestellt werden.

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.

### Formularfeldwerte

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 Dropdownfeld 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 kann ebenfalls über die Result-Eigenschaft abgefragt werden. Es kann auch so deaktiviert, aber nicht aktiviert werden:

```
lKontrollkästchenWert = doc.FormFields("Kontrollkästchen1").Result
doc.FormFields("Kontrollkästchen1").Result = 0 'Nicht aktiviert. Aktiviert wäre 1
```

Besser ist, das Kontrollkästchen konsequent über die Eigenschaft Value festzulegen:

```
doc.FormFields("Kontrollkästchen1").CheckBox.Value = True
```

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 7.18 bzw. das Listing 7.19 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 7.18** Prüfen, ob das markierte Formfeld ein Kontrollkästchen ist

```

Sub Test()
    Dim rng As Word.Range

    Set rng = Selection.Range
    MsgBox IstKontrollkästchen(rng)
End Sub

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 7.19** 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 feldName = rng.Bookmarks.get_Item(ref objIndex1).Name;
            object objFeldName = (object) feldName;
            wd.FormField ffld = rng.Document.FormFields.get_Item(ref objFeldName);
            if (ffld.CheckBox.Valid)
            {test = true;}
        }
    }
    return test;
}

```



Die Beispieldatei *Bsp07\_01\_Form.doc* mit der Prozedur finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap07*.

## Dokument 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 ab 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 ab Word 2003 unterstützt und ist das Gegenstück zur Auswahl »Keine Änderungen (schreibgeschützt)« in der Liste *Bearbeitungseinschränkungen* des Aufgabenbereichs *Dokument schützen* (Abbildung 7.18).

`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. Durch Einfügen eines Formulars in ein anderes Dokument erscheint der Inhalt des Formulars, auch ein per Kennwort geschütztes, im Zieldokument ungeschützt.

---

Das Argument `UseIRM` ist nur ab 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 erst ab Word 2003 verfügbar und schaltet die Formatierungseinschränkungen ein (siehe auch den Abschnitt »Formatieren mit Stil: Das *Style*-Objekt« in Kapitel 6). 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 7.20 bzw. das Listing 7.21, 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 ungeschützten Zustand vorliegt.

**Listing 7.20** 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 7.21** 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 Tabellenzeile befinden, da der Anwender einzelne Zeichen nicht markieren kann.



Die Beispieldatei *Bsp07\_01\_Form.doc* mit der Prozedur finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap07*.

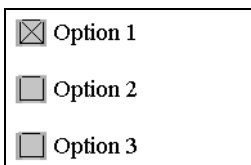


## Die Ereignisse

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 von Optionsfelder benutzen lässt (Abbildung 7.19). 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. 7.19 Kontrollkästchen verhalten sich wie Optionsfelder dank der Ereignis-Makros



Der Code hierfür befindet sich in Listing 7.22. Der `EntryMacro`-Eigenschaft jedes Kontrollkästchens wurde »KontrollkästchenEintreten« zugewiesen und der `ExitMacro`-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«-Variablen in die »VorherigesFeld«-Variable geschrieben, und der »AktuellesFeld«-Variablen 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 7.22** Kontrollkästchen wie Optionenfelder 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
    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 *Bsp07\_01\_Form.doc* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap07*.

**HINWEIS**

In Kapitel VII des Bonusteils auf der Buch-CD 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.

## Die Alternative zu Formularfeldern: ContentControls-Objekt



2007

Formulare, wir können unsere heutige Gesellschaft ohne diese Papierstapel kaum vorstellen. Vor fünfzehn Jahren, als Word 6.0 entwickelt wurde, war das papierlose Büro ein Märchen. So weit sind wir noch nicht, aber die elektronische Datenaufbewahrung schreitet zügig voran. Längst haben wir uns daran gewöhnt, Informationen in Datenmasken und über Internetseiten, und dadurch in eine Datenbank, einzugeben.

Für kurze Datenmengen geht das auch. Aber manchmal lässt sich eine Arbeit einfacher erledigen in der gestalterischen freieren Umgebung einer Textverarbeitung, als in einer Datenmaske. Statt einem Formular soll ein strukturiertes Dokument entstehen, das zusätzlich der Datengewinnung dient. Im Geschäftsalltag begegnen wir beispielsweise den folgenden Szenarien:

- Ein Teammitglied erstellt ein Word-Dokument mit Formularfeldern. Es wird an dutzende Leute geschickt und kommt ausgefüllt zurück. Jetzt sollen die Daten ausgewertet werden, und er merkt erst jetzt, wie viel Aufwand damit verbunden ist, die Eingaben aus jedem Dokument zu holen.
- Berichte auf Basis vorhandener Daten werden regelmäßig erstellt, und diese durch den Mensch angepasst oder ergänzt. Diese neuen Informationen sollen wiederum aus dem Bericht gewonnen und in die Datenbank zurückgespeichert werden.
- Ein Artikel muss in mehreren Medienarten veröffentlicht und der Inhalt zudem in einer Datenbank gespeichert werden.

Bislang bot Word für die Datensammlung Formularfelder (Abschnitt »Formulare: Das *FormField*-Objekt«) und ActiveX-Steuerelemente (Kapitel 12) an. Der große Nachteil von Formularfeldern ist, dass durch den Dokumentschutz die Stärken der Textverarbeitung größtenteils verloren gehen, da die Funktionalität gesperrt wird. Beispielsweise stehen weder Grafiken noch Formatierungswerkzeuge zur Verfügung, es sei denn viele Ressourcen werden in eine Makro-Lösung investiert. ActiveX-Steuerelemente sind Fremdkörper in einem Word-Dokument. Sie passen sich dem Textfluss nicht an, lösen die Sicherheitswarnung aus und verhalten sich »komisch«, wenn mit der Bildlaufleiste durch das Dokument geblättert wird. Eine bessere Lösung drängt sich auf.

Mit der Einführung von XML in Word als Dateiformat (Kapitel 22) wurde, was die Datengewinnung angeht, ein großer Schritt vorwärts gemacht. Der Dokumentinhalt liegt im strukturierten Textformat offen. Die Word-Anwendung muss nicht mehr automatisiert, oder gar vorhanden sein, um den Inhalt daraus zu lesen oder zu bearbeiten. Somit wurde die Zeit reif, sich der Benutzerschnittstelle zu widmen. Das Resultat sehen wir in der Form von Inhaltssteuerelementen in Word 2007. Nachfolgend einige Punkte, die Inhaltssteuerelemente auszeichnen:

- Inhaltssteuerelemente fügen sich in den Textfluss nahtlos ein. Werden sie weder durch den Mauszeiger noch die Einfügemarke berührt, bleiben sie unsichtbar.

- Verschiedene Schutzstufen, bis zum Dokumentschutz für Formularfelder, stehen zur Verfügung. Wo und was der Benutzer im Dokument tun darf ist steuerbar.
- Einige Inhaltssteuerelemente (Text, Bild, Kombinationsfeld, Dropdownliste, Datumsauswahl) können mit einem zusätzlich im Dokument gespeicherten XML-Teil verbunden werden. Dadurch erleichtert sich der Datenaustausch.
- Dank ihren Ereignissen lassen sich Inhaltssteuerelemente durch den Entwickler verfeinert steuern, als sonst in Word-Objektmodell üblich.

In den folgenden Abschnitten werden Inhaltssteuerelemente und ihre programmtechnischen Schnittstellen eingehender vorgestellt.

#### HINWEIS

Da Inhaltssteuerelemente, wie Word 2007, noch neu sind, sind Internetlinks zu Artikeln spärlich. Deutschsprachige haben wir keine gefunden. Die folgenden Seiten in englischer Sprache enthalten jedoch interessante Informationen:

- »The Microsoft Office Word Team's Blog« auf [http://blogs.msdn.com/microsoft\\_office\\_word/archive/tags/content+controls/default.aspx](http://blogs.msdn.com/microsoft_office_word/archive/tags/content+controls/default.aspx)
- Verschiedene Artikel sowie Demos von Mike Ormond, die die Arbeit mit Inhaltssteuerelementen aus der Sicht der .NET-Entwickler vorstellen auf <http://blogs.msdn.com/mikeormond/search.aspx?q=Word+Content+Controls&p=1>

## Die Grundlagen

Die Werkzeuge für Inhaltssteuerelemente befinden sich in der Gruppe *Steuerelemente* der Registerkarte *Entwicklertools* (Abbildung 7.20). Die zur Verfügung stehenden Inhaltssteuerelemente sind in Tabelle 7.4 aufgelistet mit ihrer *WdContentControlType*-Enumeration.

#### HINWEIS

Eine breite Palette Steuerelemente wird zur Verfügung gestellt, auffallend ist jedoch das Fehlen von Kontrollkästchen und Optionsfeldern. Microsoft ist der Meinung, dass Word für die Erstellung von reinen Formularen nicht das geeignete Werkzeug sei. Dafür stellt es die Anwendung »InfoPath« zur Verfügung. Die in Word bereitgestellten Inhaltssteuerelemente dienen hauptsächlich dem Erstellen von strukturierten Dokumenten und dem damit verbundenen Datenaustausch.

Abbildg. 7.20 Die Gruppe *Steuerelemente* auf der Registerkarte *Entwicklertools*

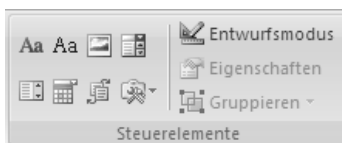










Tabelle 7.4 Die einzelne Inhaltssteuerelemente mit *WdContentControlType*-Enumeration

Symbol	Name	Enum	Konstantwert
	Rich-Text (formatierter Text)	wdContentControlRichText	5
	Nur-Text	wdContentControlText	3
	Bild	wdContentControlPicture	6
	Kombinationsfeld	wdContentControlComboBox	4
	Dropdownliste	wdContentControlDropDownList	2
	Datumsauswahl	wdContentControlDate	0
	Bausteinkatalog	wdContentControlBuildingBlockGallery	1
	Gruppieren	wdContentControlGroup	7

Um ein Inhaltssteuerelement in das Dokument einzufügen, positionieren Sie die Einfügemarke, dann klicken Sie auf die passende Schaltfläche. Das Inhaltssteuerelement erscheint im Dokument, wie links in Abbildung 7.21 ersichtlich. Falls Sie den Platzhaltertext ändern möchten, klicken Sie auf die Schaltfläche *Entwurfsmodus* und geben Sie den gewünschten Text ein (rechts im Bild).

Abbildg. 7.21 Nur-Text-Inhaltssteuerelemente im normalen sowie Entwurfsmodus



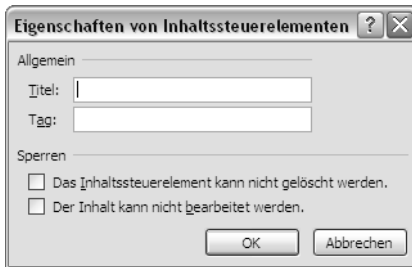
**HINWEIS** Um die Anzeige des Platzhaltertexts zu unterdrücken, wird die Eigenschaft des *ShowingPlaceholderText* des *ContentControl*-Objekts gebraucht. Programmäßig wird er mit der Methode *SetPlaceholderText* festgelegt, und mit der Eigenschaft *PlaceholderText* gelesen.

Das Aussehen des standardmäßigen Platzhaltertexts wird durch die Formatvorlage Platzhaltertext festgelegt. Standardmäßig übernimmt sie die Formatierung der Standardschriftart. Möchten Sie »leere« Inhaltssteuerelemente besser hervorheben, ändern Sie die Definition dieser Formatvorlage.

### Eigenschaften der Inhaltssteuerelemente

Durch Anklicken der Schaltfläche *Eigenschaften* können Aussehen und Verhalten der Inhaltssteuerelemente festgelegt werden. Alle Inhaltssteuerelemente haben die Eigenschaften in Abbildung 7.22 gemeinsam. Tabelle 7.5 bietet eine Übersicht aller Inhaltssteuerelement-Eigenschaften. Tiefer gehende Informationen dazu finden Sie in der Hilfe zum Word 2007-Objektmodell.

**Abbildg. 7.22** Eigenschaftenfenster eines Grafik-Inhaltssteuerelements, die alle Inhaltssteuerelemente gemeinsam haben



**Tabelle 7.5** Eigenschaften der Inhaltssteuerelemente

Eigenschaft	Beschreibung	Steht den Inhaltssteuer- elementen zur Verfügung
<i>Titel</i>	Eine Beschriftung, die über dem Inhaltssteuerelement erscheint, wenn dieses sichtbar ist. Dieser Wert muss <i>nicht</i> zwingend eindeutig sein. Steht dem Entwickler für die Identifikation zur Verfügung über die Methode <b>SelectContentControlsByTitle</b> .	Alle 
<i>Tag</i>	Stellt dem Entwickler einen Datenbehälter für jedes Inhaltssteuerelement bereit. Steht dem Entwickler für die Identifikation zur Verfügung über die Methode <b>SelectContentControlsByTag</b> .	Alle 
<i>Das Inhaltssteuer- element kann nicht gelöscht werden</i>	Der Benutzer kann das Inhaltssteuerelement nicht aus dem Dokument löschen. Wird im Objektmodell durch die Eigenschaft <b>LockContentControl</b> dargestellt.	Alle 
<i>Der Inhalt kann nicht bearbeitet werden</i>	Das Inhaltssteuerelement ist gesperrt. Wird im Objektmodell durch die Eigenschaft <b>LockContents</b> dargestellt.	Alle 
<i>Formatvorlage zum Formatieren von In- halt verwenden</i>	Der Inhalt wird zwingend mit der angegebenen Zeichen-Formatvorlage formatiert. Wird im Objektmodell durch die Eigenschaft <b>DefaultTextStyle</b> dargestellt.	Nur-Text, Rich-Text, Dropdownliste, Kombinationsfeld, Datumsauswahl, Bausteinkatalog 
<i>Wagenrückläufe zu- lassen</i>	Ermöglicht die Eingabe mehrerer Absätze in ein Nur-Text-Inhaltssteuerelement. Wird im Objektmodell durch die Eigenschaft <b>MultiLine</b> dargestellt.	Nur-Text 
<i>Inhaltssteuer- element beim Bearbei- ten des Inhalts löschen</i>	Das Inhaltssteuerelement wird bei der Texteingabe aus dem Dokument entfernt. Wird im Objektmodell durch die Eigenschaft <b>Temporary</b> dargestellt.	Nur-Text, Rich-Text 

Tabelle 7.5 Eigenschaften der Inhaltssteuerelemente (Fortsetzung)

Eigenschaft	Beschreibung	Steht den Inhaltssteuer- elementen zur Verfügung
Anzeigename	Die Beschriftung eines Listeneintrags. Wird im Objektmodell durch die Eigenschaft <b>Text</b> eines <b>DropDownListEntries.Item</b> dargestellt.	Dropdownliste, Kombinationsfeld 
Wert	Unsichtbarer Inhalt eines Listeneintrags; hat den Datentyp »String«. Wird im Objektmodell durch die Eigenschaft <b>Value</b> eines <b>DropDownListEntries.Item</b> dargestellt.	Dropdownliste, Kombinationsfeld 
Die Schaltflächen <i>Hinzufügen</i> sowie <i>Nach oben</i> und <i>Nach unten</i>	Fügt der Auflistung einen neuen Eintrag zu. Wird im Objektmodell durch die Methode <b>Add</b> dargestellt. Ändert die Position eines Eintrags in der Liste. Werden im Objektmodell durch die Methoden <b>MoveUp</b> und <b>MoveDown</b> dargestellt. Die gegenwärtige Position gibt die Eigenschaft <b>Index</b> zurück.	Dropdownliste, Kombinationsfeld 
Datum wie folgt anzeigen	Legt das Erscheinungsbild des Inhalts fest. Wird im Objektmodell durch die Eigenschaft <b>DateDisplayFormat</b> dargestellt.	Datumsauswahl 
Gebietsschema	Legt das übergeordnete regionale Datumsformat fest. Wird im Objektmodell durch die Eigenschaft <b>DateDisplayLocale</b> dargestellt, die einen <b>WdLangaugeID</b> -Konstantwert erwartet.	Datumsauswahl 
Kalendertyp	Legt die Art des Kalenders (beispielsweise chinesisch, hebräisch, arabisch usw.) fest. Wird im Objektmodell durch die Eigenschaft <b>DateCalendarType</b> dargestellt, die einen <b>WdCalendarType</b> -Konstantwert erwartet.	Datumsauswahl 
XML-Inhalt im folgenden Format speichern	Legt das Format (Datum, Datum & Zeit oder Zeichenkette) fest, in welchem der Inhalt gespeichert werden soll. Wird im Objektmodell durch die Eigenschaft <b>DateStorageFormat</b> dargestellt, die einen <b>WdContentControlDateStorageFormat</b> -Konstantwert erwartet.	Datumsauswahl 
Katalog	Legt den Katalog fest, aus welchem die aufzulistenden Bausteine zu holen sind. Wird im Objektmodell durch die Eigenschaft <b>BuildingBlockCategory</b> dargestellt, die einen <b>WdBuildingBlockTypes</b> -Konstantwert erwartet.	Bausteinkatalog (Mehr zum Thema Bausteine lesen Sie in Kapitel 6) 
Kategorie	Legt die Kategorie fest, aus welcher die aufzulistenden Bausteine zu holen sind. Wird im Objektmodell durch die Eigenschaft <b>BuildingBlockType</b> dargestellt.	Bausteinkatalog 

## Dokumente strukturiert bearbeiten

Eine der Stärken der Inhaltssteuerelemente ist die verfeinerte Kontrolle über die Dokumentbearbeitung. Die Spannweite reicht vom einfachen »Klick hier«-Zielfeld bis zur Sperre aller Bereiche außer denen ausgewählter Nur-Text-Inhaltssteuerelemente, wie in einem herkömmlichen Formular. Folgend eine Übersicht der verschiedenen Schutzstufen. Viele dieser Optionen können kombiniert werden, um den gewünschten Schutzgrad zu erreichen:

- »Klick hier« mit aktivierter Option *Inhaltssteuerelement beim Bearbeiten des Inhalts löschen*. Das Inhaltssteuerelement verschwindet bei der Texteingabe.
- »Klick hier«, ohne aktivierte Optionen. Das Inhaltssteuerelement bleibt im Dokument, außer der Benutzer markiert und löscht es, samt Inhalt.
- »Klick hier« mit aktivierter Option *Das Inhaltssteuerelement kann nicht gelöscht werden*. Stellt sicher, dass das Inhaltssteuerelement nicht versehentlich aus dem Dokument gelöscht werden kann.
- »Klick hier« mit aktivierter Option *Formatvorlage zum Formatieren von Inhalt verwenden*. Damit hat der Benutzer keinen Einfluss auf die Formatierung des Inhalts.
- »Klick hier« Inhaltssteuerelemente und der umgebende Textbereich sind gruppiert und die Option *Das Inhaltssteuerelement kann nicht gelöscht werden* ist für das Gruppe-Inhaltssteuerelement aktiviert. Innerhalb des Gruppe-Inhaltssteuerelements kann nur in den darin enthaltenen Inhaltssteuerelementen editiert werden, der normale Text ist geschützt. Die übrigen Dokumentbereiche bleiben normal bearbeitbar.
- Der Schreibschutz ist auf Dokumentebene aktiviert. Dem Benutzer stehen im Dokument nur Inhaltssteuerelemente zur Verfügung, die sich in einem freigegebenen Bereich befinden. Dieser Schutz schließt die Möglichkeit aus, dass der Benutzer die Eigenschaften der Inhaltssteuerelemente ändern könnte.
- Der Formularschutz ist aktiviert. Inhaltssteuerelemente verhalten sich wie Formularfelder; Formatierungsbefehle sind gesperrt. Bild- und Datumsauswahl-Inhaltssteuerelemente funktionieren weiterhin. Baustein-Inhaltssteuerelemente sind jedoch ebenfalls gesperrt.


Da das Konzept der Gruppierung neu ist, wird sie hier anhand eines Beispiels näher vorgestellt. Eine Firma stellt für Berichte eine Dokumentvorlage zur Verfügung, die das Deckblatt »Jährlich«, aus dem Katalog unter *Einfügen/Deckblatt*, als erste Seite hat. Titel, Untertitel, Datum sowie Exposé werden in Inhaltselemente eingegeben, die sich in einer frei auf der Seite positionierten Tabelle befinden (Abbildung 7.23).



Abbildg. 7.23 Deckblatt mit Inhaltssteuerelementen in einer Tabelle

The screenshot shows a document window titled 'Seite 1 von 2'. The main content area displays a cover page layout. At the top, there is a table with two columns. The left column contains the text '[Geben Sie den Titel des Dokuments ein]' and the right column contains '[Wählen Sie das Datum aus] [Jahr]'. Below the table, there is a large text area on the left with a placeholder text: '[Geben Sie hier das Exposee für das Dokument ein. Das Exposee ist meist eine Kurzbeschreibung des Dokumentinhalts. Geben Sie hier das Exposee für das Dokument ein. Das Exposee ist meist eine Kurzbeschreibung des Dokumentinhalts.]'. To the right of this text area, there is another text area with the text '[Geben Sie den Untertitel des Dokuments ein]'. The document window has a standard toolbar at the top with icons for undo, redo, and other editing functions. On the right side of the window, there are buttons for 'Ansichtsoptionen' and 'Schließen'.

Im ursprünglichen Zustand sind weder Tabelle noch Inhaltssteuerelemente geschützt. Die Tabelle könnte verschoben, die Inhaltssteuerelemente gelöscht, oder Text außerhalb den Inhaltssteuerelementen eingegeben werden. Um das Deckblatt vor unerwünschten Änderungen zu schützen, wird wie folgt vorgegangen:

1. Positionieren Sie den Mauszeiger über die Tabelle, bis der Anfasser oben links erscheint. Klicken Sie darauf, um die Tabelle zu markieren.
2. Halten Sie die -Taste fest und mit der Maus die Absatzmarke sowie Seitenumbruch zu oberst auf der Seite markieren.
3. Klicken Sie auf die Schaltfläche *Gruppieren* und aus der Liste den gleichnamigen Eintrag wählen (Abbildung 7.24).

Abbildg. 7.24

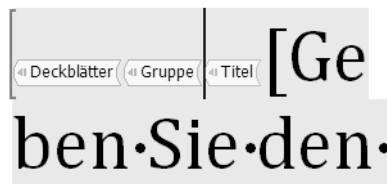
Die zu schützenden Bereiche markieren und gruppieren



4. Klicken Sie auf die Schaltfläche *Entwurfsmodus*, dann die Einfügemarke neben dem Element *Gruppe* positionieren, wie in Abbildung 7.25 ersichtlich.

Abbildg. 7.25

Das *Gruppe*-Inhaltssteuerelement hat den Fokus



5. Klicken Sie auf die Schaltfläche *Eigenschaften* und das Kontrollkästchen *Das Inhaltssteuerelement kann nicht gelöscht werden* aktivieren. (Dieses soll das einzige Steuerelement im Dialogfeld sein. Wenn Sie etwas anderes sehen, dann ist der Fokus nicht in dem *Gruppe*-Element.)
6. Schalten Sie den Entwurfsmodus aus, in dem Sie nochmals auf die Schaltfläche klicken.

Im Deckblatt kann der Benutzer einzig mit den Inhaltssteuerelementen arbeiten. Ansonsten darf er im Dokument wie üblich arbeiten. Falls andere Eigenschaften auf Dokumentebene, wie Seitengröße, Kopf- und Fußzeile geschützt werden sollen, fahren Sie wie folgt fort:

1. Klicken Sie auf die Schaltfläche *Dokument schützen*; der Aufgabenbereich *Formatierung und Bearbeitung* wird eingeblendet. Aktivieren Sie dort, unter Schritt 2, *Nur diese Bearbeitungen im Dokument zulassen* mit ausgewähltem Eintrag *Keine Änderungen (Schreibgeschützt)*.
2. Klicken Sie nochmals auf den Tabellenanfasser, um die Tabelle zu markieren, dann aktivieren Sie das Kontrollkästchen *Jeder* im Aufgabenbereich.
3. Gehen Sie zur zweiten Seite und markieren Sie den ganzen Inhalt (Rest des Dokuments), dann aktivieren Sie abermals *Jeder*.
4. Klicken Sie auf die Schaltfläche *Ja, Schutz anwenden* im Aufgabenbereich und, sofern gewünscht, ein Kennwort eingeben.

**HINWEIS**

Die Beispielvorgabe *Bsp07\_01\_CC.dotm* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap07*.

Falls Sie schon das Kapitel 22 gelesen haben oder bereits mit XML in Word vertraut sind, ist Ihnen wahrscheinlich der Bezeichner der Inhaltssteuerelemente in Abbildung 7.25 aufgefallen. Sie sehen XML-Knotenpunkten aus einem angefügten Schema sehr ähnlich, nur sind sie weiß statt rosa. Damit liegt die Vermutung nahe, dass Inhaltssteuerelemente auf der Word-XML-Funktionalität basieren, was auch stimmt und uns zum nächsten Abschnitt bringt.

## Inhaltssteuerelemente und XML

Die Abbildung 7.26 bietet einen Auszug des WordProcessingML im oben vorgestellten Beispieldokument. Teile der Definition zweier Inhaltssteuerelemente – die Gruppe sowie der Titel – sind sichtbar. Wie Sie sehen können, wird ein Inhaltssteuerelement durch das Element *w:sdt* eröffnet. Die Verschachtelung des Rich-Text-Inhaltssteuerelements in dem Gruppe-Inhaltssteuerelement ist erkennbar.

Einige der eingangs vorgestellten Eigenschaften sind auch zu sehen, beispielsweise stellt *alias* das Gegenstück zur Eigenschaft *Titel* dar, während *lock* der Eigenschaft *Das Inhaltssteuerelemente kann nicht gelöscht werden* entspricht. Das Attribut *id* wird von Word zugewiesen und identifiziert ein Inhaltssteuerelement eindeutig.

Abbildg. 7.26 Das *WordProcessingML* einiger Inhaltssteuerelemente der Beispielvorlage

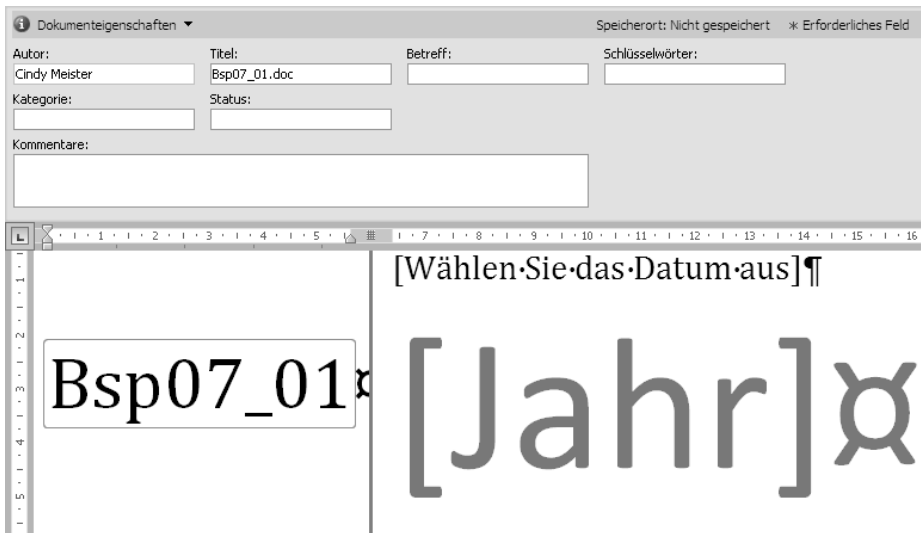
```
- <w:body>
- <w:sdt>
- <w:sdtPr>
+ <w:rPr>
  <w:id w:val="177890799" />
- <w:docPartObj>
  <w:docPartGallery w:val="Cover Pages" />
  <w:docPartUnique />
</w:docPartObj>
</w:sdtPr>
- <w:sdtEndPr>
+ <w:rPr>
</w:sdtEndPr>
- <w:sdtContent>
- <w:sdt>
- <w:sdtPr>
+ <w:rPr>
  <w:id w:val="177890875" />
  <w:lock w:val="sdtContentLocked" />
- <w:placeholder>
  <w:docPart w:val="DefaultPlaceholder_22675703" />
</w:placeholder>
  <w:group />
</w:sdtPr>
- <w:sdtEndPr>
+ <w:rPr>
</w:sdtEndPr>
- <w:sdtContent>
- <w:tbl>
+ <w:tblPr>
+ <w:tblGrid>
- <w:tr w:rsidR="000927D1">
- <w:sdt>
- <w:sdtPr>
+ <w:rPr>
  <w:alias w:val="Titel" />
  <w:id w:val="276713177" />
+ <w:placeholder>
  <w:showingPlcHdr />
  <w:dataBinding
    w:prefixMappings="xmlns:ns0='http://schemas.openxmlformats.org/package/2006/metadata/core-properties' xmlns:ns1='http://purl.org/dc/elements/1.1/'" w:xpath="/ns0:coreProperties
    [1]/ns1:title[1]" w:storeId="{6C3C8BC8-F283-45AE-878A-BAB7291924A1}" />
  <w:text />
</w:sdtPr>
```

## Datenverbindungen

Von besonderem Interesse ist das Element `w:dataBinding`. Dadurch wird das Inhaltssteuerelement mit einem XML-Teil in der OpenXML-Dokument-Verpackungseinheit verknüpft. In diesem Fall ist das Inhaltssteuerelement für den Titel mit der gleichnamigen Word-eigenen Dokumenteigenschaft (»core-properties«) *Title* verbunden. Was der Benutzer in das Inhaltssteuerelement eingibt wird automatisch in die Dokumenteigenschaft gespeichert; umgekehrt erscheint der Wert der Dokumenteigenschaft im Inhaltssteuerelement.

Sie können dies leicht selber testen. Geben Sie in das Inhaltssteuerelement für den Titel einen Text ein. Klicken Sie auf die *Office*-Schaltfläche und führen Sie die Befehlsfolge *Vorbereiten/Eigenschaften* aus. Ein Balken mit einigen Dokumenteigenschaften, u.a. der Titel, erscheint über dem Dokument, wie in Abbildung 7.27 ersichtlich. Der Text im Inhaltssteuerelement ist gleich wie im Feld *Titel*. Nun ändern Sie den Text der Dokumenteigenschaft. Sobald der Fokus das Textfeld verlässt, wird der Eintrag im Inhaltssteuerelement aktualisiert.

Abbildg. 7.27 Der Text im Inhaltssteuerelement ist mit der Dokumenteigenschaft *Titel* verbunden



#### HINWEIS

Es ist leider nicht möglich, Inhaltssteuerelemente mit benutzerdefinierten Dokumenteigenschaften zu verbinden.

#### Benutzerdefinierter XML-Teil

Noch wichtiger für den Datenaustausch und die Datengewinnung ist, dass die meisten Inhaltssteuerelementtypen mit benutzerdefinierten XML-Teilen in der OpenXML-Verpackungseinheit verbunden werden können. Die Einbindung des XML-Teils in das Dokument kann manuell oder programmtechnisch erfolgen. Die programmtechnische Methode ist eingehend im Abschnitt Inhaltssteuerelemente im Objektmodell beschrieben.

Die manuelle Methode ist ähnlich der in Kapitel 17 für die Multifunktionsleiste beschriebenen: Das Dokument wird mit der Dateiendung *.zip* umbenannt und mehrere XML-Teile geöffnet und bearbeitet. Anleitungen stehen im Internet auf mehreren Webseiten bereit (siehe den nachfolgenden Hinweis). Der Weg über OpenXML ist hauptsächlich für den Entwickler gedacht, der in seinem Programm Word-Dateien in der Form von OpenXML erstellt oder bearbeitet. Der »Power-User« darf jedoch auch davon Gebrauch machen. Für ihn gibt es ein kostenloses Werkzeug zum Herunterladen, das diese Arbeit erheblich erleichtert: das »Word 2007 Content Controls Toolkit«. Wir stellen es anhand eines Beispiels (Abbildung 7.28) kurz vor.

#### HINWEIS

Links, die sich mit dem manuellen Einbinden von XML-Teilen und ihrer Verknüpfung zu Inhaltssteuerelementen befassen:

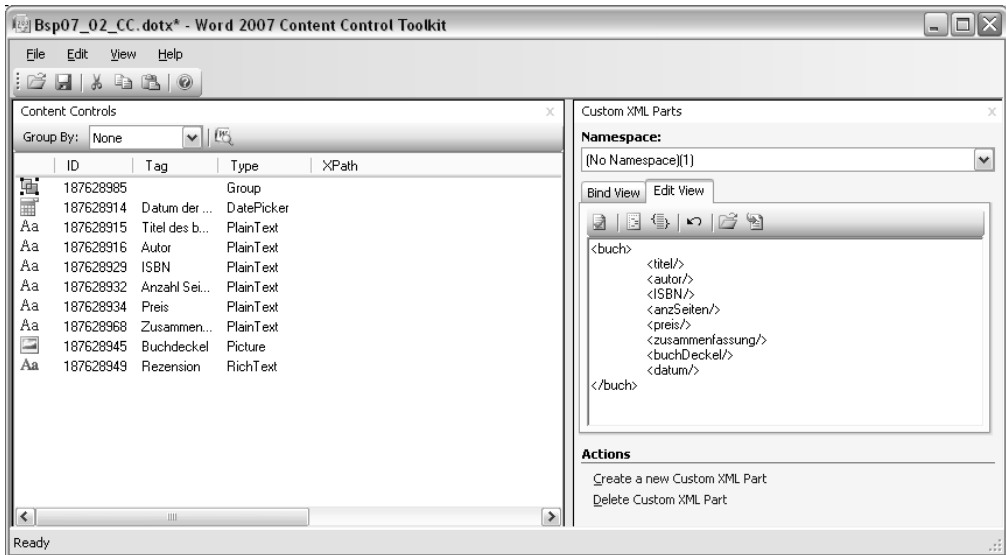
- »Word 2007 Content Controls and XML – Part 1 (the basics)« <http://blogs.msdn.com/modovan/archive/2006/05/23/604704.aspx>
- »Linking Word 2007 Content Controls to Custom XML« <http://blogs.msdn.com/acoat/archive/2007/03/01/linking-word-2007-content-controls-to-custom-xml.aspx>
- »Word 2007 Content Control Toolkit« (.NET Framework 2.0 muss auf dem Rechner installiert sein) <http://www.codeplex.com/Wiki/View.aspx?ProjectName=db>

**Abbildg. 7.28** Dokumentinhalt in Inhaltssteuerelementen mit einem benutzerdefinierten XML-Teil verbinden


Erstellen Sie zuerst das Dokument (oder die Vorlage) mit Inhaltssteuerelementen, dann speichern und schließen Sie es. Starten Sie das »Word 2007 Content Control Toolkit« und öffnen Sie das Word-Dokument (das Werkzeug arbeitet mit der OpenXML-Verpackungseinheit; es automatisiert Word *nicht*).

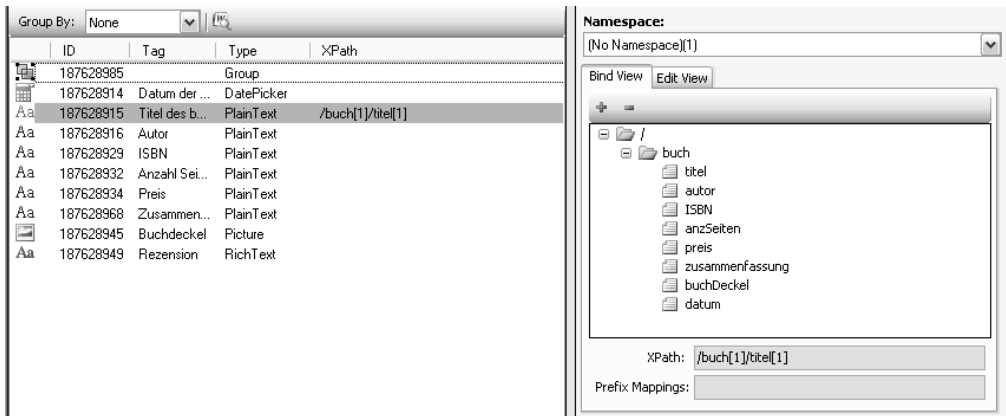
Da das Dokument noch keinen benutzerdefinierten XML-Teil enthält, klicken Sie auf den Link *Click here to create new one* im rechten Fenster. Ein XML-Teil namens *CustomXML* wird im Dokument mit allen benötigten Verweisen erstellt. Klicken Sie nun auf den Registerreiter *Edit View* (Abbildung 7.29). In diesem Fenster wird der XML-Code, der den Inhaltssteuerelementen im Dokument entspricht, eingegeben oder eine auf der Festplatte gespeicherte XML-Datei geöffnet (für das Beispiel steht die Datei *Bsp07\_02\_CC.xml* zur Verfügung).

Abbildg. 7.29 »Content Control Toolkit«: Benutzerdefiniertes XML in einen XML-Teil einfügen



Wechseln Sie zurück zur Registerkarte *Bind View*, um die XML-Knotenpunkte mit den Inhaltssteuerelementen zu verbinden. Klicken Sie auf einen XML-Knotenpunkt, dann klicken Sie nochmals darauf und halten die linke Maustaste gedrückt. Ziehen Sie den Knotenpunkt über das passende Inhaltssteuerelement im linken Fenster und geben Sie die Maustaste wieder frei (Abbildung 7.30). Wiederholen Sie diese Schritte für jeden XML-Knotenpunkt, der mit einem Inhaltssteuerelement verbunden werden soll. Anschließend speichern und schließen Sie das Dokument. Falls Ihnen ein Fehler unterläuft, klicken Sie mit der rechten Maustaste auf den Inhaltssteuerelement-Eintrag und öffnen über das Kontextmenü die Eigenschaften (*Properties*). Dort können die Verbindungen wieder gelöscht werden.

Abbildg. 7.30 »Content Control Toolkit«: XML-Knotenpunkte mit Inhaltssteuerelementen verbinden



**HINWEIS**

Es wird auffallen, dass für das Inhaltssteuerelement *Rezension* kein XML-Knotenpunkt vorhanden ist. Rich Text- und Baustein-Inhaltssteuerelemente können nicht mit einem XML-Teil verbunden werden.

Öffnen Sie das Dokument in Word und geben Sie Text in die Inhaltssteuerelemente ein. Speichern und schließen Sie das Dokument wieder und öffnen es erneut im »Content Control Toolkit«. Im Fenster *Edit View* werden die Texteingaben erscheinen. Ändern Sie den Inhalt einige der Elemente. Speichern, schließen und öffnen Sie das Dokument in Word erneut, um das Ergebnis zu sehen.



Die Beispieldateien *Bsp07\_02\_CC.dotx*, *Bsp07\_02\_CC.xml* sowie *Bsp07\_02\_CC.docx* (ein ausgefülltes Dokument) finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap07*.

Das Werkzeug »Word 2007 Content Control Toolkit« befindet sich im Ordner *\Beilagen\Word2007ContentControlToolkit*.

### Inhalte spiegeln

Ein wichtiger Bestandteil von Word ist die *Ref*-Feldfunktion, die den Inhalt einer Textmarke an anderen Stellen im Dokument wiedergibt. Word setzt sie beispielsweise für Querverweise ein. Der Formular-Entwickler benutzt sie, um den Inhalt eines Formularfelds mehrmals anzuzeigen, so dass der Benutzer die Daten nur einmal eingeben muss.

Die *Ref*-Feldfunktion funktioniert mit Inhaltssteuerelementen nicht zufrieden stellend, da das ganze Inhaltssteuerelement als Ergebnis erscheint. Der Benutzer kann dieses wie ein normales Inhaltssteuerelement bearbeiten aber, sobald die Feldfunktion aktualisiert wird, erscheint wieder der Inhalt der Textmarke. Das ist nicht nur verwirrend: es besteht auch die Gefahr des Datenverlusts.

Mit einer Datenverbindung ist das Problem gelöst. Gehen Sie wie oben beschrieben vor, um ein Dokument mit Inhaltssteuerelementen zu erstellen. Bei der Festlegung der Datenverbindung ziehen Sie den gleichen XML-Knotenpunkt zu jedem Inhaltssteuerelement, das den gleichen Inhalt anzeigen soll.

Anders als bei Formularfeldern und *Ref*-Feldfunktionen darf der Benutzer an allen Stellen den Inhalt bearbeiten. Falls Sie dies unterbinden möchten, aktivieren Sie die Eigenschaft *Der Inhalt kann nicht bearbeitet werden*.

**HINWEIS**

Da Rich-Text-Inhaltssteuerelemente mit einem XML-Part nicht verbunden werden können, kommen für die Texteingabe nur Nur-Text-Inhaltssteuerelemente in Betracht.

## Inhaltssteuerelemente im Objektmodell

Als Alternative zur manuellen Manipulation der Word-Datei besteht die Möglichkeit, die Datenverbindungen über das Objektmodell vorzunehmen. Dazu braucht es die gleichen Zutaten: ein Dokument mit Inhaltssteuerelementen sowie ein passendes XML, wobei das XML in der Form einer Datei oder einer Zeichenkette vorliegen darf. Dieses Prinzip veranschaulichen wir in einer Beispieldatei für die Erstellung von Firmenbriefen.

Das vorliegende Beispiel zeigt zudem auf, wie Sie programmtechnisch:

- Die Liste eines Dropdown- bzw. Kombinationsfeldes mit Einträgen belegen



- Inhalt in ein Inhaltssteuerelement eingeben
- Einen Satz wiederholte Knotenpunkte in den XML-Teil eingeben und mit dynamisch erstellten Inhaltssteuerelementen verbinden (»1:n«)
- Ein Inhaltssteuerelement erstellen
- Inhaltssteuerelemente schützen
- Die Ereignisse für Inhaltssteuerelemente einsetzen

Weiter werden in diesem Abschnitt die folgenden Themen vorgestellt

- Das Navigieren zwischen Inhaltssteuerelementen
- Den Umgang mit Namensräumen in XML-Teilen

Die Briefvorlage kurz vorgestellt



Die Firma Northwind stellt eine Vorlage mit Briefkopf zur Verfügung. Für Word 2007 wurde diese auf Vordermann gebracht. Sie wurde mit Inhaltssteuerelementen ausgestattet und zwar für: den Absenderort, das Datum, alle Teile der Empfängeradresse, die Betreffzeile, die Anrede, den Briefinhalt (der Bereich zwischen Anrede und Begrüßung), die Begrüßung sowie die Absenderunterschrift. In Abbildung 7.31 sehen Sie zudem eine Produkttabelle (im Rich-Text-Inhaltssteuerelement für den Briefinhalt) für eine Offerte, die ihrerseits Inhaltssteuerelemente für die Produktinformationen enthält.

Abbildg. 7.31 Ein Firmenbrief mit zu einem XML-Teil verbundenen Inhaltssteuerelement

Northwind-GmbH  
Abteilung-Verkauf

Seattle, den 28. September 2007

Alfreds Futterkiste  
Maria Anders  
Obere Str. 57  
12209 Berlin  
Deutschland

Brief an Kundschaft erstellen

Absender  
Nancy Davolio

Kunde  
Alfreds Futterkiste, Maria Anders

BriefTyp wählen

☐ allgemein

☒ Offerte

☐ Bestellung

☐ deutsch

☒ englisch

Abbrechen

OK

Produkt

Lagerbestand

☒ Chai 39

☒ Chang 17

☒ Aniseed Syrup 13

☐ Chef Anton's Cajun Seasoning 53

☐ Chef Anton's Gumbo Mix 0

Betreff: Betreff

Dear Ms. Maria Anders

Art-Nr	Beschreibung	Anzahl	Stückpreis	Preis
1	Chai	1	9,00	1,00
2	Chang	1	9,50	1,00
3	Aniseed Syrup	1	5,00	1,00
				3,00

Der Inhalt aller dieser Inhaltssteuerelemente ist mit einem im Dokument gespeicherten XML-Teil verbunden, mit Ausnahme jenes für den Briefinhalt, das ein Rich-Text-Inhaltssteuerelement ist und daher nicht verbunden werden kann. Dies ermöglicht Nordwind, über die Korrespondenz effizient Buch zu führen: Ohne die Dokumente in Word zu öffnen, weiß die Firma, wer an wen, wann geschrieben hat, worum es ging (Betreff) und, sofern es sich um eine Offerte handelte, was offeriert wurde.

Die Angaben für die Inhaltssteuerelemente werden über ein Formular (UserForm) vorgenommen. Der Benutzer kann die Art des Briefes wählen. Falls er die Option *Offerte* anklickt, wird rechts eine Galerie eingeblendet, aus der die Produkte auszuwählen sind. Bei der Initialisierung des Formulars werden die Daten für die Listen *Absender* und *Empfänger* aus einer Datenbank (in diesem Fall *Nordwind.mdb*), über eine ADO-Verbindung, gelesen. Die Produktliste wird erst beim Anklicken des Optionsfelds erstellt. Mit Hilfe der ADO-Methode *Save* und seiner Option *PersistFormat:=adPersistXML* werden diese Informationen lokal in *xml*-Dateien für den späteren Zugriff gespeichert.

Bei Bestätigung des Formulars werden die Einstellungen ausgewertet und in den Speicher eine *xml*-Datei mit der passenden Datenstruktur für die Art Brief (allgemein oder Offerte) mittels MSXML-DOM geladen. Die Knotenpunkte werden mit den benötigten Daten aus den vorher gespeicherten *xml*-Dateien gefüllt. Anschließend wird die im Speicher zusammengestellte *xml*-Datei einem XML-Teil (CustomXMLPart) zugewiesen.

Jetzt werden die Inhaltssteuerelemente im Dokument mit den Knotenpunkten des XML-Teils verbunden, was sie gleichzeitig ausfüllt. Falls es sich um eine Offerte handelt, wird eine Tabelle (die als Baustein in der Vorlage gespeichert ist) eingefügt, diese um die passende Anzahl Zeilen erweitert, und die darin enthaltenen Inhaltssteuerelemente mit den passenden Knotenpunkten des XML-Teils verbunden.

Als letzter Handlungsvorgang werden die Listen der Kombinationsfelder gefüllt und dann anschließend alle Inhaltssteuerelemente im Dokument gruppiert sowie geschützt. Die Listen für die Kombinationsfelder (für die Anrede und die Begrüßung) stehen ebenfalls in *xml*-Dateien bereit. Je nach ausgewählter Sprache werden englische oder deutsche Begriffe geladen.

Die letzte Spalte der Produkttabelle enthält Formularfelder, um die Berechnungen durchzuführen. Diese werden aktualisiert durch Auslösen eines Ereignisses beim Verlassen eines der Inhaltssteuerelemente in der Spalte *Anzahl*.

In den folgenden Abschnitten werden die Prozeduren, die Elemente des Objektmodells, die mit ContentControls und CustomXMLParts zu tun haben, erläutert.

#### HINWEIS

Aus Platzgründen haben wir die Prozeduren weggelassen, die das XML über das MSXML-DOM bearbeiten. Sie finden diese in den VBA-Modulen der Beispieldatei.



Die Beispieldatei *Bsp07\_03\_CC.dotm* mit den Begleitdateien *anreden.xml*, *grussformel.xml*, *brief.xml* sowie *brief\_offerte.xml* befinden sich auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap07*. Die Access-Datenbank *Nordwind.mdb* befindet sich im Ordner *\Datenbank*.

## Dem Dokument einen XML-Teil zufügen und die Inhaltssteuerelemente damit verbinden

Wie erwähnt, wird eine *xml*-Datei zusammengestellt, basierend auf den im Formular ausgewählten Optionen. Der XML-Teil für das Dokument in Abbildung 7.31 ist in Listing 7.23 wiedergegeben.

**TIPP**

Um das XML eines in einem Dokument gespeicherten XML-Teils einzusehen, führen Sie im Direktbereich des VBA-Editors die folgende Codezeile aus: `?ActiveDocument.CustomXMLParts(index).XML`, wobei Index dem Indexwert des XML-Teils entspricht.

Listing 7.23

Der XML-Code des im Dokument gespeicherten XML-Teils

```
<?xml version="1.0"?>
<bestellung>
  <absender ort="Seattle" name="Nancy Davolio" id="1"/>
  <kunde>
    <kunden-nr>ALFKI</kunden-nr>
    <firma>Alfreds Futterkiste</firma>
    <kontakt>Maria Anders</kontakt>
    <strasse>Obere Str. 57</strasse>
    <plz>12209</plz>
    <ort>Berlin</ort>
    <land>Deutschland</land>
  </kunde>
  <betreff/>
  <datum>28. September 2007</datum>
  <produkte>
    <produkt><id>1</id><name>Chai</name><preis>9,00</preis></produkt>
    <produkt><id>2</id><name>Chang</name><preis>9,50</preis></produkt>
    <produkt><id>3</id><name>Aniseed Syrup</name><preis>5,00</preis></produkt>
  </produkte>
</bestellung>
```

Die Prozedur *AllgemeineBriefInfo* in Listing 7.24 ruft eine Prozedur auf, die den XML-Code des Listings 7.23 zusammenstellt. Danach wird der XML-Teil (CustomXMLPart) im Dokument erstellt und der im Speicher erstellten XML-Zeichenkette zugewiesen:

```
Set cxp = doc.CustomXMLParts.Add
cxp.LoadXML sXML
```

Die restlichen Codezeilen, wovon hier nur ein Ausschnitt zu sehen ist, verbinden die Knotenpunkte des XML-Teils mit den passenden Inhaltssteuerelementen. Wie aus Listing 7.23 hervorgeht, sind die Angaben für den Absender in Attributen des XML-Elements `absender` enthalten, während die restlichen Informationen sich direkt in XML-Elementen befinden. In Office-DOM werden die Inhalte von Attributen und Elementen verschieden angesprochen; ein Attribut kann, anders als bei XSLT, über XPath nicht direkt angesprochen werden. Vergleichen Sie die folgenden zwei Codezeilen:

```
cc.XMLMapping.SetMappingByNode cxp.SelectSingleNode("//absender").Attributes(1)
cc.XMLMapping.SetMapping XPath="//datum", Source:=cxp
```

Für Elemente können Sie sowohl `SetMappingByNode` als auch `SetMapping` mit einer XPath-Zeichenkette verwenden. Für Attribute müssen Sie `SetMappingByNode` einsetzen, um auf den Inhalt zuzugreifen.

**Listing 7.24** Einen XML-Teil im Dokument erstellen und mit den Inhaltssteuerelementen verbinden

```
Private Sub AllgemeineBriefInfo(doc As Word.Document,
    ByRef cxp As Office.CustomXMLPart, frm As frmBsp07_03_CC)
    Dim xmlPfad As String
    Dim sXML As String
    Dim cc As Word.ContentControl

    xmlPfad = m_ProjectPath
    sXML = OfferteXMLErstellen(xmlPfad & "Brief_Offerte.xml", frm)
    If Len(sXML) = 0 Then
        'XML konnte nicht zusammengestellt werden
    End If

    Set cxp = doc.CustomXMLParts.Add
    cxp.LoadXML sXML
    For Each cc In doc.ContentControls
        Select Case cc.Title
            Case "Absenderort"
                cc.XMLMapping.SetMappingByNode cxp.SelectSingleNode("//absender").Attributes(1)
            Case "Datum"
                cc.XMLMapping.SetMapping XPath="//datum", Source:=cxp
        'Und so weiter für die restlichen Inhaltssteuerelemente
            Case Else
        End Select
    Next
End Sub
```

## Inhalt in ein Inhaltssteuerelement eingeben

Wenn Sie programmtechnisch mit einem Dokument arbeiten, dessen Inhaltssteuerelemente mit einem XML-Teil verbunden sind, wird der Inhalt der Inhaltssteuerelemente meistens über den XML-Teil gesteuert. Ausnahmen bilden Rich-Text- sowie Baustein-Inhaltssteuerelemente, die mit einem XML-Teil nicht verbunden werden können.

Um Inhalt in ein beliebiges, nicht geschütztes Inhaltssteuerelement zu schreiben, wird ganz einfach die Range-Eigenschaft des Inhaltssteuerelements angesprochen. In etwa:

```
Dim cc as Word.ContentControl
Set cc = ActiveDocument.ContentControls(1)
cc.Range.Text = "Text in einem Inhaltssteuerelement."
```

Im vorliegenden Beispiel wird ein Baustein mit der Produkttabelle in das Rich-Text-Inhaltssteuerelement eingefügt. Um das Steuerelement direkter ansprechen zu können, wurde ihm eine Textmarke zugewiesen.

```
Set rngProdukte = doc.Bookmarks("BriefInhalt").Range.ContentControls(1).Range
rngProdukte.InsertParagraph
rngProdukte.Collapse wdCollapseEnd
Set rngProdukte =
    doc.AttachedTemplate.BuildingBlockEntries("ProduktListe").Insert(rngProdukte, True)
Set tbl = rngProdukte.Tables(1)
```

Beim Betrachten des Beispielcodes und der Objektmodell-Hilfe zur Add-Methode der Auflistung ContentControls fällt auf, dass ein ContentControl-Objekt nur über den Indexwert direkt angesprochen werden kann. Eigentlich würden wir erwarten, dafür die Eigenschaft Titel verwenden zu dürfen. Es wurde jedoch entschieden, dass diese Eigenschaft ein Inhaltssteuerelement nicht eindeutig identifizieren würde. Sie wird stattdessen für die Widerspiegelung des Inhalts (siehe Abschnitt Datenverbindungen) eingesetzt. Um Inhaltssteuerelemente über die Eigenschaften Titel oder Tag anzusprechen, müssen die Methoden SelectContentControlsByTitle bzw. SelectContentControlsByTag eingesetzt werden. Diese geben eine Auflistung des Typs Office.CustomXMLNodes zurück. Das gesuchte Inhaltssteuerelement befindet sich dann in dieser Auflistung, wie im folgenden Abschnitt und in Listing 7.25 veranschaulicht.

**HINWEIS**

Als gültiger Indexwert für ein ContentControl-Objekt dient entweder die Reihenfolge des Inhaltssteuerelements im Dokument oder seine ID-Eigenschaft. Letztere wird von Word erstellt und kann nicht geändert werden.

## Liste eines Kombinationsfelds bzw. eine Dropdownliste füllen

In der Beispielvorlage kann der Benutzer »Anrede« und »Begrüßung« eingeben, oder Einträge aus den Kombinationsfeldlisten wählen. Da Nordwind seine Korrespondenz mehrsprachig führt, liegen sowohl deutsche als auch englische Floskeln in einer *xml*-Datei vor. Die Prozedur in Listing 7.25 lädt eine solche Datei und wählt das Element für die ausgewählte Sprache.

```
Set domNodes = domDoc.SelectNodes("//" & sprache & "/" & element)
```

Anschließend werden alle Inhaltssteuerelemente mit der gleichen Titel-Eigenschaft mittels der Methode SelectContentControlsByTitle durchschleift und den Inhalt der Knotenpunkte aus der *xml*-Datei der Kombinationsfeldliste zugefügt.

**Listing 7.25** Die Kombinationsfelderlisten mit Einträgen aus *xml*-Dateien bestücken

```
Private Sub ListeFüllen(doc As Word.Document, ccTitle As String, _
    element As String, datei As String, frm As frmBsp07_03_CC)
    Dim sprache As String
    Dim cc As Word.ContentControl
    Dim domDoc As MSXML2.DOMDocument
    Dim domNodes As MSXML2.IXMLDOMNodeList
    Dim domNode As MSXML2.IXMLDOMNode

    sprache = AusgewählteSprache(frm)
    Set domDoc = New MSXML2.DOMDocument
    domDoc.async = False
    domDoc.Load m_ProjectPath & datei
    If domDoc.parseError.ErrorCode <> 0 Then
        MsgBox domDoc.parseError.reason
        Exit Sub
    End If
    Set domNodes = domDoc.SelectNodes("//" & sprache & "/" & element)
    For Each cc In doc.SelectContentControlsByTitle(ccTitle)
        cc.DropDownListEntries.Clear
        For Each domNode In domNodes
            cc.DropDownListEntries.Add domNode.Text
        Next
    Next
```

**Listing 7.25** Die Kombinationsfelderlisten mit Einträgen aus *xml*-Dateien bestücken (Fortsetzung)

```
Exit For
Next
Set domNode = Nothing
Set domNodes = Nothing
Set domDoc = Nothing
End Sub
```

## Eine Liste wiederholter Inhaltssteuerelemente erstellen und sie mit einem XML-Teil verbinden (»1:n«)

In diesem Abschnitt wird das für Word berichtigte – weil schwer zu realisierende – Konzept »1:n« behandelt. Ob Seriendruck, Formulare oder Inhaltssteuerelemente, Word bietet kein Eigenmittel, solche Listen einfach zu erstellen.

Im vorliegenden Beispiel enthält die Offerte eine Produktliste in der Form einer Tabelle. Wie schon beschrieben, steht die Tabelle als Baustein in der Vorlage zur Verfügung. In ihrer Grundform enthält sie zwei Zeilen; in der zweiten befinden sich Inhaltssteuerelemente für die Artikel-Nr, Beschreibung, Anzahl und Einzelpreis. Es gilt, diese Zeile für alle ausgewählten Produkte zu duplizieren und die Inhaltssteuerelemente mit den korrekten Knotenpunkten des XML-Teils (siehe Listing 7.23) zu verbinden.

Eine Auflistung aller *produkt*-Elemente des XML-Teils wird erstellt und durchschleift.

```
Set nodesProdukte = cxpBrief.SelectNodes("//produkte/produkt")
For zähler = 1 To nodesProdukte.Count
```

Nur die Inhaltssteuerelemente im Bereich der zu bearbeitenden Tabellenzeile werden angesprochen. Da ihre Reihenfolge bekannt ist, werden Sie durch den Indexwert identifiziert.

```
rngProdukt.ContentControls(1).XMLMapping.SetMappingByNode _
nodesProdukte.Item(zähler).SingleNode("id")
```

Vor der Wiederholung der Schleife wird die »Muster«-Tabellenzeile (die zweite) kopiert und am Ende der Tabelle, für den nächsten Durchlauf, eingefügt.

Nachdem alle *produkt*-Elemente mit Inhaltssteuerelementen verbunden wurden, wird der Inhalt der letzten Tabellenzeile gelöscht und in der letzten Spalte eine Feldfunktion eingefügt, um die darüber liegenden Zahlen zusammenzuaddieren.

**Listing 7.26** Alle ausgewählten Produkte in der Produkttabelle auflisten

```
Private Sub OfferteSchreiben(frm As frmBsp07_03_CC, doc As Word.Document)
    Dim cxpBrief As Office.CustomXMLPart
    Dim rngProdukte As Word.Range
    Dim rngProdukt As Word.Range
    Dim tbl As Word.Table
    Dim nodesProdukte As Office.CustomXMLNodes
    Dim zähler As Long
    Dim rngFeld As Word.Range

    AllgemeineBriefInfo doc, cxpBrief, frm
```

Listing 7.26 Alle ausgewählten Produkte in der Produkttabelle auflisten (Fortsetzung)

```

'Die Tabelle für die offerierten Produkte erstellen und mit dem XML verbinden.
'Die Tabelle ist als Baustein in der Vorlage gespeichert.
Set rngProdukte = doc.Bookmarks("BriefInhalt").Range.ContentControls(1).Range
rngProdukte.InsertParagraph
rngProdukte.Collapse wdCollapseEnd
Set rngProdukte = doc.AttachedTemplate.BuildingBlockEntries( _
    "ProduktListe").Insert(rngProdukte, True)
Set tbl = rngProdukte.Tables(1)

'Durch die Produkt-Knotenpunkte schleifen
Set nodesProdukte = cxpBrief.SelectNodes("//produkte/produkt")
For zähler = 1 To nodesProdukte.Count
    Set rngProdukt = tbl.Rows(zähler + 1).Range
    rngProdukt.ContentControls(1).XMLMapping.SetMappingByNode _
        nodesProdukte.Item(zähler).SelectSingleNode("id")
    rngProdukt.ContentControls(2).XMLMapping.SetMappingByNode _
        nodesProdukte.Item(zähler).SelectSingleNode("name")
    rngProdukt.ContentControls(4).XMLMapping.SetMappingByNode _
        nodesProdukte.Item(zähler).SelectSingleNode("preis")
    'Die Tabellenzeile kopieren und einfügen
    rngProdukt.Copy
    rngProdukt.Collapse wdCollapseEnd
    rngProdukt.Paste
Next
'Den Inhalt der letzten Zeile löschen.
rngProdukt.Delete
'Eine Feldfunktion einfügen, um die Produktpreise zu addieren.
Set rngFeld = rngProdukt.Rows(1).Cells(rngProdukt.Rows(1).Cells.Count).Range
rngFeld.Collapse wdCollapseStart
rngProdukte.Fields.Add Range:=rngFeld,
    Text:="=Sum(above) \# " & Chr(34) & "0,00" & Chr(34), PreserveFormatting:=False
'Die Berechnungen aktualisieren
tbl.Range.Fields.Update
End Sub

```

## Ein neues Inhaltssteuerelement einfügen

Um programmäßig ein Inhaltssteuerelement in ein Dokument einzufügen, wird die Add-Methode der Auflistung ContentControls eingesetzt:

```

Dim ccGruppe as Word.ContentControl
Set ccGruppe = doc.ContentControls.Add(Type:=wdContentControlGroup, _
    Range:=Selection.Range)

```

## Steuerelemente gruppieren und schützen

Um mehrere Inhaltssteuerelemente zu gruppieren, wird ein Inhaltssteuerelement des Typs wdContentControlGroup eingefügt. Alle sich im Zielbereich (Argument Range) befindenden Inhaltssteuerelemente werden vom neuen Inhaltssteuerelement umfasst (siehe Abbildung 7.25).

Mit der Eigenschaft LockContentControl wird ein Inhaltssteuerelement vor dem Löschen geschützt.

```
ccGruppe.LockContentControl = False
If doc.Tables.Count > 0 Then
    doc.Tables(1).Range.Select
    Set ccGruppe = doc.ContentControls.Add(
        Type:=wdContentControlGroup, Range:=Selection.Range)
    ccGruppe.LockContentControl = True
End If
```

**HINWEIS** Mehr zum Thema lesen Sie im Internetbeitrag »Creating Word 2007 Templates Programmatically« unter <http://msdn2.microsoft.com/en-us/library/ms406053.aspx>.

## Ereignisse von Inhaltssteuerelementen

Alle Inhaltssteuerelemente teilen sich die gleichen sechs Ereignisse. Sie sind Ereignisse des Document-Objekts und müssen deshalb im Modul ThisDocument stehen. Die Hilfe-Themen dazu finden Sie in der Objektmodellreferenz für das Objekt Document.

- ContentControlAfterAdd(ByVal NewContentControl As ContentControl, ByVal InUndoRedo As Boolean)
- ContentControlBeforeContentUpdate(ByVal ContentControl As ContentControl, Content As String)
- ContentControlBeforeDelete(ByVal OldContentControl As ContentControl, ByVal InUndoRedo As Boolean)
- ContentControlBeforeStoreUpdate(ByVal ContentControl As ContentControl, Content As String)
- ContentControlOnEnter(ByVal ContentControl As ContentControl)
- ContentControlOnExit(ByVal ContentControl As ContentControl, Cancel As Boolean)

Alle Ereignisse stellen ein Argument des Typs ContentControl zur Verfügung, das den Zugriff auf das auslösende Inhaltssteuerelement ermöglicht.

Die Ereignisse, die beim Einfügen bzw. beim Löschen eines Inhaltssteuerelements ausgelöst werden, haben zudem ein Argument, das festhält, ob das Ereignis durch den Befehl *Rückgängig machen* oder *Wiederholen* ausgelöst wurde. Sie können bei Bedarf das Argument InUndoRedo prüfen und entsprechend handeln.

Die zwei Ereignisse ContentControlBeforeContentUpdate und ContentControlBeforeStoreUpdate werden nur ausgelöst, wenn das Steuerelement mit einem XML-Teil verbunden ist. Sie stellen ein Argument Content zur Verfügung, das den aktuellen Inhalt des Inhaltssteuerelements zurückgibt. ContentControlBeforeStoreUpdate wird beim Verlassen des Inhaltssteuerelement ausgelöst, wenn der Inhalt in den XML-Teil geschrieben wird. ContentControlBeforeContentUpdate wird ausgelöst, wenn der Text des mit dem Inhaltssteuerelement verbundenen Knotenpunkts (programmtechnisch) geändert wird.

Die letzten zwei Ereignisse werden beim Eintreten bzw. beim Verlassen des Inhaltssteuerelements ausgelöst. Das Argument Cancel des Ereignis ContentControlOnExit ermöglicht den Abbruch der Handlung. Damit kann der Benutzer dazu gezwungen werden, irgendeine Bedingung zu erfüllen, bevor er weitergehen darf. Das Listing 7.27 veranschaulicht dieses Ereignis.

**HINWEIS** ContentControlBeforeStoreUpdate findet logischerweise vor ContentControlOnExit statt.



Das gleiche Ereignis gilt für alle Inhaltssteuerelemente eines Dokuments. Listing 7.27 zeigt eine Möglichkeit auf, wie die Identität des auslösenden Inhaltssteuerelementes geprüft werden kann. In der Beispielvorlage interessieren nur die Inhaltssteuerelemente in der Produkttabellenspalte »Anzahl«, die alle »Anzahl« als Title-Eigenschaft haben. Der Inhalt muss numerisch sein. Ist das nicht der Fall, wird Cancel=True festgelegt und die Einfügemarke bleibt im Inhaltssteuerelement. Der Benutzer darf das Inhaltssteuerelement erst verlassen, wenn die Bedingung erfüllt ist. Sobald der Inhalt numerisch ist, werden alle Feldfunktionen der letzten Spalten aktualisiert und der Fokus verlässt das Inhaltssteuerelement.

Listing 7.27 Die Eingabe eines Inhaltssteuerelements in der Spalte *Anzahl* prüfen

```
Private Sub Document_ContentControlOnExit(ByVal ContentControl As ContentControl, _
    Cancel As Boolean)
    Select Case ContentControl.Title
        Case "Anzahl"
            If Not IsNumeric(ContentControl.Range.Text) Then
                MsgBox "Sie müssen eine Zahl eingeben"
                Cancel = True
            Else
                'Die Feldfunktionen in der Produkt-Tabelle aktualisieren
                Dim col As Word.Column
                Dim cel As Word.Cell

                Set col = ContentControl.Range.Tables(1).Columns(5)
                For Each cel In col.Cells
                    cel.Range.Fields.Update
                Next
            End If
        Case Else
            End Select
    End Sub
```

#### HINWEIS

Solange das Entwurfsmodus eingeschaltet ist, werden die Inhaltssteuerelementereignisse nicht ausgelöst.



Das Beispieldokument *Bsp07\_04\_CC.docm* enthält Ereignisstrukturen sowie einen XML-Teil, um das Verhalten der Inhaltssteuerelementereignisse zu veranschaulichen. Diese Datei befindet sich auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap07*.

## Zwischen Inhaltssteuerelementen springen

Benutzer fragen oft, wie sie mit Inhaltssteuerelementen effizient arbeiten können. Für gewöhnlich wird in einem Word-Formular mit der - oder der -Taste von einem Eingabefeld zum nächsten gesprungen. Mit der -Taste geht es auch in Dokumenten mit Inhaltssteuerelementen, so lange sie des Basistyps »Nur-Text« sind, die ein Tab-Zeichen standardmäßig nicht akzeptieren. Drücken der -Taste in Inhaltssteuerelementen des Typs »Baustein« oder »Rich-Text« fügt jedoch ein Tab-Zeichen ein.

#### HINWEIS

Um ein Tab-Zeichen in ein Inhaltssteuerelement des Typs »Nur-Text« einzufügen, drücken Sie, wie für eine Tabelle, + .

Auch das Navigieren mit der Maus gestaltet sich als problematisch, sobald die Platzhaltertexte nicht mehr sichtbar sind. Eine Lösung für den Benutzer, die ihm erlaubt, die Finger auf die Tastatur zu lassen, drängt sich auf. Folgend in Listing 7.28 ein Beispiel, wie Sie die Tastenkombinationen **F11** und **↵**+**F11**, die standardmäßig den internen Word-Befehlen *NächstesFeld* bzw. *VorherigesFeld* zugewiesen sind, für das Navigieren zwischen Inhaltssteuerelementen bereitstellen können.

Die Prozeduren *NextContentControl* und *PreviousContentControl* wurden im Beispieldokument den Tastenkombinationen **F11** bzw. **↵**+**F11** zugewiesen. Beide gehen auf gleiche Weise vor: die Anzahl Inhaltssteuerelemente bis zum Bereich der gegenwärtigen Markierung wird ermittelt. Falls dieses nicht das letzte (bzw. erste) Inhaltssteuerelement im Dokument ist, wird zum nächsten (bzw. vorherigen) Inhaltssteuerelement über den Indexwert gesprungen.

**Listing 7.28** Durch die Inhaltssteuerelemente in einem Dokument per Tastendruck schleifen

```
Sub NextContentControl() 'An die Tastenkombination F11 zuweisen
    Dim doc As Word.Document
    Dim rng As Word.Range
    Dim indexCC As Long
    Dim anzCC As Long

    Set doc = ActiveDocument
    anzCC = doc.ContentControls.Count
    If anzCC = 0 Then Exit Sub
    Set rng = Selection.Range
    rng.Start = doc.Content.Start
    indexCC = rng.ContentControls.Count

    If indexCC < anzCC Then
        doc.ContentControls(indexCC + 1).Range.Select
    Else
        doc.ContentControls(1).Range.Select
    End If
End Sub

Sub PreviousContentControl() 'An die Tastenkombination Umschalt+F11 zuweisen
    'Inhalt aus Platzgründen weggelassen, aber ähnlich wie NextContentControl
End Sub
```



Die oben dargestellten Beispiele finden Sie in der Beispieldatei *Bsp07\_05\_CC.docm* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap07*.

## Namensräume

Wenn Sie mit XML vertraut sind oder bereits das Kapitel 22 gelesen haben, sind Sie dem Begriff »Namespace« (Namensraum) wahrscheinlich schon begegnet. Kurz gefasst, ermöglicht ein Namensraum die eindeutige Zuordnung eines XML-Knotenpunktes. Die Knotenpunkte mehrerer XML-Dateien oder -Teilen können die gleiche Bezeichnung haben. Es ist wichtig, dazwischen unterscheiden zu können. Ferner können Knotenpunkte eines bestimmten Namensraums in einer größeren XML-Datei schneller und gezielter angesprochen werden.

Dieses einfache Beispiel veranschaulicht den Umgang mit Namensräumen in den XML-Teilen eines Word-Dokuments. Als Ausgangslage dient ein Dokument eines Personalvermittlungsbüros, das Vorschläge für einen Kunden auflistet (Abbildung 7.32). Name und Telefonnummer des Kunden,

sowie aller vorgeschlagenen Personen werden vom Benutzer in Inhaltssteuerelemente eingegeben. Diese sind mit einem XML-Teil verbunden, wie derjenige in Abbildung 7.33. Die gleichen Elementenamen »name« und »telefon« sind für »kunde« sowie »vermittelte« definiert. Es gilt, klar dazwischen zu unterscheiden, so dass beispielsweise mit wenig Aufwand alle vorgeschlagenen Namen gelesen werden können.

**Abbildg. 7.32** Die Inhaltssteuerelemente in den Tabellen sind mit verschiedenen Knotenpunkten gleichen Namens verbunden

IndiesemBeispielwerdenXML-Namensräume-veranschaulicht.¶

Kunde¶	Telefon¶
Die-große-Firma-in-München¶	666-9999-0000¶

¶

Vorschläge-für-vermitteltes-Personal¶	Telefon¶
Max-Müller¶	666-9999-1111¶
Doris-Schmid¶	666-9999-2222¶
Georg-Werner¶	666-9999-3333¶

¶

Hier klicken, um eine Zeile der Personaltabelle zuzufügen.¶

Hier klicken, um eine Liste der vorgeschlagenen Personen einzublenden.¶

Microsoft Word

Folgende Personen wurden vorgeschlagen:

Max Müller

Doris Schmid

Georg Werner

OK

**Abbildg. 7.33** Die XML-Struktur des XML-Teils mit Namensräumen

```
<?xml version="1.0"?>
<vorschlag xmlns="http://www.Kap07.com/Beispiel07_06_cc/vorschlag">
  <kunde xmlns="http://www.Kap07.com/Beispiel07_06_cc/kunde">
    <name/>
    <telefon/>
  </kunde>
  <vermittelte xmlns="http://www.Kap07.com/Beispiel07_06_cc/vermittelte">
    <person>
      <name/>
      <telefon/>
    </person>
  </vermittelte>
</vorschlag>
```

Die Beispielvorlage enthält die zwei Tabellen, mit je zwei Zeilen, sowie vier Inhaltssteuerelemente. Während der Vorbereitung wurden die Inhaltssteuerelemente über die Prozedur in Listing 7.29 mit dem XML-Teil verbunden. Der Benutzer erstellt ein auf dieser Vorlage basierendes Dokument und füllt die Listen aus.

Enthält ein XML-Teil einen oder mehrere Namensräume, erstellt Office dafür automatisch je ein Namensraumpräfix; bereits im XML-Code vorhandene Namensraumpräfixe werden *nicht* übernommen. Wird im Programmiercode ein Knotenpunkt angesprochen, der einem Namensraum

zugeordnet ist, muss die Angabe zwingend mit dem passenden Präfix voll qualifiziert werden (beispielsweise, mit »ns0:element« statt »element«). Dieser Umstand bedeutet, das Namensraumpräfix muss dynamisch durch den Code ermittelt werden können. Im Objektmodell wird dies durch die Methode `NamespaceManager.LookupPrefix` des Objekts `CustomXMLPart` bewerkstelligt.

```
pfxKunde = cxpVorschlag.NamespaceManager.LookupPrefix(nsKunde)
```

#### HINWEIS

Bei genauerem Betrachten der Objektvariablendeklarationen fällt auf, dass das Objekt `CustomXMLPart`, mit allen seinen Methoden und Eigenschaften, nicht Teil des Word- sondern des *Office*-Objektmodells ist. Sie würden diese Begriffe vergeblich in der Referenzbibliothek von Word suchen. Viele dieser Eigenschaften und Methoden bauen auf den MSXML Parser und werden Ihnen bekannt vorkommen, wenn Sie mit dem XML DOM schon gearbeitet haben.

Die letzten vier Befehlszeilen in Listing 7.29 veranschaulichen, wie mit Hilfe des Namensraumpräfixes die Knotenpunkte »name« und »telefon«, unter »kunde« bzw. »vermittelte«, direkt angesprochen werden, um sie mit den passenden Inhaltssteuerelementen zu verbinden. Die Alternative wäre, in jeder Befehlszeile den ganzen XPath auszuschreiben.

#### HINWEIS

Das Listing 7.29 veranschaulicht zudem den Gebrauch der ID-Eigenschaft als Indexwert für die Identifikation eines Inhaltssteuerelementes. Bitte beachten Sie, dass diese als Datentyp Zeichenkette anzugeben ist.

Listing 7.29

Inhaltssteuerelemente mit Knotenpunkten verbinden, die einem Namensraum zugeordnet sind

```
Private Const nsKunde As String =
    "http://www.Kap07.com/Beispiel07_06_cc/Kunde"
Public Const nsVermittelte As String =
    "http://www.Kap07.com/Beispiel07_06_cc/Vermittelte"
Private Const nsVorschlag As String =
    "http://www.Kap07.com/Beispiel07_06_cc/Vorschlag"

'Muss nur einmal, in der Vorlage, ausgeführt werden
Sub CC_XML_Laden()
    Dim doc As Word.Document
    Dim xmlVorschlag As String
    Dim cxpVorschlag As Office.CustomXMLPart
    Dim pfxKunde As String, pfxVermittelte As String

    Set doc = ActiveDocument
    xmlVorschlag = "<?xml version=" & Chr(34) & "1.0" & Chr(34) & "?><vorschlag xmlns=" & Chr(34) & nsVorschlag & Chr(34) & "><kunde xmlns=" & Chr(34) & nsKunde & Chr(34) & _
        & "><name /><telefon /></kunde><vermittelte xmlns=" & Chr(34) & nsVermittelte & Chr(34) & "><person><name /><telefon /></person></vermittelte></vorschlag>"
    Set cxpVorschlag = doc.CustomXMLParts.Add(xmlVorschlag)

    pfxKunde = cxpVorschlag.NamespaceManager.LookupPrefix(nsKunde)
    pfxVermittelte = cxpVorschlag.NamespaceManager.LookupPrefix(nsVermittelte)

    doc.ContentControls("58468977").XMLMapping.SetMapping _
        XPath:="//" & pfxKunde & ":name", Source:=cxpVorschlag 'Kundenname
    doc.ContentControls("58468980").XMLMapping.SetMapping _
```

**Listing 7.29** Inhaltssteuerelemente mit Knotenpunkten verbinden, die einem Namensraum zugeordnet sind (Fortsetzung)

```

XPath="/" & pfxKunde & ":telefon", Source=cxpVorschlag      'KundenTelefon
doc.ContentControls("58468981").XMLMapping.SetMapping XPath="/" & _
pfxVermittelte & ":name", Source=cxpVorschlag              'Name des 1. Vermittelten
doc.ContentControls("58468982").XMLMapping.SetMapping XPath="/" & _
pfxVermittelte & ":telefon", Source=cxpVorschlag          'Telefon des ersten Vermittelten
End Sub

```

Um der zweiten Tabelle eine neue Zeile, samt Inhaltssteuerelemente, hinzuzufügen, klickt der Benutzer auf die *MacroButton*-Feldfunktion unterhalb der Tabelle. Dadurch wird die Prozedur *NeuesPersonal* in Listing 7.30 angestoßen. Nachdem die neue Tabellenzeile eingefügt wurde, wird die Prozedur *XmlPersonalNodeErstellen* aufgerufen, die dem XML-Teil einen Satz »personal«-Knotenpunkte unter dem Element »vermittelte« zufügt.

Es ist wichtig, diese Handlung im korrekten XML-Teil vorzunehmen. Um diesen zu ermitteln, wird durch alle *CustomXMLParts* des Dokuments geschleift und der Namensraum geprüft. Liegt der korrekte vor, wird das Namensraumpräfix für den Knotenpunkt »vermittelte« nachgeschlagen und anschließend der Wurzelknotenpunkt ermittelt. Durch die Methode *AppendChildSubtree* wird dieser mit den neuen Knotenpunkten für eine Person ergänzt.

Um diese mit den neuen Inhaltssteuerelementen zu verbinden, müssen die individuellen Knotenpunkte für »name« und »telefon« bereitstehen. Da nun mehrere Knotenpunkte für »person« vorhanden sind, muss der Indexwert des letzten (des neu erstellten) ermittelt werden. Erst dann liegen alle benötigten Informationen für den XPath vor:

```

anzPersonNodes = rootNode.SelectNodes("/" & ns & ":person").Count
Set nodePerson = rootNode.SelectSingleNode("/" & ns & ":person[" & anzPersonNodes & "]")
Set nodeName = nodePerson.SelectSingleNode("./" & ns & ":name[1]")
Set nodeTelefon = nodePerson.SelectSingleNode("./" & ns & ":telefon[1]")

```

**Listing 7.30** Eine neue Tabellenzeile mit Inhaltssteuerelementen einfügen, den XML-Teil ergänzen und mit den Inhaltssteuerelementen verbinden

```

Public Sub NeuesPersonal()
    Dim doc As Word.Document
    Dim tbl As Word.Table
    Dim rw As Word.Row
    Dim ccName As Word.ContentControl
    Dim ccTelefon As Word.ContentControl
    Dim nodePerson As Office.CustomXMLNode
    Dim nodeName As Office.CustomXMLNode
    Dim nodeTelefon As Office.CustomXMLNode

    Set doc = ActiveDocument
    Set tbl = doc.Tables(2)
    Set rw = tbl.Rows.Add

    Set nodePerson = XmlPersonalNodeErstellen(doc, nodeName, nodeTelefon)

    Set ccName = InhaltssteuerelementErstellen(rw, "Name", 1)
    ccName.XMLMapping.SetMappingByNode nodeName

```

**Listing 7.30** Eine neue Tabellenzeile mit Inhaltssteuerelementen einfügen, den XML-Teil ergänzen und mit den Inhaltssteuerelementen verbinden (*Fortsetzung*)

```

Set ccTelefon = InhaltssteuerelementErstellen(rw, "Telefon", 2)
ccTelefon.XMLMapping.SetMappingByNode nodeTelefon
End Sub

Private Function XmlPersonalNodeErstellen(doc As Word.Document, ByRef nodeName, _
ByRef nodeTelefon) As Office.CustomXMLNode
Dim cpx As Office.CustomXMLPart
Dim nodePerson As Office.CustomXMLNode
Dim rootNode As Office.CustomXMLNode
Dim ns As String
Dim anzPersonNodes As Long

For Each cpx In doc.CustomXMLParts
If cpx.NamespaceURI = nsVorschlag Then
ns = cpx.NamespaceManager.LookupPrefix(nsVermittelte)
Set rootNode = cpx.SelectSingleNode("//" & ns & ":vermittelte")
rootNode.AppendChildSubtree "<person xmlns=" & Chr(34) & nsVermittelte & _
Chr(34) & "><name/><telefon/></person>"
anzPersonNodes = rootNode.SelectNodes("//" & ns & ":person").Count

Set nodePerson = rootNode.SelectSingleNode(
"//" & ns & ":person[" & anzPersonNodes & "]" & ")
Set nodeName = nodePerson.SelectSingleNode("./" & ns & ":name[1]")
Set nodeTelefon = nodePerson.SelectSingleNode("./" & ns & ":telefon[1]")
Exit For
End If
Next
Set XmlPersonalNodeErstellen = nodePerson
End Function

Private Function InhaltssteuerelementErstellen(rw As Word.Row, _
sTitle As String, index As Long) As Word.ContentControl

Dim rng As Word.Range
Set rng = rw.Cells(index).Range
rng.Collapse wdCollapseStart
Set InhaltssteuerelementErstellen = rng.ContentControls.Add(wdContentControlText, rng)
End Function

```

Das Listing 7.31 veranschaulicht nochmals, wie, dank des Namensraums, alle Knotenpunkte »name« für vermittelte Personen direkt angesprochen werden können. Diese Prozedur wird durch die zweite *MacroButton*-Feldfunktion ausgelöst und blendet eine Nachricht mit der Liste der Namen in der zweiten Tabelle ein.

**Listing 7.31** Alle Namen in der zweiten Tabelle direkt aus dem XML-Teil lesen

```

Sub NamenAuflisten()
Dim ns As String
Dim cpx As Office.CustomXMLPart
Dim xmlNodes As Office.CustomXMLNodes
Dim node As Office.CustomXMLNode
Dim s As String

```

Listing 7.31 Alle Namen in der zweiten Tabelle direkt aus dem XML-Teil lesen (Fortsetzung)

```

For Each cpx In ActiveDocument.CustomXMLParts
    If cpx.NamespaceURI = nsVorschlag Then
        ns = cpx.NamespaceManager.LookupPrefix(nsVermittelte)
        Exit For
    End If
Next

Set xmlNodes = cpx.SelectNodes("//" & ns & ":name")
For Each node In xmlNodes
    s = s & node.Text & vbCrLf
Next
MsgBox "Folgende Personen wurden vorgeschlagen:" & vbCrLf & s
End Sub

```



Die Beispieldatei *Bsp07\_06\_CC.docm* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap07*.

## 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 OpenXML-Dateiformat von Word 2007 gespeichertes Dokument mit XML-Werkzeugen zu bearbeiten. Dank des Kompatibilitätpacks können alle Versionen von Word, bis einschließlich der Version 2000, solche Dokumente öffnen, bearbeiten und speichern. Besonders, wenn mit den neuen Inhaltssteuerelementen, wie im Abschnitt »Inhaltssteuerelemente im Objektmodell« beschrieben, gezielt gearbeitet werden kann, soll diese Option ernsthaft geprüft werden.

**HINWEIS** Wie Seriendruckfelder in Inhaltssteuerelemente konvertiert werden können, beschreibt der Artikel »Migrating Mail Merge Fields to Content Controls« auf [http://blogs.msdn.com/microsoft\\_office\\_word/archive/2007/03/28/migrating-mail-merge-fields-to-content-controls.aspx](http://blogs.msdn.com/microsoft_office_word/archive/2007/03/28/migrating-mail-merge-fields-to-content-controls.aspx).

Eine relativ schnörkellose Methode ist, Daten automatisiert in das Dokument einzufügen, wie im Abschnitt »Zielscheibe Textmarke: Das *Bookmark*-Objekt« zu Beginn dieses Kapitels oder in Kapitel 10 im Abschnitt über ADO erklärt.

Der Seriendruck jedoch bietet seit frühesten Versionen eine dem Anwender vertraute Schnittstelle an, womit mit Word-eigenen Mitteln 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. Seither wurde nichts Grundlegendes verändert. Die wichtigsten Änderungen in Word 2002 waren:

- Zu den unterstützten Verbindungsmethoden DDE und ODBC gesellte sich OLE DB.
- Neue Sicherheitsmaßnahmen erschwerten das Öffnen von Seriendruck-Hauptdokumenten.
- Der eingeführte Aufgabenbereich ließ sich teilweise steuern.
- Neue »MailMerge«-Ereignisse ermöglichten einen Eingriff in das Zusammenführen des Seriendrucks (siehe Kapitel 8).

In diesem Abschnitt werden die wichtigsten Aspekte der Automatisierung vorgestellt.

### Der Makrorekorder

Der Makrorekorder leistet beim Erforschen der Seriendruckfunktionalität nur begrenzt Hilfe. Er ist vor allem beim Erkunden der Verbindungszeichenkette nützlich. Am Ende dieses Abschnitts finden Sie eine Diskussion über Verbindungsmethoden.

### Seriendruck-Hauptdokument



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.

### Datensätze ansprechen

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 ausgewählt 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 7.6 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 7.6 Konstantwerte für die Datensatz-Navigation

<code>WdMailMergeActiveRecord-Enum</code>	Wert	Beschreibung
<code>wdNoActiveRecord</code>	-1	Kein 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 Konstantwerte 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 und früher gibt es die `RecordCount`-Eigenschaft nicht. In dieser Version muss der letzte Datensatz ausgewählt und `ActiveRecord` abgefragt werden.

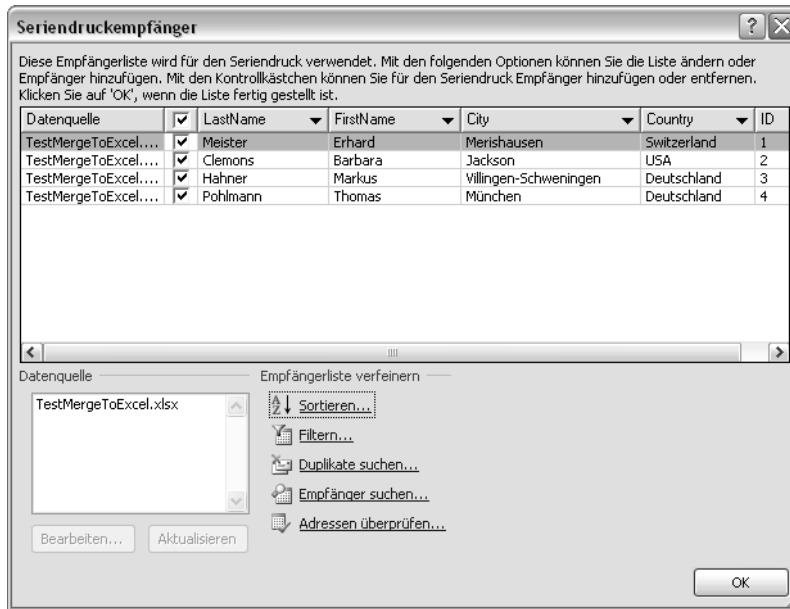
#### HINWEIS

Das Navigieren der Datensätze wird im Beispiel zu den Seriendruckereignissen in Kapitel 8 veranschaulicht.



Mit Einführung des Dialogfeldes *Seriendruckempfänger* in Word 2002 erhielt der Anwender mehr Flexibilität, die in Word 2007 erweitert wurde (Abbildung 7.34). Dadurch wurde jedoch die Navigation und Auswertung von Datensätzen durch das Objektmodell komplizierter.

Abbildg. 7.34 Das Dialogfeld *Seriendruckempfänger* ermöglicht es dem Anwender, einzelne Datensätze vom Seriendruck auszuschließen



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 7.32 und Abbildung 7.35 ersichtlich.



Bestimmte Funktionalitäten, die in früheren Versionen allen Datenquellen zur Verfügung standen, werden in Word 2007 nur bereitgestellt, wenn die Datenquelle über die Verbindungsmethode OLE DB eingebunden wurde. Beispielsweise steht das Dialogfeld in Abbildung 7.35 bei einer ODBC-Verbindung zu einer Access 2003-Tabelle nicht zur Verfügung. Auch die Methode RecordCount löst unter diesem Umstand eine Fehlermeldung aus, so dass der Entwickler auf die ältere Methode zurückgreifen muss, um die Anzahl Datensätze zu ermitteln.

Listing 7.32 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.MailMerge.DataSource

    If ActiveDocument.MailMerge.MainDocumentType = wdNotAMergeDocument _
        Then Exit Sub
    Set ds = ActiveDocument.MailMerge.DataSource
    lAnzahlDS = ds.RecordCount
    lAnzahlAktiverDS = 0
```

Listing 7.32 Durch alle Datensätze Schleifen und feststellen, ob sie ausgeschlossen sind (Fortsetzung)

```

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. 7.35 Das Ergebnis von Listing 7.32 basiert auf den Einstellungen in Abbildung 7.34

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.

Um auf die Datenfelder zuzugreifen, stehen die DataFields- sowie Fieldnames-Auflistungen bereit. Der Datenfeldinhalt kann nur gelesen und nicht beschrieben werden.

**TIPP**

Sie können mit der Included-Eigenschaft sowie der SetAllIncludedFlags-Methode Datensätze aus- und einschließen, analog zum Dialogfeld.



Die Beispieldatei *Bsp07\_01\_SD.doc* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap07*. Sie sollen sie mit der Tabelle *Personal* in der *Nordwind.mdb*-Datenbank im *\Datenbank*-Ordner verbinden.

**Datensätze filtern**

Das *Empfänger*-Dialogfeld, bzw. die Included-Eigenschaft, ist eine Möglichkeit, ab Word 2002 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 und früher ist diese die einzige Methode, um Datensätze auszuschließen.

**HINWEIS** In Word 2007 klicken Sie auf den Link *Filtern* im Dialogfeld *Seriendruckempfänger*. In Word 2002 und 2003 wird diese Funktionalität in der Benutzerschnittstelle 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` festgelegte Argumente `SQLStatement` und `SQLStatement1` (siehe »Wie *OpenDataSource* funktioniert«). Die SQL-Anweisung der `QueryString`-Eigenschaft darf höchstens 256 Zeichen betragen.

**HINWEIS** 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 angesprochen werden.

Um alle Datensätze der Datenquelle wieder verfügbar zu machen:

```
doc.MailMerge.DataSource.QueryString = "SELECT * FROM [Personal]"
```

### Datensatz 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 7.33 funktioniert in allen Versionen von Word, nur muss die richtige Methode verwendet werden.



2007

Word 2007 ist zum früheren Verhalten von FindRecord in der Version 2000 zurückgekehrt.

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 7.33

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 von der Prozedur *DatensatzSuchen* eingeblendet wird. Die Version von Word wird ermittelt und die Informationen an die passende Funktion übergeben.



Die Beispieldatei *Bsp07\_01\_SD.doc* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap07*. Sie sollen sie mit der Tabelle *Personal* in der *Nordwind.mdb*-Datenbank im *\Datenbank*-Ordner verbinden.

**Seriendruckdokument öffnen**

Eigentlich lässt sich ein Seriendruckdokument wie jedes andere Dokument öffnen. Etwas umständlicher wurde dies jedoch ab Word 2002 (mit Service Pack) 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 zufrieden stellend. Das Verhalten kann durch Festlegung eines Registry-Eintrags ausgeschaltet werden. Dieser Vorgang ist im Knowledge Base-Artikel » You receive the "Opening this will run the following SQL command" message when you open a Word mail merge main document that is linked to a data source« unter <http://support.microsoft.com/kb/825765/en-us> beschrieben.

Noch eine Änderung gibt es seit Word 2002: Es wird keine Seriendruckschnittstelle – weder die Symbolleiste, der Assistent *Aufgabenbereich*, noch die Registerkarte *Sendungen* in Word 2007 – beim Öffnen eines Seriendruck-Hauptdokuments eingeblendet. Wenn Sie dem Anwender eine Schnittstelle anbieten wollen, können Sie das mit einem AutoOpen- oder Document\_Open-Makro tun, wie in Listing 7.34.

### Der Seriendruck-Assistent

Im Allgemeinen stellt VBA keine Schnittstelle für die Änderung oder Anpassung der Aufgabenbereiche zur Verfügung. Bekanntlich bestätigt die Ausnahme die Regel: der Seriendruck-Assistent darf begrenzt angepasst werden.



Bekanntlich ersetzt in Word 2007 die Multifunktionsleiste die altbekannten Menü- und Symbolleisten früherer Versionen. Der Seriendruck wird dort über die Registerkarte *Sendungen* gesteuert. Der Seriendruck-Assistent steht weiterhin zur Verfügung. Der Befehl befindet sich im Dropdownmenü der Schaltfläche *Seriendruck starten* in der gleichnamigen Gruppe.

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.

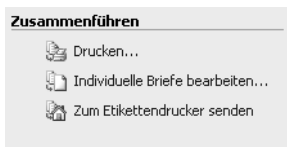
Der Inhalt des Aufgabenbereichs jedoch kann nur im sechsten und letzten Schritt beeinflusst werden: die Methode ShowSendToCustom ermöglicht die Hinzufügung eines zusätzlichen, vom Entwickler definierten Eintrags. Und das Ereignis MailMergeWizardSendToCustom wird beim Anklicken des Eintrags ausgeführt, 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-Symbolleiste 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 7.36 hinzugefügt, den der Benutzer auswählen soll, um den Seriendruck zum richtigen Drucker zu senden.

**Abbildg. 7.36** 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 7.34 sowohl vom `Document_New`- als auch vom `Document_Open`-Ereignis aufgerufen.

**Listing 7.34** Den Seriendruck-Assistenten 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
        'Hat in Word 2007 keine Wirkung. Das Ribbon (Multifunktionsleiste) der
        'Vorlage muss, wie in Kapitel 17 beschrieben, angepasst werden, um die
        'Registerkarte oder darin enthaltene Befehle zu sperren.
        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 *Bsp07\_02\_SD.dot* finden Sie auf der CD-ROM zum Buch im Ordner `\Beispiele\Kap07`. Sie ist mit der Tabelle *Personal* in der *Nordwind.mdb*-Datenbank im `\Datenbank`-Ordner verbunden.

### Seriendruck zusammenführen

Ein Seriendruck kann in ein neues Dokument, direkt zum Drucker, zum Fax- oder zum E-Mail-Versand zusammengeführt werden (vorausgesetzt, das Ausgabezielgerät ist vorhanden und verfügbar). 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 7.35.



## 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 ein im Seriendruck-Hauptdokument gespeichertes Dokument-Variable-Objekt (nicht mit einer VBA-Variablen zu verwechseln). Das Resultat »erbt« diese. Somit kann ihr Vorhandensein getestet werden.

Wird ab Word 2002 automatisiert, besteht die Möglichkeit, mit Seriendruckereignissen zu arbeiten. Das `MailMergeAfterMerge`-Ereignis stellt ein entsprechendes `Document`-Objekt zur Verfügung. Mehr über Seriendruckereignisse erfahren Sie im folgenden Kapitel 8.

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 in den Word-Versionen vor 2007 zu diesem Zeitpunkt jedoch nicht vollständig. Falls der Anwender das Resultat mit Makros bearbeiten soll, muss nach dem Zusammenführen die Verknüpfung zur Vorlage explizit hergestellt werden:

```
ActiveDocument.AttachedTemplate = docSD_Hauptdokument.AttachedTemplate
```



2007

Werden die Makros über die Multifunktionsleiste aufgerufen, entsteht das beschriebene Problem nicht. Makros in Symbolleisten, die in der angehängten Vorlage gespeichert sind, stehen dem Seriendruckergebnis nicht zur Verfügung.

### Hauptdokument anlegen

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.

### Umschläge

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.

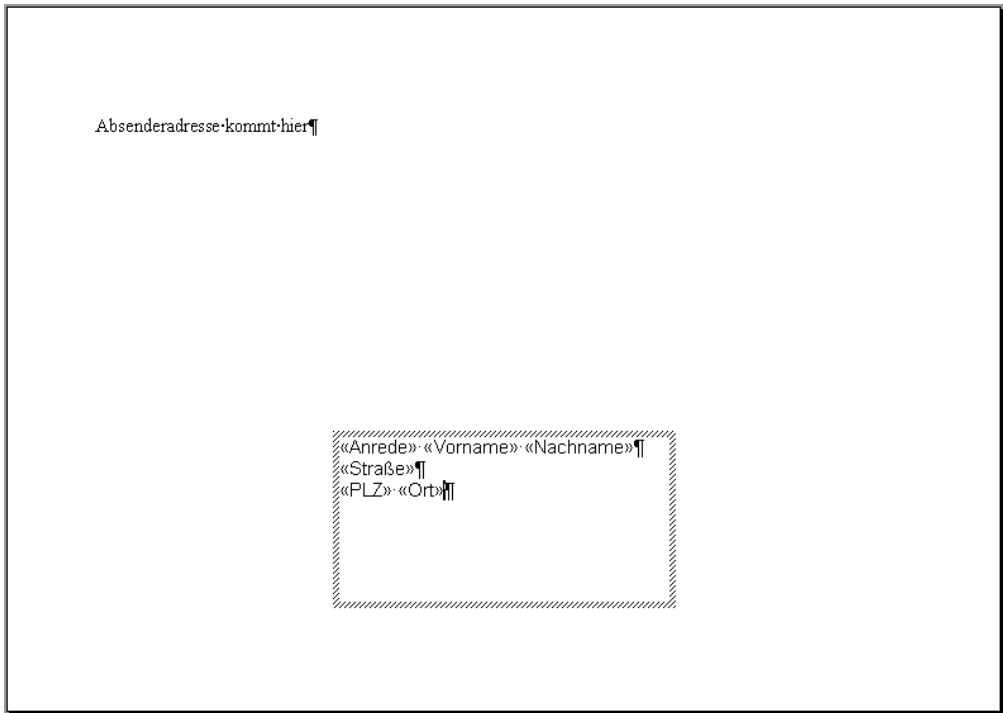
Der Entwickler muss alles selbst erledigen, und der Makrorekorder leistet bei der Suche nach den nötigen Befehlen keinen großen Dienst. Er nimmt lediglich die Festlegung der Art des Seriendruck-Hauptdokuments auf, aber nicht das Umschlagformat. Das Listing 7.35 zeigt, was Sie alles so benötigen, um einen 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 7.37 veranschaulicht.

Abbildg. 7.37 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 7.35 eingefügt.

Listing 7.35 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
    doc.PageSetup.PaperSize = wdPaperEnvelopeC5
    doc.PageSetup.Orientation = wdOrientLandscape

    Selection.Range.Text = "Absenderadresse kommt hier"
```

**Listing 7.35** Einen Umschlag als Seriendruck-Hauptdokument erstellen (Fortsetzung)

```

Set rng = doc.Range
rng.InsertAfter vbCrLf
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 = vbCrLf
    FeldEinfuegen "Straße", rng
    rng.Text = vbCrLf
    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

```



Die Beispieldatei *Bsp07\_03\_SD.doc* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap07*. Sie sollen sie mit der Tabelle *Personal* der Datenbank *Nordwind.mdb* im Ordner *\Datenbank* verbinden.

## Etiketten

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 7.36 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.

In Word 2007 kann der Befehl über `Application.CommandBars.ExecuteMso` ausgeführt werden. Die Mso-Bezeichnung lautet: `MailMergeUpdateLabels`.



2007

**HINWEIS** Dieses Beispiel ist für Word 2002 und später 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 7.36 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 >>" & "\1 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
        Select Case Val(Application.Version)
            Case 12
                'In Word 2007 wird der Benutzer Dialogfelder quittieren müssen
                WordBasic.MailMergeUseOutlookContacts
            Case Else
                .OpenDataSource Name:="Kontakte", SQLStatement:=strSQL, _
                    SubType:=wdMergeSubTypeOutlook
        End Select
        '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 *Bsp07\_04\_SD.doc* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap07*.

## Datenquelle 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 7.7 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 7.7

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 <code>wdOpenFormatAuto</code> 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

**Tabelle 7.7** Parameter der Methode *OpenDataSource* (Fortsetzung)

Parameter-Name	Bemerkungen
<b>WritePasswordDocument</b>	<p>Optionaler Variant-Wert. Falls ein Word-Dokument mit einem Kennwort zum Ändern gespeichert wurde, das Kennwort hier übergeben, um das Datenquellen-Dokument zu öffnen.</p> <p>Fehlt der Parameter oder wird nur eine leere Zeichenkette übergeben, blendet Word eine Eingabeaufforderung ein.</p> <p>Wird das falsche Kennwort übergeben, schlägt die <b>OpenDataSource</b>-Methode fehl.</p>
<b>WritePasswordTemplate</b>	* Optionaler Variant-Wert
<b>Connection</b>	<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>
<b>SQLStatement</b>	Optionaler Variant-Wert. Bestimmt eine Abfrage, um die Daten zu filtern.
<b>SQLStatement1</b>	Optionaler Variant-Wert. Falls die SQL-Anweisung länger als 255 Zeichen ist, kann sie mit diesem Parameter um zusätzliche 255 Zeichen erweitert werden.
<b>OpenExclusive</b> (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.
<b>SubType</b> (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>wdMergeSubType0AL</b></p> <p><b>wdMergeSubType0LEDBText</b></p> <p><b>wdMergeSubType0LEDBWord</b></p> <p><b>wdMergeSubType0ther</b></p> <p><b>wdMergeSubTypeOutlook</b></p> <p><b>wdMergeSubTypeWord</b></p> <p><b>wdMergeSubTypeWord2000</b></p> <p><b>wdMergeSubTypeWorks</b></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.	

## 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. Ab Word 2002 ebenfalls, wenn SubType:=wdMergeSubType2000.
- Für alle anderen Datenbankarten wird Word mangels ODBC DSN versuchen, in Word 97 und 2000, sowie ab Word 2002 bei SubType:=wdMergeSubType2000, die Daten über einen Konvertierfilter bereitzustellen. (Dies wird voraussichtlich nur mit Word-Dokumenten, zeichengetrennten Textdateien und Tabellenkalkulationsblättern gelingen, wobei der Tabellenblattkonvertierfilter, der nicht mehr im Lieferumfang von Office 2007 enthalten ist, vorhanden sein und registriert sein muss.)
- Ab Word 2002 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 und später. 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.

## Einige Beispiel-Datenverbindungen

Wegen der Vielfalt an Datenverbindungen und des mangelnden Platzes werden hier nur einige allgemeine Verbindungsbeispiele als »Wegweiser« vorgestellt. Der Text zu diesem Abschnitt aus der ersten Auflage, mit eingehenden Erläuterungen und Beispielen, finden Sie in der Datei *SD\_Verbindungen.zip* auf der CD-ROM zum Buch im Ordner \Beilagen. Für Word 2007 empfehlen wir die Angaben auf der Webseite von Peter Jamieson unter <http://tips.pjmsn.me.uk/t2007.htm>.

### Word-Dokument sowie RTF- und HTM-Dateien

Um zu einem Word-Dokument, einer RTF- oder HTML-Datei mit Tabelle eine Verbindung herzustellen:

```
ActiveDocument.MailMerge.OpenDataSource Name:="c:\Datenkbank\bestellungen-tab-1-fn.doc"
```

Word bedient sich dabei eines Konvertierfilters, um die Nicht-Word-Dateien zu öffnen.

### Textdateien – einfachste Version über einen Konvertierfilter

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.

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:\Datenbank\bestellungen-tab-1-fn.txt", _
    SubType:=wdMergeSubTypeOther
```

Um auch eine Steuersatzdatei einzubinden:

```
ActiveDocument.MailMerge.OpenHeaderSource _
    Name:="c:\Datenbank\bestellungen-tab-1-nur-fn.txt"
ActiveDocument.MailMerge.OpenDataSource Name:="c:\Datenbank\bestellungen-tab-1.txt"
```

### Microsoft Access

#### Über DDE

DDE war bis zur Einführung von OLE DB die standardmäßige Verbindungsmethode. Sie setzt voraus, dass die Anwendung lokal installiert und ausgeführt werden kann. Nur mit dieser Verbindungsmethode werden die Formatierungen von Zahlen und Datumsangaben in das Seriendruckergebnis übernommen. Unter Umständen steht die Verbindungsmethode auf einzelnen Installationen nicht zur Verfügung.

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 ab Word 2002 eine solche Verbindung aufzubauen, brauchen Sie die `SubType`-Parameter.



### Einfache Verbindung zu einer Tabelle, ohne SQL-Anweisung

```
ActiveDocument.MailMerge.OpenDataSource Name="C:\Datenbank\Nordwind.mdb", _
Connection:"TABLE Bestellungen", Subtype:= wdMergeSubtypeWord2000
```

### Einfache Verbindung zu einer Abfrage, ohne SQL-Anweisung

```
ActiveDocument.MailMerge.OpenDataSource Name="C:\Datenbank\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:\Datenbank\Nordwind.mdb", _
SQLStatement:"SELECT [Bestell-Nr], [Kunden-Code], Straße " & "FROM Bestellungen"
```

### Verbindung zu einer Abfrage, mit SQL-Anweisung

```
ActiveDocument.MailMerge.OpenDataSource Name="C:\Datenbank\Nordwind.mdb", _
SQLStatement:"SELECT EinzelPreis FROM [Die zehn teuersten Artikel]"
```

### Über ODBC

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*.

Ab Word 2002 ist die Methode mit Subtype:= wdMergeSubTypeWord2000 zu ergänzen.

```
ActiveDocument.MailMerge.OpenDataSource Name="C:\Datenbank\Nordwind.mdb", _
Connection:"DSN=Microsoft Access-Datenbank;" & _
"DBQ=C:\Datenbank\Nordwind.mdb;" & _
"DriverId=25;FIL=MS Access;MaxBufferSize=2048;PageTimeout=5;", _
SQLStatement:"SELECT * FROM 'Bestellungen' WHERE 'Kunden-Code'='CHOPS'", _
SQLStatement1:""
```

### Über OLE DB

#### Verbindung zu einer geschützten .mdb-Datei (Access) über eine leere .odc-Datei

Der Seriendruck in Word erwartet von einer OLE DB-Verbindung zusätzliche Informationen in Form einer \*.odc-Datei. Diese ist eine reine Textdatei, die Informationen in einem XML-Format enthält. Für einfache Verbindungen kann diese Datei leer sein, es ist aber vorteilhaft, wenn sie zumindest existiert und an dem erwarteten Ort zu finden ist.

```
ActiveDocument.MailMerge.OpenDataSource Name="c:\wbdb\leer.odc", _
Connection:"Provider=Microsoft.Jet.OLEDB.4.0;Password=sdkw;" & _
"User ID=wb;Data Source=c:\wbdb\artikelsd.mdb;" & _
"Jet OLEDB:System Database=c:\wbdb\secured.mdw;", _
SQLStatement:"SELECT * FROM 'Artikel'", _
SubType:=wdMergeSubTypeOther
```

## Microsoft Excel

### Über eine DDE-Verbindung

Das Argument `Connection` enthält den Zielbereich. Es kann in der Form eines Bereichsnamens, eines Zellenbereichs oder des Texts "Gesamtes Tabellenblatt" vorliegen.

```
ActiveDocument.MailMerge.OpenDataSource Name:="C:\Datenbank\Bestellungen.xls", _
    Connection:="R1C1:R6C6", SubType:=wdMergeSubTypeWord2000
```

### Über eine ODBC-Verbindung

Am besten klappt es, wenn der Dateipfad in der Verbindungszeichenkette angegeben und der Tabellenname Teil der SQL-Anweisung ist:

Um ab Word 2002 eine solche Verbindung aufzubauen, brauchen Sie die `SubType`-Parameter.

```
ActiveDocument.MailMerge.OpenDataSource Name:="", _
    Connection:="DSN=Datenbank-excel-us;", _
    SQLStatement:="SELECT * FROM 'Bestellungen'"
```

### Über Jet-OLE DB und eine leere .odc-Datei

Um ab Word 2002 eine solche Verbindung aufzubauen, brauchen Sie die `SubType`-Parameter.

```
ActiveDocument.MailMerge.OpenDataSource Name:="c:\Datenbank\leer.odc", _
    Connection:="Provider=Microsoft.Jet.OLEDB.4.0;" & _
        "Data Source=c:\Datenbank\Bestellungen.xls;" & _
        "Extended Properties=\"Excel 8.0;HDR=Yes\";", _
    SQLStatement:="SELECT * FROM 'Bestellungen'", _
    SubType:=wdMergeSubTypeOther
```

# Zusammenfassung

In diesem Kapitel wurde ein Überblick über die Arbeit mit Daten im Objektmodell von Word vermittelt. Es wurden unterschiedliche Möglichkeiten vorgestellt.

- Begonnen wurde das Kapitel mit einer Diskussion zum Thema Textmarken (Seite 340).
- Anschließend folgte eine Vorstellung der verschiedenen Nutzen einer Tabelle in Word-Dokumenten (Seite 349 ff.).
- Danach wurde der Umgang mit Feldfunktionen (Seite 378 ff.) beschrieben.
- In den nächsten Abschnitten wurden Formularfelder (Seite 391 ff.) und Inhaltssteuerobjekte (Seite 399 ff.) erläutert.
- Eine Diskussion zum Thema Seriendruck (Seite 427 ff.) rundete 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 8

# Ereignisse in Word

### In diesem Kapitel:

Auto-Makros als Pseudo-Ereignisse	448
Ereignisse auf Dokumentebene	450
Ereignisse auf Applikationsebene	453
Klassenmodule einrichten und initialisieren	455
Übersicht über die verfügbaren Ereignisse	458
Seriendruckereignisse	472
Zusammenfassung	478

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 ab 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:

#### ■ Befehlereignisse

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 Befehlereignisse, 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 16 sowie in Kapitel 20.

#### ■ 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 als Pseudo-Ereignisse« in diesem Kapitel).

In diesem Kapitel erfahren Sie, welche Ereignisse Word 2003 und Word 2007 auf Dokument- und Anwendungsebene besitzen und wie Sie diese nutzen können.

## Auto-Makros als Pseudo-Ereignisse

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.

Diese Makros werden automatisch ausgeführt, wenn ein Makro mit diesem Namen erstellt und die mit dem Auto-Makro verknüpfte Aktion ausgeführt wird. Damit Word diese auszuführenden Makros als solche erkennt, müssen sie besonders benannt werden.

Tabelle 8.1 Zusammenstellung der Auto-Makros und deren Ausführungszeitpunkt

Bezeichnung	Zeitpunkt der Ausführung
AutoExec	Beim Starten von Word bzw. beim Laden eines Add-Ins
AutoOpen	Beim Öffnen eines bestehenden Dokuments
AutoClose	Beim Schließen eines Dokuments
AutoNew	Beim Erstellen eines neuen Dokuments
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.

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 *ThisDocument* 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*.

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 (siehe auch Kapitel 14).

**ACHTUNG** 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.

Hingegen werden folgende Makros nicht in Add-Ins ausgeführt:

- `AutoOpen`
- `AutoNew`

Diese werden nicht ausgeführt, da Add-Ins von Word beim Start mitgeladen und keine Dokumente auf ihrer Basis erstellt werden.

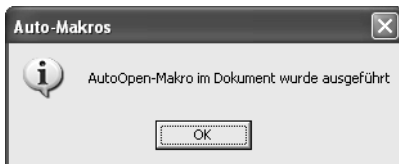
Die Reihenfolge für das Laden der im Autostart-Ordner gespeicherten Add-Ins kann dabei nicht beeinflusst werden.

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.

Dadurch, dass das Auto-Makro `AutoExec` in jedem Add-In ausgeführt wird, kann mit diesem Makro z.B. sichergestellt werden, dass die Schnittstelle zu einem Klassenmodul auf jeden Fall initialisiert wird (siehe den Abschnitt »Klassenmodule einrichten und initialisieren« in diesem Kapitel).


Abbildg. 8.1 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 (dazu mehr in Kapitel 1).

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 5.

Auch in Word 2007 hat sich an diesen Auto-Makros nichts geändert und können wie gewohnt eingesetzt werden.

## Ereignisse auf Dokumentebene

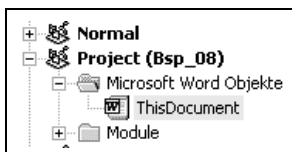
Neben den Auto-Makros gibt es auf Dokumentebene vergleichbare Ereignisse, mit denen auf das Öffnen, Erstellen oder Schließen von Dokumenten reagiert werden kann.

Tabelle 8.2 Ereignisse auf Dokumentebene

Name des Ereignisses	Aufrufende 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. 8.2 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 21.

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 8.3 gezeigte Fehlermeldung.

**Abbildg. 8.3** 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 8.1 in das Modul *ThisDocument* einer Dokumentvorlage oder erstellen Sie ein neues Dokument auf der Basis der Beispiel-Dokumentvorlage *Bsp\_08-1.dot*.



Die Beispiel-Dokumentvorlage *Bsp\_08-1.dot* finden Sie auf der CD-ROM zum Buch im Ordner `\Beispiele\Kap08`.

Kopieren Sie die Dokumentvorlage in ein beliebiges Verzeichnis auf Ihrer Festplatte (z.B. in den Ordner *Eigene Dateien* unter Windows XP/2003 bzw. in den Ordner *Dokumente* unter Windows Vista) 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 8.4 angezeigt.

**Listing 8.1** 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  
    ' Initialisieren des Applikationereignisses  
    If oApp Is Nothing Then Set oApp = ThisDocument.Application  
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 8.4 angezeigt werden.



Abbildg. 8.4 Ausgabe beim Erstellen eines neuen Dokuments basierend auf einer Vorlage mit *Document\_New*-Ereignis



**HINWEIS** Die weiteren Dokument-Ereignisse, die Objekt-gebunden sind, werden in Kapitel 7 (Inhaltssteuerelemente) und Kapitel 12 (ActiveX-Steuerelemente) behandelt.

## 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 ein-

richten und initialisieren müssen (siehe den Abschnitt »Klassenmodule 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\_08-1.dot* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap08* zum einen direkt mit einem Doppelklick aufrufen und zum anderen 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. 8.5 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 8.6) erkennen, dass keine Ereignisse im *ThisDocument*-Klassenmodul ausgelöst werden.

Abbildg. 8.6 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.

# Klassenmodule 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.

## Klassenmodule 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 8.2** 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 8.3 aufgeführte Prozedur.

**Listing 8.3**      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
  - Quit
- Dokument-spezifische Ereignisse
  - NewDocument
  - DocumentOpen
  - DocumentChange
  - DocumentBeforeClose
  - DocumentBeforePrint
  - DocumentBeforeSave
  - DocumentSync
- Serienbrief-spezifische Ereignisse
  - MailMergeAfterMerge
  - MailMergeAfterRecordMerge
  - MailMergeBeforeMerge
  - MailMergeBeforeRecordMerge
  - MailMergeDataSourceLoad
  - MailMergeDataSourceValidate
  - MailMergeDataSourceValidate2 (Word 2007)
  - 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.

## Klassenmodule 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 8.4). Mit dieser Zuweisung entfällt gleichzeitig der ansonsten notwendige Objektverweis mittels `Set`-Anweisung (siehe das Kapitel 4).

Listing 8.4

Globale Deklaration einer neuen Klasseninstanz vom Klassenmodul *EventClassModule*

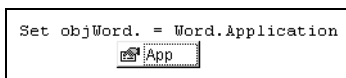
```
Dim objWord As New EventClassModule
```

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 8.2), so dass in der Auswahlliste nur diese Variable `App` angeboten wird (siehe Abbildung 8.7).

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 21) repräsentiert wird.

Abbildg. 8.7

Auswahlliste mit allen öffentlichen Variablen des Klassenmoduls



Damit das Klassenmodul einer Add-In-Vorlage automatisch beim Start von Word initialisiert wird, erfolgt die Objektzuweisung in der Prozedur *AutoExec()*.

**Listing 8.5** 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 und in Word 2007 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. Der Wechsel vom Dokument zum Überarbeitungsfenster löst dieses Ereignis ebenfalls aus. Die Parameter sind in Tabelle 8.3 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 8.6; das Ergebnis ist in Abbildung 8.8 ersichtlich.

**Tabelle 8.3** Beschreibung der Parameter der Ereignis-Prozedur *WindowActivate*

Parameter	Beschreibung
Doc	Referenz auf das aktivierte Dokument mit allen Eigenschaften und Methoden eines <code>Document</code> -Objekts
Wn	Referenz auf das aktivierte Fenster mit allen Eigenschaften und Methoden eines <code>Window</code> -Objekts

**Listing 8.6** Beispiel zur Ereignis-Prozedur *WindowActivate*, das die jeweiligen Parameter anzeigt

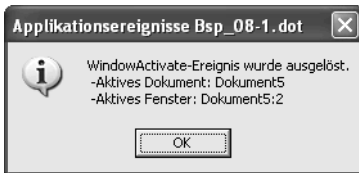
```
Private Sub App_WindowActivate(ByVal Doc As Document, ByVal Wn As Window)  
    Dim strMSG As String  
    Const c_Title As String = "Applikations-Ereignisse "  
    strMSG = "WindowActivate-Ereignis wurde ausgelöst." & vbCrLf & _
```

Listing 8.6 Beispiel zur Ereignis-Prozedur *WindowActivate*, das die jeweiligen Parameter anzeigt (Fortsetzung)

```

" -Aktives Dokument: " & Doc.Name & vbCrLf & _
" -Aktives Fenster: " & Wn.Caption & vbCrLf
MsgBox strMSG, vbInformation, c_Title & ThisDocument.AttachedTemplate.Name
End Sub

```

Abbildg. 8.8 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 8.7, das Ergebnis ist in Abbildung 8.9 ersichtlich.

Listing 8.7 Die Ereignis-Prozedur *WindowSize* gibt die Größe des aktuellen Fensters aus, wenn es verkleinert oder vergrößert wird

```

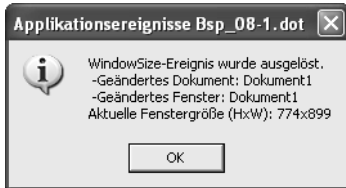
Private Sub App_WindowSize(ByVal Doc As Document, ByVal Wn As Window)
    Dim strMSG As String
    Const c_Title As String = "Applikations-Ereignisse "
    strMSG = "WindowSize-Ereignis wurde ausgelöst." & vbCrLf & _
        " -Geändertes Dokument: " & Doc.Name & vbCrLf & _
        " -Geändertes Fenster: " & Wn.Caption & vbCrLf
    Select Case Wn.WindowState

```

**Listing 8.7** Die Ereignis-Prozedur *WindowSize* gibt die Größe des aktuellen Fensters aus, wenn es verkleinert oder vergrößert wird (*Fortsetzung*)

```
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. 8.9** 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 8.4 aufgelistet. Die Syntax lautet:

```
Private Sub App_WindowBeforeDoubleClick(ByVal Sel As Selection, Cancel As Boolean)
```

**Tabelle 8.4** 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 8.8 wird geprüft, ob das markierte Objekt ein *InlineShape* ist. Nur für diesen Fall wird gefragt (Abbildung 8.10), ob das standardmäßig angezeigte Dialogfeld zum Formatieren der Grafik angezeigt werden soll oder nicht.

**Listing 8.8** 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
```



Listing 8.8 Beispiel zur Ereignis-Prozedur *WindowBeforeDoubleClick* (Fortsetzung)

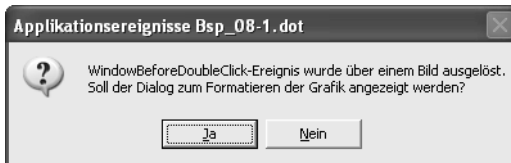
```

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

```



**ACHTUNG** Im Gegensatz zu Word 2003 wird in Word 2007 der Dialog bei einem Doppelklick auf die Grafik nicht angezeigt, da dort die Formatierungseinstellungen entweder über die Registerkarte *Bildtools/Format* oder über das Kontextmenü erreicht werden.

Abbildg. 8.10 Ausgabe der Aktion der Ereignis-Prozedur *WindowBeforeDoubleClick*

## 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 8.9 wird dann ein geändertes Kontextmenü angezeigt, über das Sie den Formularschutz ein- bzw. ausschalten können (Abbildung 8.11).

Listing 8.9 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
    ElseIf ActiveDocument.ProtectionType = wdNoProtection Then
        Dim fldFF As FormField

```

**Listing 8.9** Die Ereignis-Prozedur *WindowBeforeRightClick* blendet bei Formularfeldern ein eigenes Kontextmenü ein (*Fortsetzung*)

```

    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 8.9 funktioniert, müssen Sie in ein Modul die Prozeduren *ProtectDoc* und *UnprotectDoc* zum Setzen und Aufheben des Formularschutzes kopieren, wie in Listing 8.10.

**Listing 8.10** Prozeduren zum Setzen und Aufheben des Formularschutzes

```

Public Sub ProtectDoc()
    If ActiveDocument.ProtectionType = wdNoProtection Then
        ActiveDocument.Protect Type:=wdAllowOnlyFormFields, NoReset:=True
    End If

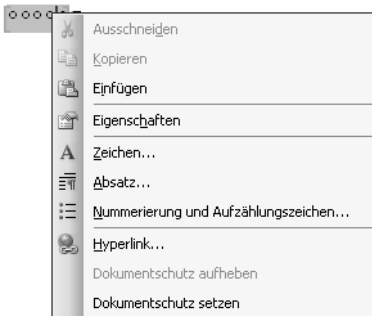
```

Listing 8.10 Prozeduren zum Setzen und Aufheben des Formularschutzes (Fortsetzung)

```

End Sub
Public Sub UnprotectDoc()
    If ActiveDocument.ProtectionType = wdAllowOnlyFormFields Then
        ActiveDocument.Unprotect
    End If
End Sub

```

Abbildg. 8.11 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 16.

## 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 8.5 aufgelistet, die Syntax lautet:

```
Private Sub App_WindowSelectionChange(ByVal Sel As Selection)
```

Tabelle 8.5 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**

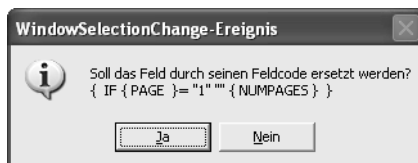
Das Ereignis *WindowSelectionChange* wird nicht ausgelöst, wenn Sie mit der Tastatur die Einfügemarke verschieben, z.B. bei der Texteingabe.

Im Beispiel in Listing 8.11 wird geprüft, ob die Einfügemarke in ein Feld verschoben wurde. Nur für diesen Fall wird gefragt (Abbildung 8.12), ob das entsprechende Feld durch seinen Feldcode ersetzt werden soll oder nicht.

**Listing 8.11** 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
                        Exit For
                    End If
                End If
            End If
        End If
    Next fldField
End Sub
```

**Abbildg. 8.12** Ein Feld wird mittels *WindowSelectionChange* durch seinen Feldcode ersetzt



## 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 8.6 aufgelistet.

**Tabelle 8.6** Der Parameter der Ereignis-Prozedur *NewDocument*

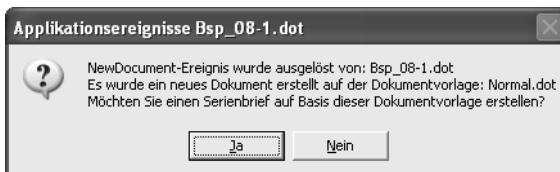
Parameter	Beschreibung
Doc	Referenz auf das neu erstellte Dokument mit allen Eigenschaften und Methoden eines <code>Document</code> -Objekts

Sie können mit diesem Ereignis z.B. direkt eine Aktion mit dem erstellten Dokument verknüpfen. Wenn es auf einer bestimmten Dokumentvorlage basiert (in Listing 8.12 auf der *Normal.dot*), können Sie z.B. direkt den Assistenten zum Erstellen eines Serienbriefs aufrufen.

**Listing 8.12** 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. 8.13** Erstellen eines Serienbriefdokuments als Beispiel zur Ereignis-Prozedur *NewDocument*



**HINWEIS**

Entwickler, die Word über ein COM-Add-In oder eine andere Anwendung steuern, müssen diese Ereignisse auswerten, um festzustellen, ob ein Dokument neu erstellt, geöffnet oder allenfalls geschlossen wurde. AutoMakros sowie Ereignisse im Modul ThisDocument stehen außerhalb von VBA nicht zur Verfügung.

## 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 8.13 wird geprüft, ob der Titel in den Eigenschaften ausgefüllt ist und ggf. das Dialogfeld mit den Dokumenteigenschaften anzeigt. 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 8.13

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 8.14 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 8.14** 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 8.7 aufgelistet.

```
Private Sub App_DocumentBeforeClose(ByVal Doc As Document, Cancel As Boolean)
```

**Tabelle 8.7** 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 8.15 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 8.14), ob er die Änderungen sehen und prüfen möchte.

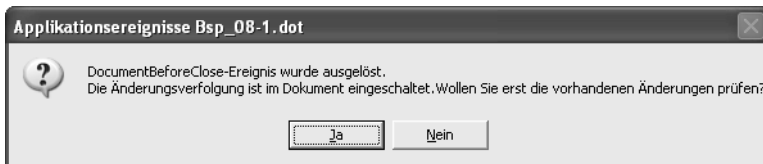
**Listing 8.15** Überprüft das zu schließende Dokument, ob Überarbeitungsänderungen vorhanden sind

```
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
```

**Listing 8.15** Überprüft das zu schließende Dokument, ob Überarbeitungsänderungen vorhanden sind (Fortsetzung)

```
End If
End Sub
```

**Abbildg. 8.14** 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 8.8 aufgelistet, die Syntax lautet:

```
Private Sub App_DocumentBeforeSave(ByVal Doc As Document, SaveAsUI As Boolean, _
    Cancel As Boolean)
```

**Tabelle 8.8** 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 8.16 wird vor dem Speichern gefragt (Abbildung 8.15), 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.

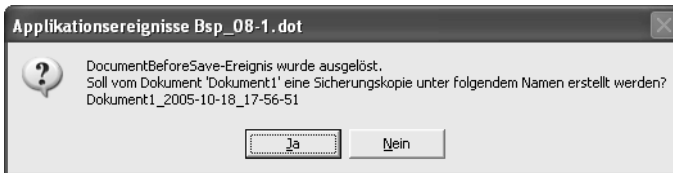


Listing 8.16 Die Ereignis-Prozedur *DocumentBeforeSave* fragt nach, ob eine Sicherungskopie erstellt werden soll

```

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 = IIf(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
        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

```

Abbildg. 8.15 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 8.17 wird das Ereignis dazu verwendet, um vor dem Ausdruck ein Wasserzeichen in Form des Schriftzuges »Kopie« in das Dokument einzufügen (Abbildung 8.16).

Listing 8.17 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

```

**Listing 8.17** Einfügen eines Wasserzeichens mit der Ereignis-Prozedur *DocumentBeforePrint* (Fortsetzung)

```

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, _
        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. 8.16** 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 8.9 aufgelistet, die Syntax lautet:

```
Private Sub app_DocumentSync(ByVal Doc As Document, _
    ByVal SyncEventType As Office.MsoSyncEventType)
```

**Tabelle 8.9** 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 8.18** 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 = "Applikations-Ereignisse "
    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 7 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 8.10 erläutert. Die Syntax lautet:

```
MailMergeBeforeMerge(ByVal Doc As Document, ByVal StartRecord As Long, _  
ByVal EndRecord As Long, Cancel As Boolean)
```

Tabelle 8.10 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.

Verschiedene Leser und Newsgruppen haben gemeldet, es wird auch sonst bei der Zusammenführung mit dem Ereignis `MailMergeBeforeRecordMerge` nur ein Datensatz zusammengeführt. Die Autoren und Microsoft haben dieses Verhalten nicht reproduzieren können, vermuten jedoch, dass es sich um ein Problem mit ungenügenden Systemressourcen handeln könnte. Wir raten also, solche Lösungen auf Einsatz- statt Entwicklerrechnern zu entwickeln und testen.

## 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 Seriendruck-Hauptdokument 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 8.17 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. 8.17 Das Seriendruck-Hauptdokument ist in englischer Sprache, weil die meisten Lieferanten sich außerhalb Deutschlands befinden

Nordwind GmbH
München, den 31. Oktober 2005

Bsp08\_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 8.18 abgebildet. Ein Vergleich der beiden Abbildungen zeigt auf, dass die Tabelle in Abbildung 8.18 an die Stelle der Textmarke in Abbildung 8.17 eingefügt wird.

**HINWEIS**

Eine Diskussion des Seriendruck-Objektmodells befindet sich in Kapitel 7.

Abbildg. 8.18 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:

Product	Quantity
Gorgonzola Telino	22
12 x 100-g-Packungen	
Mozzarella di Giovanni	22
24 x 200 g-Packungen	

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. Den Namen 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 8.19 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 8.19** 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 8.20

erscheint. Sie ruft die Funktion *DatenSatzPrüfen* in Listing 8.21 auf. Gibt diese »Wahr« zurück, wird die Zusammenführung dieses Datensatzes unterbunden und Word geht sofort zum nächsten (MailMergeAfterRecordMerge wird nicht ausgelöst).

**Listing 8.20** 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« zurückgeben. Liegt doch eine Bestellung vor, wird die Liste ins Dokument in den Textmarkenbereich eingefügt und in eine Tabelle umgewandelt.

**Listing 8.21** 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

    Dim bIstAuslauf As Boolean
    Dim lBestellt As Long
```



Listing 8.21 Die 1:n-Listen jedes Lieferanten aufbauen (Fortsetzung)

```

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 8.22 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 8.22 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
    'Tabelle mit den Produkten entfernen und Textmarke
    'wieder erstellen, so dass das Dokument wieder in seinem

```

**Listing 8.22** Im Ereignis *MailMergeAfterRecordMerge* wird das Hauptdokument zurückgesetzt (*Fortsetzung*)

```
'ursprünglichen Zustand für den nächsten Datensatz bereit steht.
rng.Tables(1).Delete
doc.Bookmarks.Add Name:=sBkmName, Range:=rng
End Sub
```

Nachdem alle Datensätze abgearbeitet wurden, kommt das Ereignis *MailMergeAfterMerge* (Listing 8.23) 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 8.23** 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 *Bsp08\_03.doc* mit dem Beispielcode sowie *Bsp08\_Beiispiel.doc* mit dem Seriendruckresultat finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap08*. Die Seriendruck-Datenquelle *Artikelliste.txt* befindet sich im Ordner *\Beispiele\Kap08\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 20), stehen diese Automatisierungscodes auch außerhalb eines Word-VBA-Projekts zur Verfügung. Sie können beispielsweise in COM-Add-Ins sowie VSTO-Projekten benutzt werden.

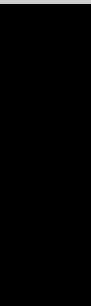
- Dieses Kapitel stellt auf Seite 450 sowie auf Seite 453 eine Übersicht der Dokument- bzw. Anwendungs-Ereignisse vor.
- Wie das Klassenmodul, das die Ereignisprozeduren enthält, erstellt und initialisiert wird, steht auf Seite 455 ff. beschrieben.
- Die Anwendungs- sowie Dokument-Ereignisse sind ab Seite 458 näher erläutert.
- Abschließend, beginnend auf Seite 472, haben wir die Seriendruckereignisse diskutiert. Ein praxisnahes Beispiel folgte ab Seite 473.

## Teil C

# Steuern und gesteuert werden

### In diesem Teil:

<b>Kapitel 9</b>	Grundlagen der Fernsteuerung	481
<b>Kapitel 10</b>	Word von anderen Umgebungen aus steuern	499
<b>Kapitel 11</b>	Andere Programme von Word aus steuern	559
<b>Kapitel 12</b>	Eingebettete OLE-Objekte	589



## Kapitel 9

# Grundlagen der Fernsteuerung

Steuern und gesteuert werden

### In diesem Kapitel:

Verweise auf externe Bibliotheken im VB-Editor	482
Early versus Late Binding	489
Zusammenfassung	497

In Teil B dieses Buches wurden das Objektmodell von Word sowie die Dokument- und Programmereignisse näher vorgestellt. In diesem Teil erläutern wir nun das Zusammenspiel von verschiedenen Applikationen:

- Im aktuellen Kapitel wird erläutert, wie externe Bibliotheken in das Projekt eingebunden und deren Ressourcen genutzt werden.
- In Kapitel 10 wird zunächst dargestellt, 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 11 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 12 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 im VB-Editor

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 12.0 Object Library
- Die zugehörige Dokumentvorlage

- OLE-Automation
- Microsoft Office 12.0 Object Library

Die beiden letzten Einträge sind optional und können bei Nichtgebrauch deaktiviert werden.

**HINWEIS** Die Versionsnummer der Microsoft Word bzw. Microsoft Office Object Library ist von der installierten Office-Version abhängig. Die einzelnen Versionsnummern sind: Office 97 – 8.0, Office 2000 – 9.0; Office 2002 – 10.0; Office 2003 – 11.0; Office 2007 – 12.0.

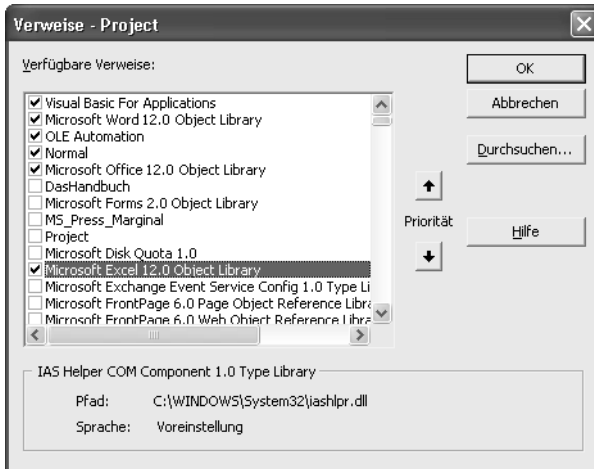


Verweis  
einfügen

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 9.1 ersichtlich, im Listenfeld *Verfügbare Verweise* die benötigte Bibliothek.

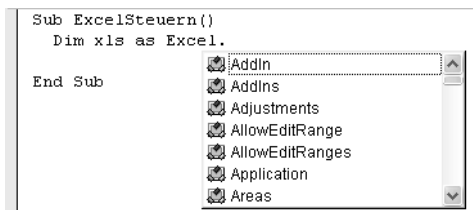
Wie Verweise auch dynamisch während der Laufzeit des Programms gesetzt werden können, ist in Kapitel 21 detailliert beschrieben.

Abbildg. 9.1 Fügen Sie einen Verweis auf die *Microsoft Excel 12.0 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 9.1 zeigt ein einfaches Beispiel, wie das Programm gesteuert werden kann.

Abbildg. 9.2 Unterstützung durch IntelliSense nach dem Einbinden der *Microsoft Excel 12.0 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 9.1** Erstellen einer Excel-Instanz und diese steuern

```
Sub ExcelSteuern()
'Das Beispiel verwendet einen Verweis auf die
'Microsoft Excel 12.0 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
```

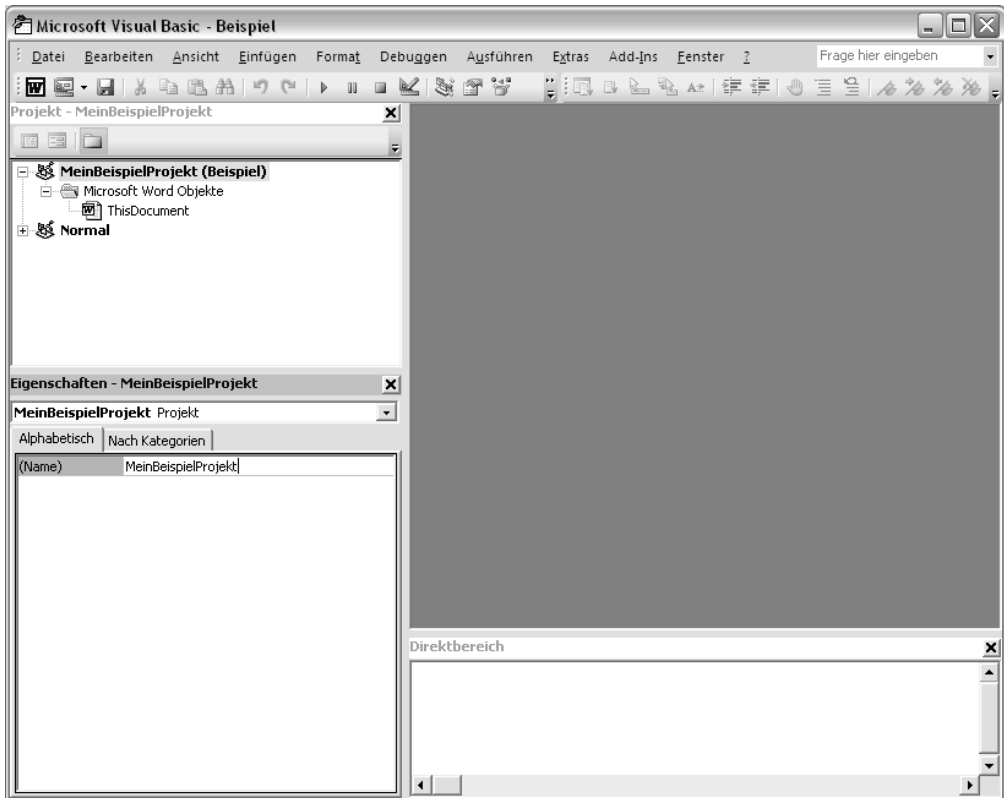
In Abbildung 9.1 erkennen Sie unter *Verfügbare Verweise* die Einträge »DasHandbuch« sowie »MS\_Press\_Marginal«. Es handelt sich dabei um geladene Add-Ins (*.dot*-Dateien), die Makros und Symbolleisten enthalten. Auch auf diese Add-Ins kann ein Verweis gesetzt werden, der es ermöglicht, die Prozeduren 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 10 aufgeführt.

Als Vorgabewert wird jedem VBA-Projekt innerhalb einer Dokumentvorlage der Name »Template-Project« zugewiesen. Bei einem Dokument wird der Name »Project« eingetragen. Diese Projekte erscheinen in der Liste *Verfügbare Verweise* unter eben diesen Namen. Sind mehrere Vorlagen mit



der gleichen Projektbezeichnung als Add-Ins geladen, entsteht ein Namenskonflikt. Es wäre nicht möglich, Makros mit dem gleichen Namen, in gleichnamigen Modulen eindeutig zu erkennen. Es empfiehlt sich somit, jedem VB-Projekt einen eindeutigen Namen zuzuweisen, so wie dies die Abbildung 9.3 veranschaulicht.

Abbildg. 9.3 Jedes VBA-Projekt soll einen eindeutigen Namen tragen



**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.

**HINWEIS**

In Kapitel 21 wird gezeigt, wie fehlende Verweise dynamisch korrigiert werden können.

### Einbinden von zusätzlichen Steuerelementen

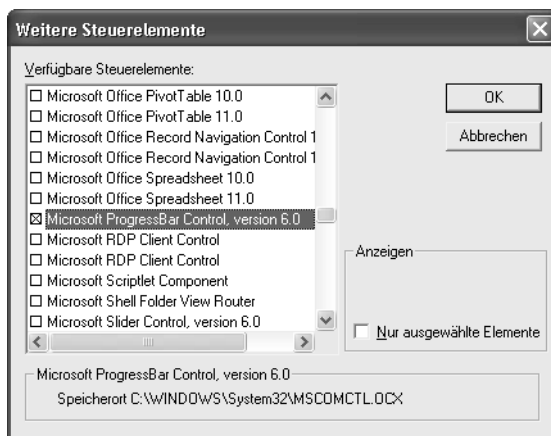
Grundsätzlich können neben den bestehenden Steuerelementen, die durch die *Microsoft Forms 2.0 Object Library* zur Verfügung gestellt werden, weitere Komponenten eingebunden werden.



Steuerelement einbinden

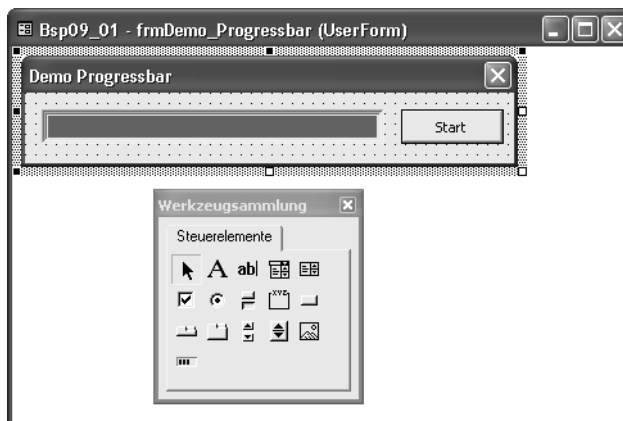
Dazu öffnen Sie innerhalb der VB-Entwicklungsumgebung ein Formular und wählen anschließend den Menübefehl *Extras/Zusätzliche Steuerelemente*. Im Listenfeld *Verfügbare Steuerelemente* aktivieren Sie nun, wie in Abbildung 9.4 ersichtlich, das Kontrollkästchen des gewünschten Steuerelements. Die Abbildung 9.5 zeigt das zusätzliche Steuerelement in der *Werkzeugsammlung*.

**Abbildg. 9.4** 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.

**Abbildg. 9.5** Verwenden Sie das zusätzliche Steuerelement *Progressbar* wie alle anderen Steuerelemente



**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 *Mscmctl.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\Kap09` auf der CD-ROM zum Buch.

### 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>)



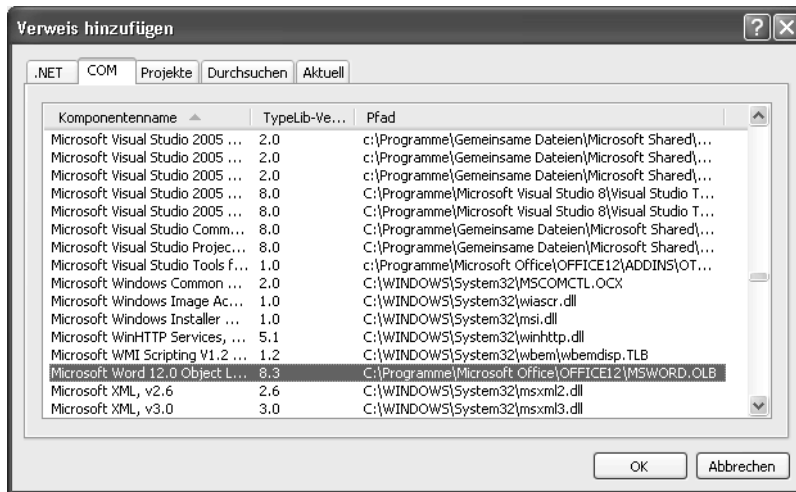
Das dargestellte Beispiel mit dem zusätzlichen Steuerelement finden Sie in der Beispieldatei *Bsp09\_01.doc* auf der CD-ROM zum Buch im Ordner `\Beispiele\Kap09`.

## Verweise in Visual Studio 2005

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 sogenannten »Interop Assemblies« (IAs). Wenn Sie in Visual Studio im Dialogfeld *Verweis hinzufügen* (siehe Abbildung 9.6) 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.

**Abbildg. 9.6** Das Dialogfeld *Verweis hinzufügen* (Registerkarte *COM*) wird über das *Projekt*-Menü eingeblendet



Für Office 2007, Office 2003 sowie Office XP stellt Microsoft »Primary Interop Assemblies« (PIAs) zur Verfügung, die für diese Aufgabe optimiert sind. Die Office 2007- und 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 und wird im Abschnitt »Late Binding in Visual Studio .NET« kurz vorgestellt.

#### **HINWEIS**

Wir bieten in diesem Buch nur eine Übersicht zum Thema »Interop Assemblies«. 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, USA.

# 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 9.2) zeigt, wie ein Makro mittels Early Binding auf die Kontakte von Microsoft Outlook zugreift und die gefundenen Einträge auflistet.

**Listing 9.2** Auflisten aller Kontakte aus Outlook unter Verwendung von Early Binding

```
Sub Earlybinding_Outlook()
'Das Beispiel verwendet einen Verweis auf
'Microsoft Outlook 12.0 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 *Bsp09\_02.doc* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap09*.

## 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 12.0 angepasst werden, wenn das Makro auf einer Arbeitsstation mit installiertem Word 2007 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 9.3 greift – wie Listing 9.2 – 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 9.3

Auflisten aller Kontakte aus Outlook unter Verwendung von Late Binding

```
Sub Latebinding_Outlook()
    Dim doc As Word.Document
    Dim oAnw 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 oAnw = CreateObject("Outlook.Application")

    'Kontakte-Ordner setzen
    Set olOrdner = oAnw.Session.GetDefaultFolder(10)
```

**Listing 9.3** Auflisten aller Kontakte aus Outlook unter Verwendung von Late Binding (Fortsetzung)

```
'Einträge setzen
Set olItem = olOrdner.Items

'Alle Kontakte auflisten
For Each olKontakt In olItem
    If olKontakt.Class = 40 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
```

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 Beispielmakro aus Listing 9.4 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 12.0 Object Library* manuell hinzugefügt oder eben wieder entfernt werden.



Listing 9.4 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 12.0 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. Im Weiteren können Compiler-Konstanten nur auf Modulebene deklariert werden und stehen nur in diesem Modul zur Verfügung.

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 *Bsp09\_03.doc* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap09*.

## Late Binding in Visual Studio .NET

Wie in Kapitel 5 in den Diskussionen zu WordBasic sowie Suchen und Ersetzen demonstriert, ist es möglich, wenngleich etwas umständlich, in C# geschriebene Office-Anwendungen mit Late Binding zu automatisieren. Diese Funktionalität gehört der Klasse `System.Runtime.InteropServices` 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>).

Wenn Sie mit VB.NET arbeiten und `Option Strict On` verwenden, werden Sie mit dem gleichen Problem konfrontiert und müssen wie für C# beschrieben vorgehen. `Option Strict On` stellt eine starke Typisierung bereit, um unbeabsichtigte Typkonvertierungen mit Datenverlust zu verhindern.

Es ist möglich, im gleichen Projekt `Option Strict On` und `Option Strict Off` zu kombinieren. Das Listing 9.5 veranschaulicht dieses Prinzip anhand einer einfachen Windows-Anwendung. `Option Strict` ist für die Klasse des Formulars aktiviert. Late Binding ist nur über `GetType().InvokeMember` möglich.

Für die Klasse `Class1` hingegen ist `Option Strict` deaktiviert und Late Binding funktioniert wie in VBA oder klassischem Visual Basic. Beide Klassen arbeiten mit der gleichen Instanz der Word-Anwendung.

**Listing 9.5** Late Binding in VB.NET mit *Option Strict* ein- sowie ausgeschaltet

```
Option Strict On
Public Class Form1
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
        Handles Button1.Click
        Dim app As Object
        Dim cls As New Class1
        app = cls.GetWordApp()
        Dim wdDocs As Object = Nothing
        wdDocs = app.GetType().InvokeMember("Documents", _
            Reflection.BindingFlags.GetProperty, Nothing, app, Nothing)
        Dim wdDoc As Object = Nothing
        wdDoc = wdDocs.GetType().InvokeMember("Add", _
            Reflection.BindingFlags.InvokeMethod, Nothing, wdDocs, Nothing)
        app = Nothing
    End Class

Option Strict Off
Public Class Class1
    Public Function GetWordApp() As Object
        Dim wdApp As Object
        wdApp = CreateObject("Word.Application")
        wdApp.visible = True
        Return wdApp
    End Function
End Class
```

#### **HINWEIS**

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>)



Das Beispielprojekt finden Sie in der Beispieldatei *Kap09\_VB.zip* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap09*.

# Zusammenfassung

In diesem Kapitel wurden die Grundlagen der Fernsteuerung vermittelt. Dabei ging es in erster Linie um allgemein gültiges Wissen.

- Als Erstes wurde dargestellt, wie ein Verweis auf eine Programmbibliothek dem aktuellen VBA-Projekt hinzugefügt wird, damit die zugehörige Applikation ferngesteuert werden kann (Seite 482).
- Im Weiteren wurde gezeigt, wie zusätzliche Steuerelemente eingebunden und auf einem Userform-Objekt verwenden werden können (Seite 486).
- Dann wurden die Themen Verweise und Interop Assemblies in Visual Studio .NET 2005 vorgestellt (Seite 487).
- Es wurden die Vor- und Nachteile von Early Binding gegenüber von Late Binding erläutert (Seite 487). Und es wurde aufgezeigt, wie man sich die Vorteile beider Varianten zu Nutzen machen kann (Seite 492).
- Abschließend wurde Late Binding innerhalb des .NET Framework kurz diskutiert (Seite 495).



## Kapitel 10

# Word von anderen Umgebungen aus steuern

### In diesem Kapitel:

Fernsteuern von Microsoft Word	500
Programmzeilen zentral verwalten und gemeinsam nutzen (Add-Ins)	508
Fernsteuerung aus Office-fremden Umgebungen	513
Die Word-Anwendung über Visual Studio .NET fernsteuern	513
VSTO 2005 Dokumentlösungen	519
VSTO COM-Add-Ins	543
Zusammenfassung	558

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 2005 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 in Programmierumgebungen außerhalb von VSTO erzeugt und genutzt werden, übersteigt den Rahmen dieses Buches. Informationen dazu finden Sie auf der MSDN-Webseite, beispielsweise im Artikel »How to build an Office 2000 COM add-in in Visual Basic« (<http://support.microsoft.com/kb/238228/en-us>).

## 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« weiter hinten 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 10.1 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 10.1 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 13 zusammengefasst.

**HINWEIS** Damit das Beispielmakro aus Listing 10.1 fehlerfrei abgearbeitet werden kann, muss unbedingt die Datei *Bsp10\_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 *Bsp10\_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 14 nachgeschlagen werden.

**Listing 10.1** Ein Excel-Makro steuert Word durch Verwendung der *Run*-Methode

```
Sub Demo WordFernsteuernRun()
'Damit dieses Beispiel funktioniert, muss die Datei Bsp10_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

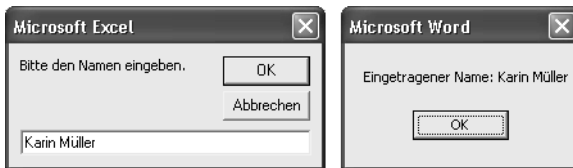
        Set docApp = Nothing
    End If
End Sub
```



Die Prozedur *Demo\_WordFernsteuernRun* befindet sich in der Datei *Bsp10\_02.xls* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap10*. 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 *Bsp10\_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 10.1 dargestellt.

Abbildg. 10.1 Eingabe des Werts in Excel, Ausgabe desselben in Word



Aus dem in Listing 10.1 aufgezeigten Excel-Makro erfolgt ein Aufruf eines Word-Makros anhand der *Run*-Methode. Die entsprechende Programmsequenz ist in Listing 10.2 aufgeführt.

Listing 10.2 Das aus Excel heraus aufgerufene 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 10.1).

### 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 10.1 fehlerfrei ausgeführt werden kann, muss die Datei *Bsp10\_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 5, im Abschnitt zum Thema Templates, näher vorgestellt.

In Listing 10.3 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 *Bsp10\_02.xls*).

**Listing 10.3** Zusätzliche Add-Ins können bei Bedarf dynamisch geladen und entladen werden

```
Const strADDIN As String = "C:\WordBuch\Beispiele\Kap10\Bsp10_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 10.4 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 *Bsp10\_Brief.dot* basiert.

**Listing 10.4** Aus Excel heraus Word fernsteuern und die Empfängeradresse in den Brief eintragen

```
Sub Demo_AdresseInDokumentÜbertragen()
    Const strDOT_BRIEF As String = "C:\WordBuch\Beispiele\Kap10\Bsp10_Brief.dot"

    #Const EARLYBINDING = False 'oder True

    'Variablen und Objekte anlegen
    #If EARLYBINDING Then
        'Wird mit Early Binding gearbeitet, muss ein Verweis auf
```

Listing 10.4 Aus Excel heraus Word fernsteuern und die Empfängeradresse in den Brief eintragen (Fortsetzung)

```

'die "Microsoft Word 11.0 bzw. 12.0 Object Library" aktiviert werden.
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."

```

**Listing 10.4** Aus Excel heraus Word fernsteuern und die Empfängeradresse in den Brief eintragen (Fortsetzung)

```
End If

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 *Bsp10\_02.xls*. Als zusätzliche Hilfsdateien werden die drei Dokumentvorlagen *Bsp10\_03.dot*, *Bsp10\_04.dot* und *Bsp10\_Brief.dot* benötigt. Sämtliche Dateien befinden sich auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap10*.

## 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 11 beim Fernsteuern von weiteren Anwendungen mehrmals zum Einsatz kommen.

**Listing 10.5** Ausdrucken eines Dokuments mittels Fernsteuerung von Word

```
Sub Demo_WordFernsteuern()
    Const strDATEI_NAME As String = "C:\WordBuch\Beispiele\Kap10\Bsp_Demo.doc"

    System.Cursor = wdCursorWait

    'Variablen und Objekte anlegen
    Dim docApp As Word.Application
```

Listing 10.5 Ausdrucken eines Dokuments mittels Fernsteuerung von Word (Fortsetzung)

```

Dim docDoc As Word.Document

'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 10.5 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 »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 5 vorgestellt.

**WICHTIG**

Wir empfehlen Ihnen, 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 *Bsp10\_01.doc* auf der CD-ROM zum Buch im Ordner `\Beispiele\Kap10`.

## Programmzeilen zentral verwalten und gemeinsam nutzen (Add-Ins)

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 2 oder die deklarierten APIs aus Kapitel 3 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 9 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 10.6 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 10.2).

**Listing 10.6** Aufruf der allgemeinen Funktion *funcAlterBestimmen* aus dem Add-In

```
Sub Demo_AlterBestimmen()
    Dim dateMeinGeburtstag As Date

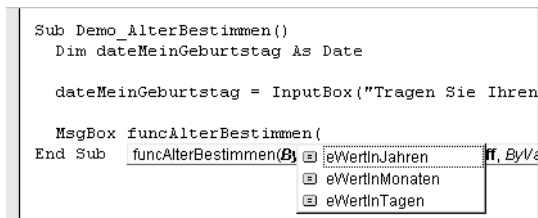
    dateMeinGeburtstag = InputBox("Tragen Sie Ihren Geburtstag ein.", , "24.12.2001")

    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 2 zu finden.

In diesem Beispiel wurde die Aufzählung *EDateDiff* und die Konstante *cAppName* zentral im Add-In deklariert.

**Abbildg. 10.2** 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 10.7 wird gezeigt, wie das gewünschte Rückgabeformat als Parameter an die Funktion übergeben werden kann.

**Listing 10.7** Zentrale Funktion zur Berechnung des Alters

```
Public Enum EDateDiff
    eWertInTagen = 0
    eWertInMonaten = 1
    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 10.2).

```
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 10.6 finden Sie in der Beispieldatei *Bsp10\_06.doc*. Als zusätzliche Hilfsdatei wird die Datei *Bsp10\_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\Kap10*.

## 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 15 entnommen werden.

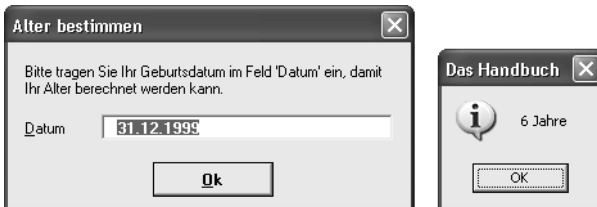
Da es sich bei einem UserForm-Objekt um eine spezielle Art von Klassen handelt, besteht keine Möglichkeit, diese direkt aus dem aktuellen Makro heraus aufzurufen oder eine neue Instanz des UserForm-Objekts zu erzeugen und zu starten. Stattdessen muss wie in Listing 10.8 der Umweg über eine allgemeine 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 10.9 aufgeführt.

**Listing 10.8** 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. 10.3** 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 10.9** 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 10.8 finden Sie in der Beispieldatei *Bsp10\_06.doc*. Als zusätzliche Hilfsdatei wird die Datei *Bsp10\_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\Kap10*.

## 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 Kapitels 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 10.10 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 10.10** 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("Bsp10_05.dot").Path & "\" _
        & wdapp.AddIns("Bsp10_05.dot").Name
    'Add-In unsichtbar öffnen
    Set doc = wdapp.Documents.Open(Filename:=sDateiName, _
```

Listing 10.10 Prozeduren in einem Word-Add-In außerhalb von Word aufrufen (Fortsetzung)

```

    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

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 *Bsp10\_07.xls*. Diese befindet sich auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap10*. Es ruft Prozeduren in der Vorlage *Bsp10\_05.dot* auf, die sich im *StartUp*-Ordner von Word befinden muss.

## Fernsteuerung aus Office-fremden Umgebungen

Die Fernsteuerung von Word aus anderen COM-Programmierungsumgebungen 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 9 beschrieben, kann sie sowohl über Early als auch Late Binding erfolgen.

Wir werden hier auf die alten Technologien nicht näher eingehen, Sie finden jedoch einige Beispiele der Fernsteuerung aus dem klassischen Visual Basic 6 heraus in den Kapiteln 18, 23 sowie 24.

## Die Word-Anwendung über Visual Studio .NET fernsteuern



Die Kapitel 5, 6 und 7 enthalten reichlich Beispiele für die Fernsteuerung des Word-Objektmodells aus der .NET-Umgebung. Der Code all dieser Lösungen ist von Word und seinen Dokumenten abgekoppelt, im Vergleich zu den Prozeduren, die in Word-VBA vorgestellt werden und sich inner-

halb eines Word-Dokumentes oder einer Word-Vorlage befinden. Deshalb werden hier die Grundlagen für die Fernsteuerung der Word-Anwendung vorgestellt.

In diesem Abschnitt werden der Aufbau und die Verwaltung der Verbindung zur COM-Anwendung Word kurz vorgestellt.

## Eine laufende Instanz ansprechen sowie Word starten

Bei der Planung einer Anwendung, die Word fernsteuert, muss zuerst abgeklärt werden, ob eine eigens dafür unabhängige Instanz von Word benötigt wird oder ob eine Verbindung zu einer bereits laufenden Instanz herzustellen ist. Erfolgt die Arbeit in Word ohne Mitwirken des Benutzers, ist eine unabhängige Instanz oft vorzuziehen. Es darf jedoch nicht vergessen werden, dass dies mehr System-Ressourcen beansprucht.

### TIPP

Falls die Anwendung dem Benutzer Werkzeuge zur Verfügung stellt, sollte ein COM-Add-In in Erwägung gezogen werden, das in der Word-Umgebung integriert ist. Da ein COM-Add-In automatisch mit Word geladen und geschlossen wird, entfallen die Verwaltungs- und Synchronisationssorgen einer getrennten Anwendung.

Eine unter Windows installierte COM-Anwendung kann durch eine .NET-Anwendung prinzipiell auf zwei Arten gestartet werden:

- Die .exe-Datei wird ausgeführt, als wenn der Benutzer auf ein Desktop-Symbol doppelklicken würde. Will der Entwickler ausschließlich Late Binding verwenden, um Word fernzusteuern, wird diese Methode gewählt. Hierfür wird `System.Diagnostics.Process.Start` verwendet.
- Die Anwendung wird über ihre COM-Registrierung (`Word.Application` beispielsweise) mit dem Schlüsselwort `new` instanziiert. Dieser Vorgang startet immer eine neue Word-Instanz, die unabhängig von allfällig schon laufenden bleibt.

Die erste Methode hat den Vorteil, dass Word mit Startoptionen (siehe Anhang D) gestartet werden kann. Ihr Nachteil ist, dass Word der .NET-Anwendung nicht automatisch als Automatisierungs-Objekt zur Verfügung steht und zwecks Fernsteuerung nachträglich eingebunden werden muss.

Um eine Verbindung zu einer laufenden Instanz herzustellen, wird `System.Runtime.InteropServices.Marshal.GetObject` benutzt.

### PROFITIPP

Laufen mehrere Instanzen einer COM-Anwendung gleichzeitig, ist es nur bedingt möglich, eine bestimmte anzusprechen. Sie können es mit `System.Runtime.InteropServices.Marshal.BindToMoniker` versuchen. Dieser Methode wird die Pfadangabe zu einem Dokument übergeben. Falls es schon geöffnet ist, wird eine Verbindung zur Anwendung hergestellt, ansonsten wird das Dokument in einer neuen Instanz geöffnet.

```
Word.Document doc = Marshal.BindToMoniker(PfadAngabe) as Word.Document;
Word.Application wdApp = doc.Application as Word.Application;
```

Das Listing 10.11 veranschaulicht, wie eine C#-Anwendung mittels der ersten Methode dem Benutzer ein neues Word-Dokument bereitstellt. Zuerst führt die Prozedur `btnWordStart_1_Click` die

Funktion *GetWordInstance\_1* aus. Diese prüft, ob eine laufende Word-Instanz vorhanden ist, indem sie nach Prozessen namens »WinWord« sucht:

```
Process[] wdPcs = Process.GetProcessesByName("WinWord");
```

Sind welche vorhanden, wird die zuletzt gestartete der Word-Objektvariable *wdApp* zugewiesen.

```
if (wdPcs.Length > 0)
{
    pcs = wdPcs[0];
    wdApp = (wd.Application) Marshal.GetActiveObject("Word.Application");
}
```

Falls nach diesem Codeblock die Objektvariable *wdApp* noch nicht instanziiert wurde, wird die Word-Anwendung explizit gestartet und diese Instanz *wdApp* zugewiesen.

```
if (wdApp == null)
{
    Process wdProcess = Process.Start("winword.exe");
    wdApp = (wd.Application) Marshal.GetActiveObject("Word.Application");
}
```

#### PROFITIPP

Unter Umständen startet die Anwendung nicht schnell genug, um sie sofort der Objekt-Variablen zuweisen zu können. Deshalb ist in der Prozedur *GetWordInstance\_1* eine Schleife eingebaut, worin getestet wird, ob die Zuweisung schon stattgefunden hat. Zusätzlich läuft ein Timer-Objekt, um nicht Gefahr einer unendlichen Schleife zu laufen.

Am Schluss wird nochmals kontrolliert, ob *wdApp* initiiert wurde, und der entsprechende Wert wird zurückgegeben. Ist dieser »Wahr«, wird das Word-Anwendungsfenster maximiert. (Das Starten von Word über seine \*.exe-Datei erstellt automatisch ein neues Dokument in einem sichtbaren Fenster.)

#### Listing 10.11 Word über die Kommandozeile starten und die laufende Instanz einbinden

```
using System.Diagnostics;
using System.Runtime.InteropServices;
using wd = Microsoft.Office.Interop.Word;
public partial class Form1 : Form
{
    //Variablen auf der Klassenebene
    wd.Application wdApp;
    Process pcs;
    object missing = System.Reflection.Missing.Value;
    object objTrue = true;

    private void btnWordStart_1_Click(object sender, EventArgs e)
    {
        if (GetWordInstance_1())
        {
            try
            {
                wdApp.ActiveWindow.WindowState =
```

**Listing 10.11** Word über die Kommandozeile starten und die laufende Instanz einbinden (*Fortsetzung*)

```

        Microsoft.Office.Interop.Word.WdWindowState.wdWindowStateMaximize;
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}

private bool GetWordInstance_1()
{
    bool retVal = false;
    try
    {
        Process[] wdPcs = Process.GetProcessesByName("WinWord");
        if (wdPcs.Length > 0)
        {
            pcs = wdPcs[0];
            wdApp = (wd.Application)Marshal.GetActiveObject("Word.Application");
        }
        if (wdApp == null)
        {
            Process wdProcess = Process.Start("winword.exe");
            Timer t = new Timer();
            t.Start();
            while (wdApp == null && (t.Interval < 10000))
            {
                try
                {
                    //Es braucht Zeit, bis die Anwendung läuft und angesprochen werden kann.
                    wdApp = (wd.Application)Marshal.GetActiveObject("Word.Application");
                }
                catch { }
            }
            t.Stop();
            t = null;
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
    finally
    {
        if (wdApp != null) retVal = true;
    }
    return retVal;
}

```

Vergleichen Sie nun die Prozeduren *btnWordStart\_2\_Click* sowie *GetWordIntance\_2* in Listing 10.12. Falls Word noch nicht läuft, wird mit dem Schlüsselwort *new* eine neue Instanz der Word-Anwendung gestartet und gleichzeitig der Word-Objektvariablen *wdApp* zugewiesen. In diesem Fall startet Word unsichtbar und ohne neues Dokument, weshalb der Code diese Handlungen ausführt. Meistens wird diese Methode der Fernsteuerung verwendet.



Listing 10.12 Eine neue Word-Instanz über das Schlüsselwort *new* starten

```

private void btnWordStart_2_Click(object sender, EventArgs e)
{
    wd.Document wdDoc;
    if (GetWordInstance_2())
    {
        try
        {
            wdApp.Visible = true;
            wdDoc = wdApp.Documents.Add(ref missing, ref missing, ref missing, ref objTrue);
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
        }
    }
}

private bool GetWordInstance_2()
{
    bool retVal = false;
    try
    {
        Process[] wdPcs = Process.GetProcessesByName("WinWord");
        if (wdPcs.Length > 0)
        {
            pcs = wdPcs[0];
            wdApp = (wd.Application)Marshal.GetActiveObject("Word.Application");
        }
        if (wdApp == null)
        {
            //Die folgende Zeile startet Word und weist es der wdApp-Variabel zu
            wdApp = new wd.Application();
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
    finally
    {
        if (wdApp != null) retVal = true;
    }
    return retVal;
}

```

## COM-Anwendung-Ressourcen freigeben

In der Welt des COM-Entwicklers müssen Ressourcen explizit und bewusst verwaltet werden. Wird dies nicht getan, werden Anwendungen zu Ressourcen-Fressern, die über kurz oder lang Windows in die Knie zwingen.

Der .NET-Entwickler hat gelernt, sich auf die Funktionalität Garbage Collection von .NET Framework zu verlassen. Wurde ein Objekt freigestellt, sorgt diese dafür, dass früher oder später Windows

die Ressourcen wieder zur Verfügung stehen. Probleme können jedoch in der Schnittstelle .NET/COM auftauchen, da Garbage Collection die Ressourcen unter Umständen noch nicht freigegeben hat, wenn COM es erwartet.

Aus diesem Grunde ist es ratsam, die Garbage Collection zu bestimmten Zeitpunkten (bei Beendigung der fernsteuernden Anwendung beispielsweise) ausdrücklich auszuführen, und zwar zweimal in Folge, um sicherzugehen, dass alle »Überbleibsel« ausgeräumt werden, die aus Gründen des zeitlichen Ablaufs beim ersten Aufruf noch nicht beseitigt wurden. Das Listing 10.13 zeigt zwei Variationen auf. In der Prozedur *btnWordFreigeben\_Click* wird die Verbindung zur Word-Anwendung freigestellt. Falls Word noch läuft, merkt der Benutzer nichts – Word bleibt geladen. Die Prozedur *btnWordBeenden\_Click* hingegen beendet die Word-Anwendung, bevor die Ressourcen freigestellt werden.

---

**HINWEIS** Falls Sie mehr zu diesem Thema erfahren möchten, empfehlen wir das Microsoft Press-Buch »Microsoft .NET Development for Microsoft Office« von Andrew Whitechapel.

---

**Listing 10.13** COM-Ressourcen im .NET-Projekt freigeben

```
private void btnWordFreigeben_Click(object sender, System.EventArgs e)
{
    try
    {
        //Auf der Klassenebene deklarierte Variablen freigeben
        wdApp = null;
    }
    catch
    {
    }
    finally
    {
        GC.Collect();
        WaitForPendingFinalizers();
        GC.Collect();
        GC.WaitForPendingFinalizers();
        //Application.Exit(); //Für den Notfall
    }
}

private void btnWordBeenden_Click(object sender, EventArgs e)
{
    try
    {
        //Speicheraufforderung bei Bedarf einblenden lassen
        object objPromptSaveChanges = wd.WdSaveOptions.wdPromptToSaveChanges;
        wdApp.Quit(ref objPromptSaveChanges, ref missing, ref missing);
    }
    catch
    {
    }
    try
    {
        //Auf der Klassenebene deklarierte Variablen freigeben
        wdApp = null;
    }
    catch { }
    finally
    {
    }
}
```

Listing 10.13 COM-Ressourcen im .NET-Projekt freigeben (Fortsetzung)

```

{
    GC.Collect();
    GC.WaitForPendingFinalizers();
    GC.Collect();
    GC.WaitForPendingFinalizers();
}

```



Zu diesem Abschnitt sind Beispieldateien für C# (*Kap10\_CS.zip*) sowie VB.NET (*Kap10\_VB.zip*) vorhanden. Diese befinden sich auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap10*.

## VSTO 2005-Dokumentlösungen

Manchmal ist eine dokumentunabhängige Lösung wie eine *.exe*-Datei oder ein COM-Add-In genau das, was gesucht wird. Es gibt jedoch Aufgaben, für die eine dateiorientierte 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 bereitstellt.

Neben 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 2003 die notwendigen Zusätze, um eine VSTO-Lösung zu laden. Benutzern mit der Standard-Version von Office 2003 steht diese Funktionalität nicht zur Verfügung.

### HINWEIS

Visual Studio 2005 Tools for Office (läuft auf .NET Framework 2.0 sowie 3.0) wurde für die Zusammenarbeit mit Office 2003 Professional entwickelt. VSTO 2005-Dokumentlösungen laufen auch unter Office 2007 (alle Versionen, nicht ausschließlich Professional). Sie unterstützen jedoch weder die neuen RibbonX- noch die Custom Task Pane-Technologien. Diese werden erst Teil der nächsten Version von Visual Studio, die seit Ende 2007 erhältlich ist.

Die gegenwärtige Version 2005 hat mit ihrem Vorgänger (VSTO 2003) sehr wenig gemeinsam. Sie ist leistungsfähiger und entwicklerfreundlicher. Dokumentlösungen für Word enthalten einige wichtige Neuerungen gegenüber VSTO 2003:

- **Dokument-Entwurfsmodus.** Das mit der VSTO-Lösung verbundene Dokument wird in Visual Studio angezeigt und kann dort bearbeitet werden (siehe Abbildung 10.7).
- **Windows Forms-Steuerelemente.** Werden von VSTO in ein Office-gerechtes ActiveX-Format »gepackt« und bereitgestellt, so dass sie in ein Word-Dokument eingefügt werden können.
- **Dokumentaktionen-Aufgabenbereich.** Das VSTO-Team hat eine entwicklerfreundliche Schnittstelle für die Smart Document-Technologie bereitgestellt. Somit steht ein eigener, dokumentspezifischer Aufgabenbereich zur Verfügung, der auch Windows Form Controls enthalten kann.
- **Data Binding.** Daten mit einer externen Datenquelle dynamisch verbinden und im Dokument (über Textmarken-Steuerelemente oder XML-Nodes) anzeigen.
- **Data Caching.** Daten können in einem Zwischenspeicher im Dokument in XML-Format gelesen und geschrieben werden, ohne das Dokument in Word öffnen zu müssen.
- **Dokumentspezifische Smarttags.** Begriffe werden nur im VSTO-Dokument erkannt und entsprechende Handlungen im Smarttag-Kontextmenü zur Verfügung gestellt.

Eine eingehende Diskussion der VSTO-Technologie würde ein ganzes Buch füllen. Deshalb bietet der folgende Abschnitt lediglich einen Überblick einiger Facetten der Technologie, mit Beschreibung der ersten Schritte, um den Einstieg zu erleichtern.

**HINWEIS**

Visual Studio 2005 Tools for Office ist im Lieferumfang von Visual Studio 2005 Team Suite enthalten, kann aber auch als separates Produkt erworben werden.

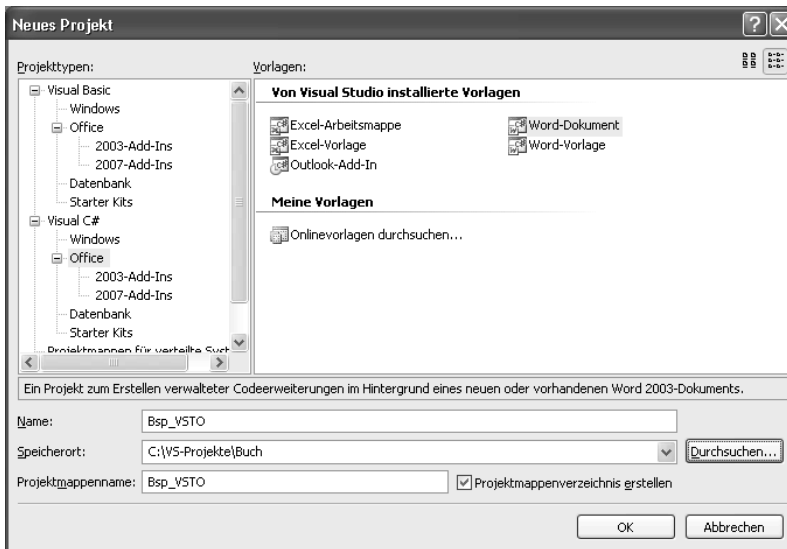
VSTO 2005 Second Edition, die Werkzeuge für die Entwicklung von Office Add-Ins bereitstellt, kann kostenlos von der Microsoft-Website unter <http://www.microsoft.com/downloads/details.aspx?familyid=F5539A90-DC41-4792-8EF8-F4DE62FF1E81&displaylang=de> heruntergeladen werden. Voraussetzung ist Visual Studio 2005 Professional oder höher. VSTO 2005 SE wird im Abschnitt »VSTO COM-Add-Ins« vorgestellt.

Bitte beachten Sie, dass Visual Studio 2005-Produkte sprachspezifisch sind. Es ist beispielsweise nicht möglich, die englische Version von VSTO 2005 SE zusammen mit der deutschen Version von Visual Studio 2005 Professional zu installieren. Achten Sie also darauf, dass immer die korrekte Sprachversion heruntergeladen wird.

## Eine VSTO-Lösung anlegen

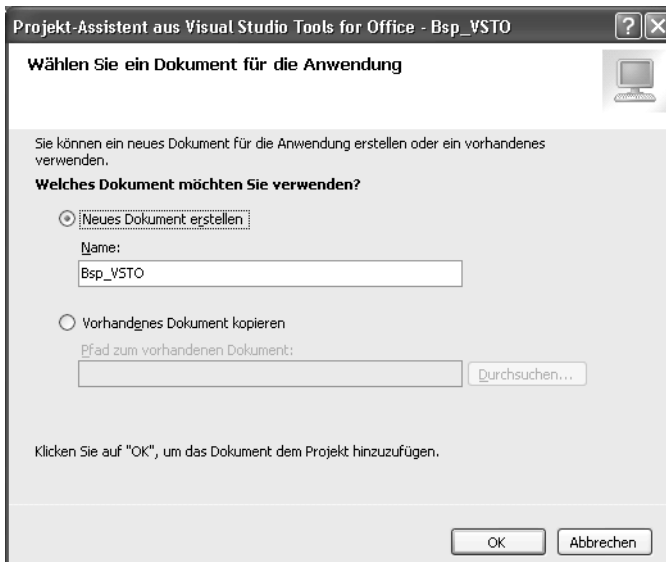
Nach erfolgreicher Installation von Visual Studio 2005 Tools for Office erscheinen unter den Visual Basic- und C#-Ordnern des Visual Studio-Dialogfelds *Neues Projekt* Ordner für *Office*, wie in Abbildung 10.4 ersichtlich.

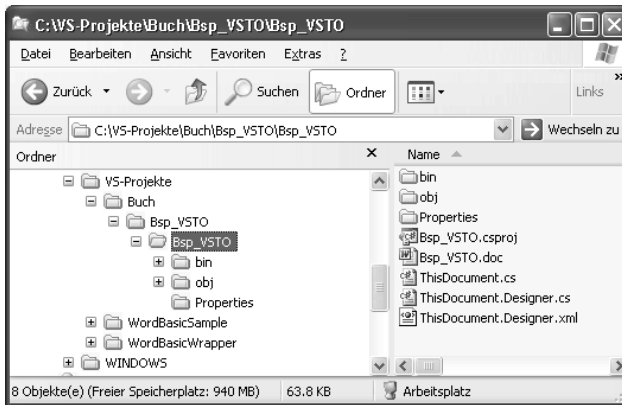
Abbildg. 10.4 Visual Studio 2005-Projektvorlagen mit installiertem VSTO 2005 sowie VSTO 2005 SE



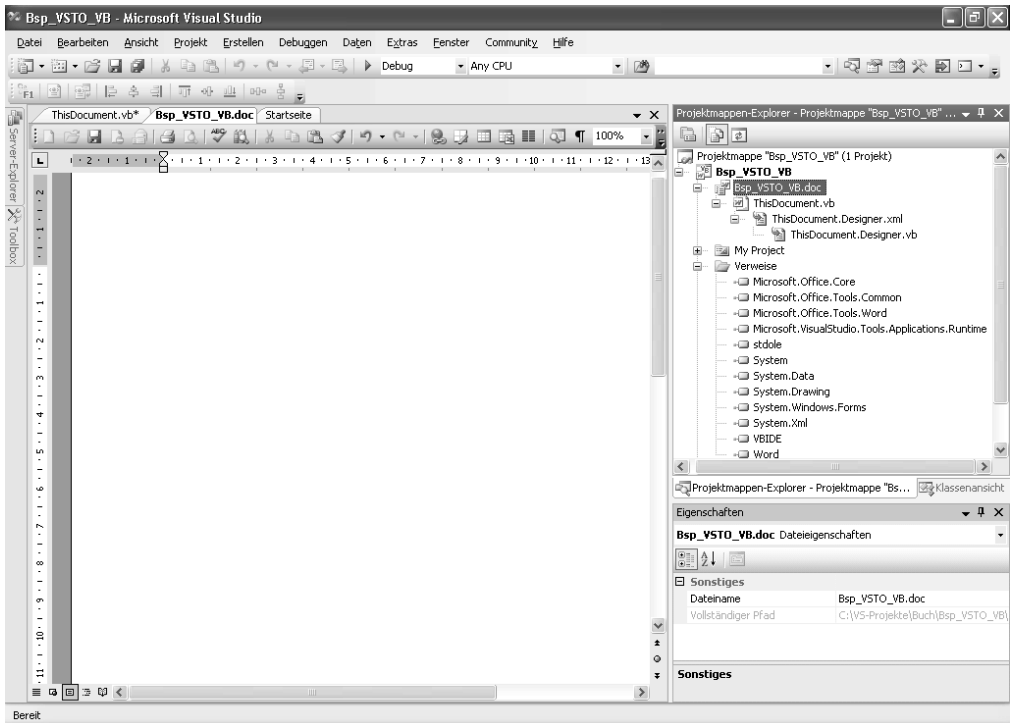
Nach Auswahl eines Word-Eintrags (*Word-Dokument* bzw. *Word-Vorlage*) wird das Dialogfeld aus Abbildung 10.5 eingeblendet. Standardmäßig wird angeboten, ein neues Dokument zu erstellen. Soll stattdessen das Projekt auf einem bestehenden basieren, muss die Option *Vorhandenes Dokument kopieren* aktiviert werden. Wie der Beschriftung zu entnehmen ist, bleibt die ursprüngliche Datei unangetastet; die von VSTO erstellte Kopie befindet sich im Ordner mit weiteren, zum Projekt gehörenden Dateien (siehe Abbildung 10.6).

Abbildg. 10.5 Festlegen, ob das Projekt auf einem neuen oder vorhandenen Dokument basieren soll



**Abbildg. 10.6** Von VSTO angelegte Ordner-Struktur und Projekt-Dateien, inklusive des Word-Dokuments


Nach einigen Sekunden erscheint im Visual Studio-Anwendungsfenster der VSTO Designer (der Entwurfsmodus) samt Word-Dokument, dargestellt in Abbildung 10.7. Die Menüleiste enthält zusätzliche Menüpunkte: Darüber stehen alle Word-Befehle zur Verfügung, die in einer VSTO-Lösung einsetzbar sind. Gemeinsame Menünamen werden zusammengefasst. Beispielsweise befindet sich unter *Extras* ein Menüeintrag *Microsoft Office Word Extras*, worunter das *Extras*-Menü von Word zu finden ist.

**Abbildg. 10.7** Neu in VSTO 2005: Das Word-Dokument kann in Visual Studio bearbeitet werden


Hinter den Kulissen hat VSTO ein unsichtbares ActiveX-Steuerelement der OLE-Klasse `VSTO.StorageRuntime.1` als Mitglied der Shapes-Auflistung ins Dokument eingefügt. Es ist für die Verwaltung der VSTO-Funktionalität im Word-Dokument verantwortlich.

**HINWEIS** Mehr Informationen zum »Runtime Storage Control« finden Sie auf der Seite [http://msdn2.microsoft.com/en-us/library/7ydwehbf\(VS.80\).aspx](http://msdn2.microsoft.com/en-us/library/7ydwehbf(VS.80).aspx).

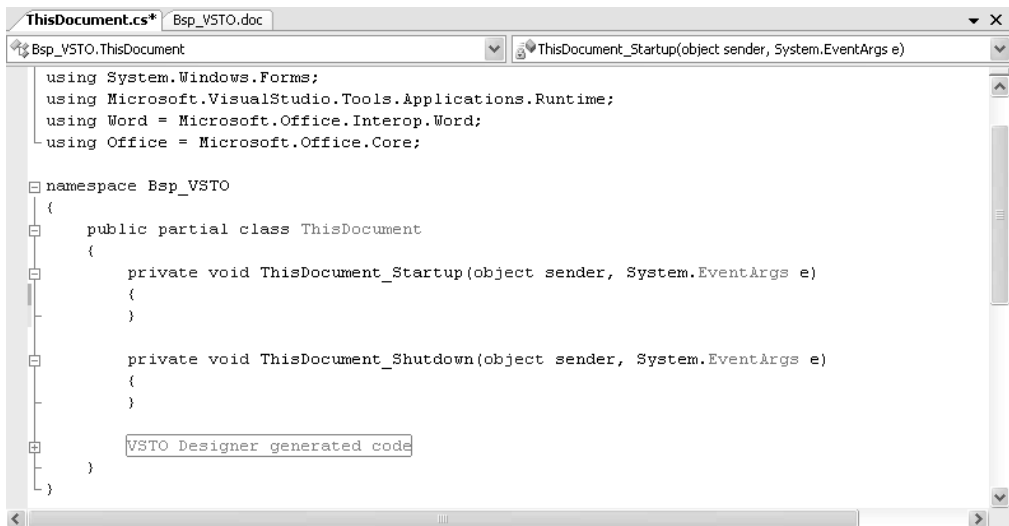
## Der Code »hinter dem Dokument«

Statt VBA-Code im Dokument zu speichern, wird in einer VSTO-Lösung der Code getrennt aufbewahrt. Als Verbindung dient die *ThisDocument*-Datei (mit der Dateiendung `.cs` für C# bzw. `.vb` für Visual Basic) mit der Teilklasse *ThisDocument*. Sie enthält nur den Code, der für die Zusammenarbeit mit dem Word-Dokument benötigt wird. Wie in Visual Studio 2005 üblich, werden die vom Visual Studio Designer erstellten Support-Strukturen durch weitere (in Abbildung 10.7 ersichtliche) Dateien aufrechterhalten. Im Alltag muss sich der VSTO-Entwickler selten damit befassen, für einen vertiefenden Einblick in die Wirkungsweise von VSTO bieten diese Ergänzungsdateien jedoch eine interessante Lektüre.

Wie die Abbildung 10.7 zudem veranschaulicht, erledigt VSTO einige Aufgaben für den Entwickler. Verweise zu den benötigten .NET-DLL und COM-Bibliotheken (PIAs) werden erstellt und die C#-Schnittstelle fügt benötigte *using*-Befehlszeilen ein (der VB-Entwickler muss gewünschte Imports-Zeilen selbst eingeben). Außerdem definiert VSTO die zwei Prozeduren *ThisDocument\_Startup* sowie *ThisDocument\_Shutdown* (Abbildung 10.8). Diese werden beim Starten bzw. beim Schließen des Dokuments ausgeführt: Hier ist der Ansatzpunkt für jede VSTO-Lösung.

**TIPP** Die Code-Ansicht kann über den entsprechenden Befehl im Kontextmenü des *ThisDocument*-Eintrags im Projektmappen-Explorer eingeblendet werden.

Abbildg. 10.8 Der von VSTO generierte Code »hinter dem Dokument« (C#-Version)



## Das VSTO-Dokument vorbereiten

Falls das VSTO-Dokument oder die VSTO-Vorlage mit Standardtext oder -inhalt bestückt werden soll, kann die Bearbeitung wahlweise in Word oder im VSTO-Entwurfsmodus erfolgen. Allfällige WinForms- oder VSTO-Steuerelemente (einzig das Textmarken-Steuerelement steht für Word-Lösungen bereit) müssen jedoch im VSTO-Entwurfsmodus hinzugefügt werden.

Als Beispiel für unseren Überblick dient ein Firmenbrief. Die Eingabestellen für Standardangaben wie Empfängeradresse, Betreffzeile und Anrede werden mit VSTO-Textmarken-Steuerelementen gekennzeichnet. Der Benutzer kann diese anklicken und den Text direkt in das Dokument eingeben. Er kann sich auch des Aufgabenbereichs bedienen, der aus der Firmendatenbank eine Liste mit Kundenadressen bereitstellt, woraus er nach Belieben einen Eintrag wählen kann. Der Aufgabenbereich wird mit einer eigens für diese Lösung erstellten Symbolleiste ein- und ausgeblendet.

Die in den Textmarken enthaltenen Daten werden in den Daten-Cache (Dokumentzwischenspeicher) geschrieben, wo sie für die VSTO-Technologie auch bei geschlossenem Dokument lesbar sind. Dazu dient das `ServerDocument`-Objekt der VSTO Runtime.

Ferner zeigt das Beispiel, wie der Benutzer eine VSTO-Dokumentanpassung entfernen kann, falls das Dokument an jemanden weitergeleitet wird, der keinen Zugang zur VSTO-Lösung hat.

### HINWEIS

Im Gegensatz zu den Code-Beispielen für die Word- und Office-Objektmodelle wird in den folgenden Teilen VB.NET verwendet. Da die von VSTO zur Verfügung gestellten Eigenschaften und Methoden den Regeln von .NET-Framework entsprechen, erfolgt die Umwandlung in C# problemlos. Hingegen wird die Leserschaft, die aus der VBA-Ecke kommt, den VB-Code-Beispielen besser folgen können, was den Einstieg in VSTO etwas erleichtern wird.

## VSTO-Textmarken-Steuerelemente

VSTO-Textmarken-Steuerelemente bieten gegenüber gewöhnlichen Textmarken zwei wichtige Vorteile: Sie können mit einer Datenquelle und einem Daten-Cache verbunden werden, und sie stellen fünf Ereignisse bereit. Diese Ereignisse basieren auf Word-Anwendungsereignissen:

- `BeforeDoubleClick` (vor einem Doppelklick)
- `BeforeRightClick` (vor einem Rechtsklick)
- `Deselected` (der Fokus verlässt den Textmarkenbereich)
- `Selected` (der Fokus tritt in den Textmarkenbereich ein)
- `SelectionChange` (der Fokus im Textmarkenbereich wird durch Bedienen der Maus oder der Tastatur geändert)

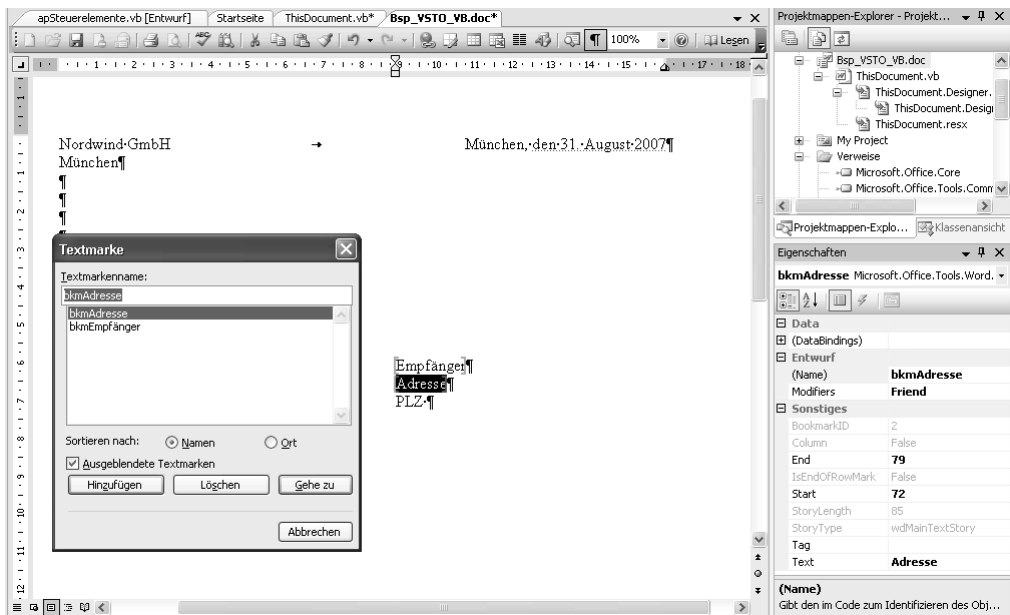
Textmarken-Steuerelemente können entweder über die Befehlsfolge *Einfügen/Textmarke* (Abbildung 10.9) oder über die Visual Studio-Toolbox (Abbildung 10.10) eingefügt werden. Egal, welche Methode gewählt wurde, können Namen sowie Textmarkenbereich nachträglich ganz einfach über das Eigenschaftenfenster geändert werden. Um den Namen zu ändern, genügt es, irgendwo im Textmarkenbereich zu klicken. Um den Bereich zu ändern, muss dieser zuerst markiert werden (mindestens ein Zeichen des gegenwärtigen Bereichs muss sich innerhalb der Markierung befinden).

### ACHTUNG

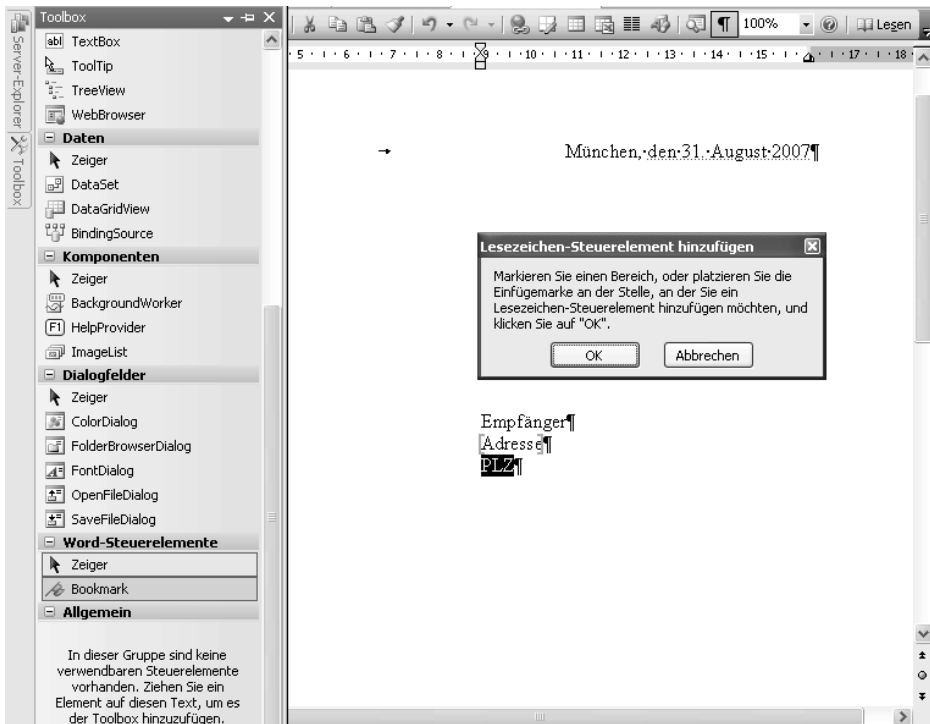
Wenn Sie auf ein VSTO-Textmarke-Steuerelement doppelklicken, wird eine Klick-Ereignis-Prozedur in der Code-Ansicht erstellt. Eine Markierung muss also mit der Tastatur oder durch Klicken und Ziehen erfolgen.



Abbildg. 10.9 Ein Textmarken-Steuerelement über Words internen Einfügen/Textmarke-Befehl einfügen



Abbildg. 10.10 Ein Textmarken-Steuerelement aus der Visual Studio-Toolbox einfügen



Textmarken sind bekanntlich äußerst »fragil«: Sie sind schnell gelöscht, ohne dass der Benutzer es merkt. VSTO-Textmarken-Steuerelemente sind leider genauso anfällig. Um diesem Problem entgegenzuwirken, bedient sich die Beispiel-Lösung der *Select*- sowie der *Deselect*-Ereignisse, um dem Benutzer den Umgang mit den Textmarken zu erleichtern. Der Code dafür befindet sich in Listing 10.14.

Bewegt der Benutzer die Einfügemarke in ein Textmarken-Steuerelement, wird dessen *Select*-Ereignis ausgelöst, das über das Argument *e* die neue Markierung (*Selection*) bereitstellt. So wird die Textmarke ermittelt und in einer globalen Variablen (*selectedBkm*) festgehalten. Damit der Benutzer mit der Texteingabe sofort beginnen kann, wird der Textmarkenbereich markiert. Anschließend wird der Bereich um ein Zeichen erweitert (die Markierung ändert sich dadurch nicht), so dass die Benutzereingabe das letzte Zeichen der Textmarke nicht erfasst und diese folglich nicht entfernt.

**HINWEIS**

Die Codezeile *selectedBkm.End = selectedBkm.End + 1* in Listing 10.14 zeigt noch einen Vorteil auf, den VSTO bietet. Eine Word-Textmarke hat keine *Start*- und *End*-Eigenschaften, diese gehören dem *Range*-Objekt. VSTO überträgt diese auf das Textmarken-Steuerelement und spart damit dem Entwickler einige Tastenschläge. Ähnlich steht es um die *Select*-Methode (*selectedBkm.Select()*).

Wichtig ist natürlich, dass nach der Bearbeitung der Textmarkenbereich zurückgesetzt wird, sonst würde er bei jedem Eintritt in das Textmarken-Steuerelement um ein weiteres Zeichen wachsen. Hierfür verantwortlich zeichnet das *Deselect*-Ereignis.

Problematisch bei diesem Vorgang ist die Reihenfolge, in der die beiden Ereignisse ausgelöst werden. Wenn der Benutzer zwischen Textmarken wechselt, müssen zwei Textmarken verwaltet und deren Bereiche korrekt erweitert bzw. gekürzt werden. Es stellt sich heraus, dass diese Reihenfolge nicht vorhersehbar ist: Oft wird zuerst *Deselect* ausgelöst, aber nicht immer. Daher verwendet der Code drei zusätzliche, globale Variablen: *deselectedBkm* hält die Textmarke, die es zurückzusetzen gilt, fest; *selectedAusgeführt*, ob das Ereignis *Select* schon ausgelöst wurde; während *deselectedAusgeführt* den Status des Ereignisses *Deselect* festhält. Die Ereignisse fragen den Stand dieser vier Variablen ab, um den richtigen Zeitpunkt der Ausführung von *deselectedBkm = selectedBkm* zu berechnen.

**Listing 10.14** Den Umgang mit Textmarken mit den *Select*- sowie *Deselect*-Ereignissen erleichtern

```
Dim selectedBkm As Word.Bookmark = Nothing
Dim deselectedBkm As Word.Bookmark = Nothing
Dim selectedAusgeführt As Boolean = False
Dim deselectedAusgeführt As Boolean = False

Private Sub bkmSelect(ByVal sender As System.Object, _
    ByVal e As Microsoft.Office.Tools.Word.SelectionEventArgs) _
    Handles bkmAdresse.Selected, bkmEmpfänger.Selected, bkmPLZ.Selected, _
    bkmLand.Selected, bkmOrt.Selected, bkmRegion.Selected, bkmBetreff.Selected, _
    bkmAnrede.Selected

    'Die Textmarke in einer globalen Variablen festhalten.
    selectedBkm = e.Selection.Bookmarks(1)
    'Den Inhalt der gewählten Textmarke markieren
    selectedBkm.Select()
    'Die Textmarke bis zum Endpunkt des Bereichs erweitern.
    selectedBkm.End = selectedBkm.End + 1
```

Listing 10.14 Den Umgang mit Textmarken mit den *Select*- sowie *Deselect*-Ereignissen erleichtern (Fortsetzung)

```

If Not selectedAusgefuehrt And Not deselectedAusgefuehrt Then
    If deselectedBkm Is Nothing Then
        'Eine Textmarke wird erstmals markiert
        deselectedBkm = selectedBkm
    Else
        'Sonst wurde Select vor Deselect ausgefuehrt
        deselectedAusgefuehrt = False
        selectedAusgefuehrt = True
    End If
ElseIf selectedAusgefuehrt = False And deselectedAusgefuehrt = True Then
    'Deselect wurde zuerst ausgefuehrt
    selectedAusgefuehrt = False
    deselectedAusgefuehrt = False
    deselectedBkm = selectedBkm
Else
    selectedAusgefuehrt = True
End If
End Sub

Private Sub bkmDeselect(ByVal sender As System.Object, _
    ByVal e As Microsoft.Office.Tools.Word.SelectionEventArgs)
    Handles bkmEmpfänger.Deselected, bkmAdresse.Deselected, _
    bkmOrt.Deselected, bkmRegion.Deselected, bkmLand.Deselected, _
    bkmBetreff.Deselected, bkmAnrede.Deselected

    'Den Textmarkenbereich um ein Zeichen kürzen, so dass sie
    'den Text enthält, den der Benutzer eingegeben hat.
    deselectedBkm.End = deselectedBkm.End - 1

    If selectedAusgefuehrt = True Then
        'Select wurde zuerst aufgefuehrt
        Me.deselectedBkm = selectedBkm
        selectedAusgefuehrt = False
        deselectedAusgefuehrt = False
    ElseIf selectedAusgefuehrt = False Then
        'Deselect wird zuerst ausgefuehrt
        deselectedAusgefuehrt = True
    End If
End Sub

```

## Die Benutzerschnittstellen einer VSTO-Lösung

Ob VBA-Makros oder VSTO-Code: Wir erwarten, dass der Benutzer interaktiv mit der programmierten Lösung zusammenarbeitet. Neben den Dialogfeldern und Office-Symbolleisten (CommandBars) steht dem VSTO-Entwickler für dokumentspezifische Lösungen der Aufgabenbereich *Dokumentaktionen* als zusätzliche Schnittstelle zur Verfügung.

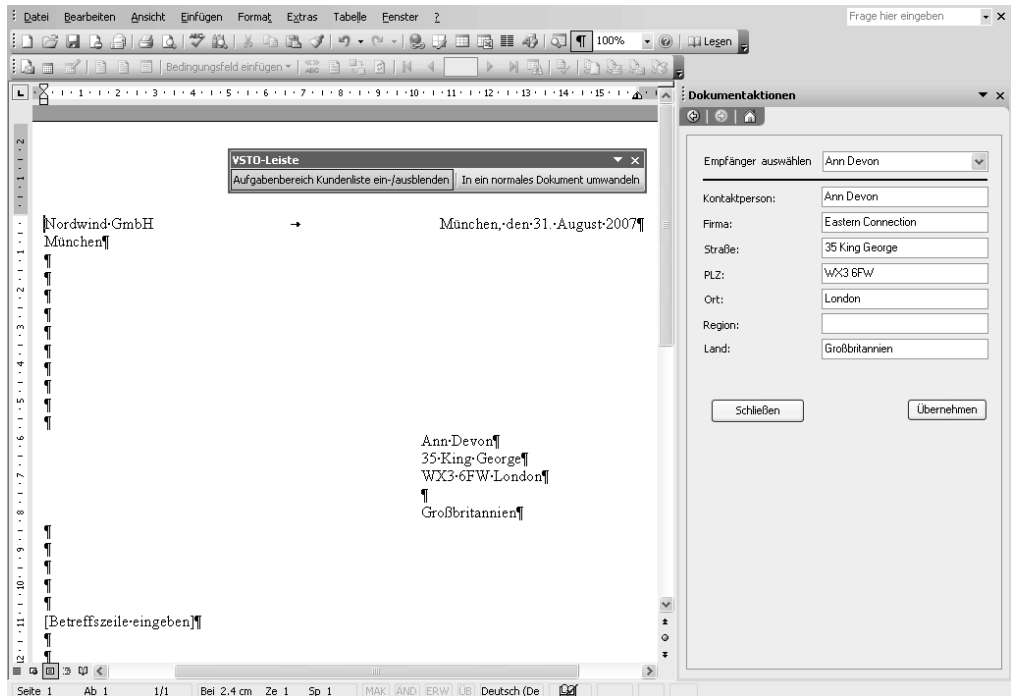
Eine weitere Methode, auf Benutzerhandlungen zu reagieren, bilden die Word- und VSTO-Ereignisse, wie jene für Textmarken, die im vorherigen Abschnitt vorgestellt wurden.

Egal, welche Schnittstellen der Entwickler wählt, die Grundlagen dafür werden in der Prozedur *ThisDocument\_Startup* gelegt. Und wo erforderlich, werden sie in der Prozedur *ThisDocument\_Shutdown* außer Kraft gesetzt.

Die Beispiellösung zeigt auf, wie eine Symbolleiste mit zwei Symbolschaltflächen sowie ein *Dokumentaktionen*-Aufgabenbereich angelegt werden, wie in Abbildung 10.11 zu sehen. Die erste Symbolschaltfläche schaltet den Aufgabenbereich ein und aus. Die zweite entfernt die VSTO-Lösung aus dem Dokument, so dass Word-Benutzer ohne Zugang zur Lösung störungsfrei damit arbeiten können.

Dieser Aufgabenbereich ermöglicht die Auswahl einer Kundenadresse aus der Firmendatenbank und deren nachträgliche Bearbeitung, bevor die Informationen in die Textmarken eingefügt werden.

**Abbildg. 10.11** VSTO-Dokumentlösung mit Symbolleiste und *Dokumentaktionen*-Aufgabenbereich



## Symbolleisten und Symbolschaltflächen anlegen

Die Handhabung der CommandBar-Objekte und -Eigenschaften von Office in VSTO unterscheidet sich nicht wesentlich von jener unter VBA (siehe Kapitel 16). Das Listing 10.15 zeigt die Prozedur *ThisDocument\_Startup*, die beim Laden des Dokuments ausgeführt wird. Standardmäßig gilt das VSTO-Dokument (im Beispiel die Dokumentvorlage) als Speicherort (CustomizationContext) für alle im Code vorgenommenen Änderungen; es ist nicht notwendig, diesen ausdrücklich anzugeben.

Alle Symbolleistenobjekte werden auf Klassenebene als globale Variablen deklariert. Somit haben sie Gültigkeit, bis das Projekt aus dem Speicher entfernt wird, und werden nicht vorzeitig durch Garbage Collection entfernt. Falls weitere Klassen in der VSTO-Lösung darauf zugreifen, sollen sie als »Friend« oder »Public« deklariert werden.

Die OnAction-Eigenschaft, die eine Symbolschaltfläche mit ihrem VBA-Makro verbindet, ist außerhalb einer VBA-Lösung nicht wirksam. Stattdessen wird der Symbolschaltfläche ein Click-Ereignis

zugewiesen. Dieses wird ausdrücklich über AddHandler-Codezeilen mit der auszuführenden Prozedur verbunden:

```
AddHandler cbBtnAufgabenbereich.Click, AddressOf AufgabenBereichEinAus
AddHandler cbBtnInNormalesDokWandeln.Click, AddressOf InNormalesDokWandeln
```

Da das Anlegen von Symbolleisten Änderungen in die VSTO-Vorlage schreibt, wird Word den Benutzer irgendwann fragen, ob er diese speichern möchte. Um solche Meldungen zu unterbinden, wird am Schluss der Prozedur die Saved-Eigenschaft der Vorlage auf »Wahr« gesetzt: `ThisTemplate.Saved = True`. Auch Speicheraufforderungen wegen Änderungen im Dokument selbst werden ebenso wie für die Vorlage *Normal.dot* im Beispielcode unterbunden. Möglicherweise müssen Sie diese Zeilen wiederholt in der Lösung verwenden, um die lästigen Meldungen zu vermeiden.

**Listing 10.15** Eine Symbolleiste in *ThisDocument\_Startup* erstellen

```
Imports System.Windows.Forms
Imports System.Data
Imports Word = Microsoft.Office.Interop.Word
Imports Tools = Microsoft.Office.Tools
Imports Office = Microsoft.Office.Core

'Globale Variablen
Dim cb As Office.CommandBar 'Symbolleiste
Friend cbBtnAufgabenbereich As Office.CommandBarButton 'Symbolschaltfläche
Dim cbBtnInNormalesDokWandeln As Office.CommandBarButton

Private Sub ThisDocument_Startup(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles Me.Startup

    Try
        'Die VSTO-Symbolleiste erstellen...
        Me.cb = ThisApplication.CommandBars.Add(Name:="VSTO-Leiste", _
            Position:=Office.MsoBarPosition.msoBarFloating, MenuBar:=False)
        '...und mit Schaltflächen bestücken
        Me.cbBtnAufgabenbereich = CType(cb.Controls.Add(
            Type:=Office.MsoControlType.msoControlButton), Office.CommandBarButton)
        With Me.cbBtnAufgabenbereich
            .Style = Microsoft.Office.Core.MsoButtonStyle.msoButtonCaption
            .Caption = "Aufgabenbereich Kundenliste ein-/ausblenden"
            .Tag = "AufgabenbereichEinAus"
            .State = Microsoft.Office.Core.MsoButtonState.msoButtonUp
            .Visible = True
        End With
        Me.cbBtnInNormalesDokWandeln = CType(cb.Controls.Add(
            Type:=Office.MsoControlType.msoControlButton), Office.CommandBarButton)
        With Me.cbBtnInNormalesDokWandeln
            .Style = Microsoft.Office.Core.MsoButtonStyle.msoButtonCaption
            .Caption = "In ein normales Dokument umwandeln"
            .Tag = "InNormalesDokumentWandeln"
            .Visible = True
            .BeginGroup = True
        End With
        Me.cb.Visible = True

        AddHandler cbBtnAufgabenbereich.Click, AddressOf AufgabenBereichEinAus
```

**Listing 10.15** Eine Symbolleiste in *ThisDocument\_Startup* erstellen (Fortsetzung)

```

AddHandler cbBtnInNormalesDokWandel.Click, AddressOf InNormalesDokWandel

'Sicherstellen, dass das Dokument in der korrekten Ansicht erscheint
Me.ActiveWindow.View.Type = Word.WdViewType.wdPrintView
'dass diese Symbolleiste nicht sichtbar ist,
ThisApplication.CommandBars("Control Toolbox").Visible = False
'dass die Textmarken nicht sichtbar sind und,
Me.ActiveWindow.View.ShowBookmarks = False
'dass keine Speicheraufforderung erscheint, falls
'der Benutzer das Dokument sofort schließt

'Die Daten einlesen
kundenDaten = New KundenListe
kundenDaten.ReadXml(Me.AttachedTemplate.Path & "\KundenDaten.xml")

'Speicheraufforderungen unterbinden
ThisTemplate = Me.AttachedTemplate
ThisTemplate.Saved = True
Me.Saved = True
Catch ex As System.Exception
    MessageBox.Show(ex.Message)
End Try

InitializeDataCache()
ThisApplication.NormalTemplate.Saved = True
End Sub

```

## Der Aufgabenbereich *Dokumentaktionen*

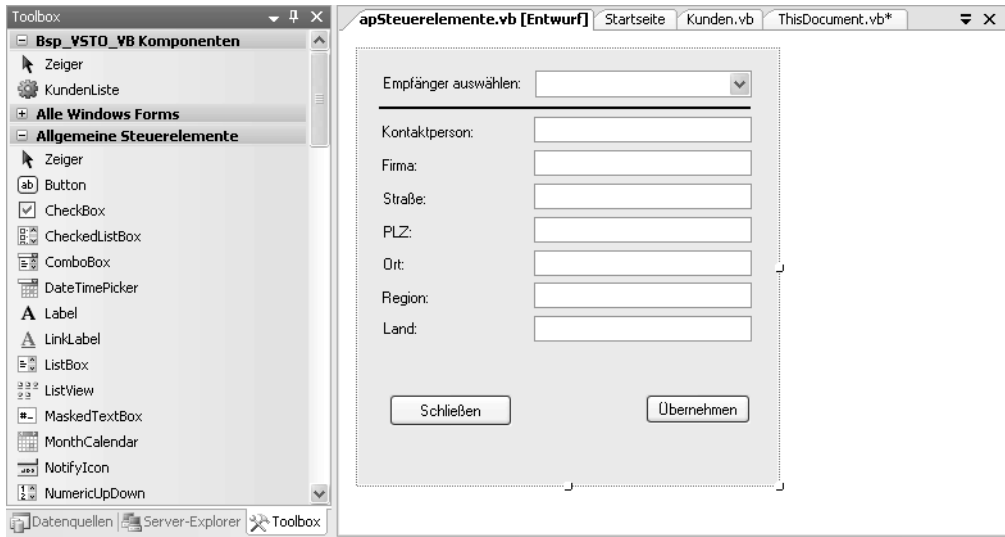
Der von VSTO zur Verfügung gestellte Aufgabenbereich *Dokumentaktionen* ist eigentlich Teil der Smart Document-Funktionalität (siehe Kapitel 24). Die Nachfrage seit Erscheinen von Office XP nach einem frei definierbaren Aufgabenbereich war jedoch so groß, dass das VSTO-Team diese Funktionalität integrierte, um VSTO-Entwicklern einen Aufgabenbereich zur Verfügung zu stellen. VSTO übernimmt die mühsame Arbeit, ein XML-Schema und -Manifest zu erstellen, diese mit dem Dokument zu verbinden und sie dynamisch zu verwalten. Der Entwickler kann sich voll auf die Gestaltung und die programmtechnischen Aspekte der Schnittstelle konzentrieren.

Neben HTML-Steuerelementen unterstützt der VSTO-Aufgabenbereich auch WinForms-Steuerelemente. VSTO packt diese in Aufgabenbereich-gerechte COM-ActiveX-Steuerelemente ein. Obwohl es möglich ist, zur Laufzeit einzelne WinForms-Steuerelemente dem Aufgabenbereich hinzuzufügen, stellen die meisten Entwickler bald fest, dass diese Vorgehensweise etwas mühsam ist. Die Positionierungs- sowie Formatierungsmöglichkeiten sind beschränkt und das Ergebnis ist oft wenig zufrieden stellend.

Optimaler ist es, ein WinForms-Benutzersteuerelement im Projekt zu erstellen und dieses in den Aufgabenbereich einzubinden. Damit entfällt die Definition mehrerer einzelner Steuerelemente bei Laufzeit – sie können im Entwurfsmodus erstellt und formatiert werden. Ein Benutzersteuerelement fügen Sie dem Projekt über die Befehlsfolge *Projekt/Benutzersteuerelement hinzufügen* hinzu. Im Dialogfeld *Neues Element hinzufügen* stellen Sie sicher, dass der Eintrag *Benutzersteuerelement* markiert ist, und geben ihm dann einen aussagekräftigen Namen. Nach Betätigung der Schaltfläche *Hinzufügen* erscheint die leere Entwurfsfläche, die wie ein Windows-Formular mit Steuerelementen aus

der Toolbox ergänzt wird (Abbildung 10.12). Auch das Eigenschaftensfenster steht wie üblich bei der Arbeit mit WinForms zur Verfügung.

Abbildg. 10.12 Das Benutzersteuerelement für den VSTO-Aufgabenbereich



Der Code, der den Aufgabenbereich initialisiert und einblendet, ist in Listing 10.16 ersichtlich. Da diese Handlungen ebenfalls Speicheraufforderungen für die *Normal.dot* verursachen, enthält die Prozedur Code, um den erwähnten Meldungen zuvorzukommen. Hat der Benutzer jedoch eigene Anpassungen vorgenommen, sollen diese nicht verworfen werden. Deshalb wird am Anfang der Status der Saved-Eigenschaft festgehalten und am Ende die Saved-Eigenschaft entsprechend angewendet oder nicht.

Bei der Einblendung des Aufgabenbereichs müssen drei Objekte berücksichtigt werden: das im Word-Aufgabenbereich eingebettete VSTO-Steuerelement, die der Lösung dienenden Steuerelemente (das Benutzersteuerelement) sowie der eigentliche Word-Aufgabenbereich. Diese werden in den globalen Variablen *ap*, *apControls* beziehungsweise *tp* festgehalten. Im Normalfall müssen sie alle nur einmal initialisiert werden. Deshalb testet *AufgabenbereichEinAus* zuerst, ob *ap* sowie *tp* gleich *Nothing* sind. Entsprechend wird die Prozedur *AufgabenbereichInitialisieren* aufgerufen.

Da einer VSTO-Dokumentlösung nur ein möglicher Aufgabenbereich zur Verfügung steht, existiert dieser schon, wir müssen ihn lediglich ansprechen: *Me.ap = Me.ActionsPane*. Beim Benutzersteuerelement hingegen handelt es sich um eine Klasse, von der ein neues Objekt erstellt werden muss: *Me.apControls = New apSteuerelemente*. Nach Instanziierung des Benutzersteuerelements kann es dem VSTO-Aufgabenbereich hinzugefügt werden: *Me.ap.Controls.Add(apControls)*. Schließlich wird der Word-Aufgabenbereich seiner Variablen zugewiesen: *Me.tp = ThisApplication.TaskPanes(Word.WdTaskPanes.wdTaskPaneDocumentActions)*. Anschließend ruft die Prozedur eine weitere auf, um die Steuerelemente mit Daten zu füllen. Darauf kommen wir im nächsten Abschnitt zurück.

**PROFITIPP**

In einer VSTO-Dokumentlösung kann auf die Word- (oder Excel-)Anwendung über das Objekt `ThisApplication` direkt zugegriffen werden. `Me` (oder in C# `this`) stellt die VSTO-Lösung dar, die ihrerseits die meisten Dokumentmethoden und -Eigenschaften zur Verfügung stellt (implementiert). Solche sind jedoch nicht mit den »echten« Eigenschaften und Methoden zu verwechseln und verhalten sich gelegentlich anders, was zu unerwarteten Ergebnissen führt. Falls Sie ein Word-Objekt direkt ansprechen möchten, geht das über die Eigenschaft `InnerObject`. Um beispielsweise das Word-Dokument (statt die VSTO-Lösung) anzusprechen: `Dim wdDoc As Word.Document = Me.InnerObject`. Als weiteres Beispiel sehen Sie die Prozedur `FillBookmark` in Listing 10.18.

Da die Initialisierung des VSTO-Aufgabenbereichs ihn auch einblendet, testet der Code, ob dies gerade erfolgt ist. Falls nicht, wird er »getoggelt«. Danach wird durch die Prozedur `ToggleButtonState` der Status der Symbolschaltfläche entsprechend festgelegt: Falls der Aufgabenbereich eingeblendet ist, sieht sie »gedrückt« aus, sonst nicht.

Letztlich wird der Fokus in den eingeblendeten Aufgabenbereich gesetzt. Die Methode `SetFocus` aktiviert lediglich seine Titelleiste. Um ein darin enthaltenes Steuerelement anzuspringen, steht nur `SendKeys` zur Verfügung, was nicht gerade »schön« ist, aber es funktioniert (meistens).

**Listing 10.16** Die Betätigung der entsprechenden Symbolschaltfläche führt diese Prozedur aus

```
'Globale Variablen auf Klassenebene
Dim ap As Tools.ActionsPane          'Dokumentaktionen-Aufgabenbereich
Dim apControls As apSteuerelemente  'Benutzerdefiniertes Steuerelement
Friend tp As Word.TaskPane           'allgemeiner Word-Aufgabenbereich

Private Sub AufgabenBereichEinAus(ByVal ctrl As Office.CommandBarButton, _
    ByRef Cancel As Boolean)
    'Hält fest, ob die Normal.dot nicht gespeicherte Daten enthält.
    Dim normalTemplateSaved As Boolean = True
    Dim NormalDot As Word.Template = ThisApplication.NormalTemplate
    Dim apInitialisiert As Boolean = False
    Try
        'Nach Ausführung dieser Prozedur gibt es Änderungen in der NormalTemplate.
        'Falls bislang keine vorhanden sind, die gespeichert werden sollen,
        'wird am Schluss die Abfrage nach Speicherung von Änderungen unterdrückt.
        normalTemplateSaved = NormalDot.Saved

        'Falls der Aufgabenbereich Dokumentaktionen noch nicht initialisiert wurde ...
        If Me.ap Is Nothing Or Me.tp Is Nothing Then
            '...dies tun, ihm das Steuerelement zufügen, und anzeigen.
            apInitialisiert = AufgabenBereichInitialisieren()
        End If
        'Wurde der Aufgabenbereich gerade initialisiert ist er sichtbar
        'und soll nicht umgehend wieder unsichtbar gemacht werden.
        If Not apInitialisiert Then tp.Visible = Not tp.Visible

        'Die Symbolschaltfläche der Sichtbarkeit des Aufgabenbereichs anpassen
        ToggleButtonState(cbBtnAufgabenbereich)

        'Dem Aufgabenbereich den Fokus geben
        If Me.tp.Visible Then
            ThisApplication.CommandBars("Task Pane").Controls(1).SetFocus()
            System.Windows.Forms.SendKeys.Send("{Down}")
        End If
    End Try
End Sub
```



Listing 10.16 Die Betätigung der entsprechenden Symbolschaltfläche führt diese Prozedur aus (Fortsetzung)

```

Catch ex As Exception
    MessageBox.Show(ex.Message)
Finally
    If normalTemplateSaved Then NormalDot.Saved = True
    ThisTemplate.Saved = True
End Try
End Sub

Friend Function AufgabenbereichInitialisieren() As Boolean
    Dim success As Boolean = False
    Try
        Me.ap = Me.ActionsPane
        Me.apControls = New apSteuerelemente
        Me.ap.Controls.Add(apControls)
        Me.tp = ThisApplication.TaskPanes(Word.WdTaskPanes.wdTaskPaneDocumentActions)
        AufgabenbereichDatenEinbinden(apControls)
        success = True
    Catch ex As Exception
        MessageBox.Show(ex.Message)
    End Try
    Return success
End Function

Friend Sub ToggleButtonState(ByVal btn As Office.CommandBarButton)
    Select Case btn.Tag
        Case "AufgabenbereichEinAus"
            If Me.tp.Visible Then
                Me.cbBtnAufgabenbereich.State = Office.MsoButtonState.msoButtonDown
            Else
                Me.cbBtnAufgabenbereich.State = Office.MsoButtonState.msoButtonUp
            End If
        End Select
    End Sub

```

## Datenquelle in den Aufgabenbereich einbinden

Als Datenquelle für die Beispiellösung dient eine XML-Datei, die die Kundenliste aus der *Nordwind.mdb*-Datenbank enthält. Diese wird mit dem VSTO-Dokument in den gleichen Ordner kopiert. Für diese Daten wurde ein »Typed Dataset« (die Klasse *KundenListe*) erstellt, das während *ThisDocument.Startup* instanziiert wird:

```

kundenDaten = New KundenListe
kundenDaten.ReadXml(Me.AttachedTemplate.Path & "\KundenDaten.xml")

```

Die Prozedur *AufgabenbereichDatenEinbinden* in Listing 10.17 greift auf das Datenset *kundenDaten* zu und verbindet, wie in ADO.NET üblich, seine Datenfelder mit den Steuerelementen des Benutzersteuerelements (übergeben an die Prozedur durch das Argument *apInnerControl* ) im Aufgabenbereich.

**Listing 10.17** Die Steuerelemente im Aufgabenbereich mit der Datenquelle verbinden

```
'Variable auf Klassenebene
Dim kundenDaten As KundenListe

Private Sub AufgabenbereichDatenEinbinden(ByVal apInnerControl As Windows.Forms.Control)
    Dim tkunden As DataTable = kundenDaten.Kunde
    Dim dvKunden As New DataView(tkunden)
    Dim cboKundenListe As ComboBox
    Try
        dvKunden.Sort = "Kontaktperson"
        'Combobox mit Daten füllen
        cboKundenListe = CType(apInnerControl.Controls("KontaktpersonComboBox"), ComboBox)
        cboKundenListe.DataSource = dvKunden
        cboKundenListe.DisplayMember = "Kontaktperson"

        'Die Textboxen mit der Datenquelle verknüpfen
        apInnerControl.Controls("KontaktpersonTextbox").DataBindings.Add("Text", dvKunden, _
            "Kontaktperson")
        apInnerControl.Controls("FirmaTextbox").DataBindings.Add("Text", dvKunden, "Firma")
        apInnerControl.Controls("StraßeTextbox").DataBindings.Add("Text", dvKunden, "Straße")
        apInnerControl.Controls("PLZTextbox").DataBindings.Add("Text", dvKunden, "PLZ")
        apInnerControl.Controls("OrtTextbox").DataBindings.Add("Text", dvKunden, "Ort")
        apInnerControl.Controls("RegionTextbox").DataBindings.Add("Text", dvKunden, "Region")
        apInnerControl.Controls("LandTextbox").DataBindings.Add("Text", dvKunden, "Land")
    Catch ex As Exception
        MessageBox.Show(ex.Message)
    End Try
End Sub
```

## Im Aufgabenbereich angezeigte Daten ins Dokument schreiben

Der Aufgabenbereich enthält zwei Schaltflächen. Die eine schließt den Aufgabenbereich und passt den Status der Symbolschaltfläche entsprechend an. Die andere schreibt den Inhalt der Textboxen in die Textmarken des Dokuments. Listing 10.18 veranschaulicht den Vorgang. Falls die Textmarke im Dokument noch vorhanden ist, bekommt sie den Text der entsprechenden Textbox, ansonsten wird eine Meldung eingeblendet.

### PROFITIPP

Da der Code für die Steuerelemente sich in einer anderen Klasse als *ThisDocument.vb* befindet, hat er auf das VSTO-Objekt keinen direkten Zugriff. Die VSTO-Lösung stellt ein übergeordnetes Objekt – *Globals* – bereit, das den Zugang ermöglicht.

**Listing 10.18** Im Aufgabenbereich enthaltene Daten in die Dokument-Textmarken schreiben

```
Private Sub btnDatenUebernehmen Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles btnDatenUebernehmen.Click

    With Globals.ThisDocument
        If Not FillBookmark(.bkmEmpfänger, Me.KontaktpersonTextBox) Then _
            MissingBookmark(.bkmEmpfänger.Name)
        If Not FillBookmark(.bkmAdresse, Me.StraßeTextBox) Then _
            MissingBookmark(.bkmAdresse.Name)
        If Not FillBookmark(.bkmPLZ, Me.PLZTextBox) Then _
            MissingBookmark(.bkmPLZ.Name)
    End With
End Sub
```

**Listing 10.18** Im Aufgabenbereich enthaltene Daten in die Dokument-Textmarken schreiben (Fortsetzung)

```

If Not FillBookmark(.bkmOrt, Me.OrtTextBox) Then _
    MissingBookmark(.bkmOrt.Name)
If Not FillBookmark(.bkmRegion, Me.RegionTextBox) Then _
    MissingBookmark(.bkmRegion.Name)
If Not FillBookmark(.bkmLand, Me.LandTextBox) Then _
    MissingBookmark(.bkmLand.Name)
End With
End Sub

Friend Function FillBookmark(ByVal bkm As Tools.Word.Bookmark, _
    ByVal txt As Windows.Forms.TextBox) As Boolean

    Dim doc As Tools.Word.Document = Globals.ThisDocument
    Dim success As Boolean = False
    Try
        If doc.Bookmarks.Exists(bkm.InnerObject.Name) Then
            bkm.Text = txt.Text
            success = True
        End If
    Catch ex As Exception
        MessageBox.Show(ex.Message)
    End Try
    Return success
End Function

Sub MissingBookmark(ByVal bkmName As String)
    MessageBox.Show("Die Textmarke " & bkmName & _
        " ist im Dokument nicht vorhanden. Die Daten konnten nicht eingefügt werden.")
End Sub

```

## Der Datenaustausch bei geschlossenem Dokument

Wie eingangs erwähnt, bietet VSTO die Möglichkeit, Daten in ein geschlossenes Dokument zu schreiben beziehungsweise aus einem geschlossenen zu lesen. Das im Dokument eingebettete VSTO-ActiveX-Steuerelement stellt diese Funktionalität über einen Daten-Cache (Zwischenspeicher) zur Verfügung.

Das Listing 10.19 veranschaulicht, wie der Daten-Cache eines VSTO-Dokuments initialisiert wird. In der Beispiellösung werden die Werte mehrerer Variablen des Datentyps String zwischengespeichert. (Die Funktionalität unterstützt zusätzliche Datentypen wie Datasets und DataTables; Voraussetzung ist, dass der Datentyp mit System.Xml.Serialization in XML umgewandelt werden kann. Zudem müssen die Variablen (oder Properties) als «Public» deklariert werden.)

Am Ende der Prozedur *ThisDocument\_Startup* (Listing 10.15) steht die Kommandozeile *InitializeDataCache()*. Diese Methode stellt sicher, dass die Variablen für den Zwischenspeicher initialisiert sind und Daten enthalten. (Bleibt ein Teil eines Zwischenspeichers leer, wird er als beschädigt betrachtet und die Daten können nicht gelesen werden. Es ist deshalb wichtig, sicherzustellen, dass alle Elemente des Zwischenspeichers Werte enthalten.)

Beim Speichern des VSTO-Dokuments wird das Ereignis `ThisDocument_BeforeSave` ausgeführt. Diese Prozedur sorgt dafür, dass der Inhalt der Textmarken (sofern sie im Dokument noch vorhanden sind) in die Variablen des Zwischenspeichers geschrieben werden. Fehlt eine Textmarke, wird stattdessen eine entsprechende Nachricht hinterlegt.

**Listing 10.19** Den Zwischenspeicher initialisieren und Daten hineinschreiben

```
'''Variablen für das Daten-Cache
<Cached()> Public CachedEmpfänger As String
<Cached()> Public CachedAdresse As String
<Cached()> Public CachedPLZ As String
<Cached()> Public CachedOrt As String
<Cached()> Public CachedLand As String
<Cached()> Public CachedBetreff As String

Private Sub InitializeDataCache()
    If (Not Me.IsCached("CachedEmpfänger")) Or (CachedEmpfänger Is Nothing) Then
        CachedEmpfänger = "Empfänger nicht bekannt"
    End If
    If (Not Me.IsCached("CachedAdresse")) Or (CachedAdresse Is Nothing) Then
        CachedAdresse = "Adresse nicht bekannt"
    End If
    If (Not Me.IsCached("CachedPLZ")) Or (CachedPLZ Is Nothing) Then
        CachedPLZ = "PLZ nicht bekannt"
    End If
    If (Not Me.IsCached("CachedOrt")) Or (CachedOrt Is Nothing) Then
        CachedOrt = "Ort nicht bekannt"
    End If
    If (Not Me.IsCached("CachedLand")) Or (CachedLand Is Nothing) Then
        CachedLand = "Land nicht bekannt"
    End If
    If (Not Me.IsCached("CachedBetreff")) Or (CachedBetreff Is Nothing) Then
        CachedBetreff = "Betreff nicht bekannt"
    End If
End Sub

Private Sub ThisDocument_BeforeSave(ByVal sender As Object, _
    ByVal e As Microsoft.Office.Tools.Word.SaveEventArgs) Handles Me.BeforeSave
    'Daten in die Daten-Cache schreiben
    CacheTextmarkenDaten(CachedEmpfänger, Me.bkmEmpfänger)
    CacheTextmarkenDaten(CachedAdresse, Me.bkmAdresse)
    CacheTextmarkenDaten(CachedPLZ, Me.bkmPLZ)
    CacheTextmarkenDaten(CachedOrt, Me.bkmOrt)
    CacheTextmarkenDaten(CachedLand, Me.bkmLand)
    CacheTextmarkenDaten(CachedBetreff, Me.bkmBetreff)
End Sub

Private Function CacheTextmarkenDaten(ByRef CacheVariable As Object, _
    ByVal bkm As Tools.Word.Bookmark) As String

    Dim success As Boolean = False
    If Not bkm Is Nothing Then
        Try
            CacheVariable = bkm.Text
            success = True
        Catch ex As Exception
```

Listing 10.19 Den Zwischenspeicher initialisieren und Daten hineinschreiben (Fortsetzung)

```

        MessageBox.Show(ex.Message)
        CacheVariable = "Fehler beim Schreiben."
    End Try
Else
    CacheVariable = "Textmarke nicht vorhanden."
End If
Return success
End Function

```

Von außerhalb des Dokuments greift die »ServerDocument«-Funktionalität des Namespaces `Microsoft.VisualStudio.Tools.Applications.Runtime` auf den Dokumentzwischenspeicher zu. Dieser Namespace ist nicht Teil der `Microsoft.Office.Tools-DLL`, die der Entwickler in einer VSTO-Dokumentlösung benutzt. Er befindet sich vielmehr in der Runtime-DLL, die auf dem Benutzerrechner installiert werden muss, um VSTO-Lösungen zu unterstützen. Diese DLL kann auch auf einem Netzlaufwerk oder einem Web-Server installiert werden.

Um die Funktionalität zu veranschaulichen, wurde eine kleine WinForms-Anwendung (siehe Abbildung 10.13) erstellt. Den zugehörigen Code finden Sie in Listing 10.20. Der Dateiname eines VSTO-Dokuments in einem im Code festgelegten Ordner wird in das erste Textfeld eingegeben. Die Betätigung der Schaltfläche *Daten-Cache lesen* schreibt die zwischengespeicherten Informationen in das zweite Textfeld.

Abbildg. 10.13 Die Daten aus dem Zwischenspeicher eines geschlossenen VSTO-Dokuments anzeigen



Falls das Dokument im vorgegebenen Pfad (*Eigene Dokumente*) vorhanden ist, wird ein neues `ServerDocument`-Objekt (`serverDok`) angelegt. Sein Zwischenspeicher wird angesprochen (`serverDok.CachedData`), dann die Auflistung aller sich darin befindenden »HostItems«. (Word hat nur ein »HostItem« – das Dokument. Excel hat für die Arbeitsmappe sowie jedes Arbeitsblatt eins.) Jedes »HostItem« in der Auflistung kann mehrere »DataItems« (Datenpunkte oder -elemente) enthalten. Es wird durch alle Datenpunkte von jedem »HostItem« geschleift.

Die vom `ServerDocument` gelieferten Informationen bestehen aus XML. Die Funktion `ExtractTextAusXML` liest den Inhalt des innersten XML-Elements eines Datenpunkts, worin sich der Text aus den Textmarken befindet. Das Ergebnis wird der Zeichenkette `cachedInhalt` hinzugefügt, die am Ende der Prozedur `btnDatenLesen_Click` in das zweite Textfeld geschrieben wird.

**Listing 10.20** Daten aus dem Zwischenspeicher eines geschlossenen VSTO-Dokuments lesen

```
Imports Tools = Microsoft.VisualStudio.Tools.Applications.Runtime

Public Class Form1

Private Sub btnDatenLesen_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles btnDatenLesen.Click

    Dim dokPfad As String = System.Environment.GetFolderPath( _
        Environment.SpecialFolder.MyDocuments) & "\"
    Dim serverDok As Tools.ServerDocument
    Try
        Dim dokName As String = Me.txtDokumentName.Text
        If dokName.Length > 0 Then
            Dim dokPfadMitName = dokPfad & dokName
            If System.IO.File.Exists(dokPfadMitName) Then
                serverDok = New Tools.ServerDocument(dokPfadMitName)
                Dim datenAusDemCache As Tools.CachedData = serverDok.CachedData
                Dim datenHostItems As Tools.CachedDataHostItemCollection = _
                    datenAusDemCache.HostItems
                Dim datenHostItem As Tools.CachedDataHostItem
                Dim datenPunkt As Tools.CachedDataItem
                Dim cachedInhalt As String = ""
                For Each datenHostItem In datenHostItems
                    For Each datenPunkt In datenHostItem.CachedData
                        cachedInhalt = cachedInhalt & datenPunkt.Id & ": " & _
                            ExtractTextAusXML(datenPunkt.Xml) & vbCrLf
                    Next
                Next
                'Zeigt den Dateninhalt im Fenster an
                Me.txtDaten.Text = cachedInhalt
            Else
                MessageBox.Show("Das Dokument '" & dokName & "' konnte im Pfad '" & dokPfad _
                    & "' nicht gefunden werden.")
            End If
        Else
            MessageBox.Show("Bitte oben einen Dokumentnamen eingeben.")
        End If
    Catch ex As Exception
        MessageBox.Show(ex.Message)
    Finally
        If Not serverDok Is Nothing Then serverDok.Close()
    End Try
End Sub

Private Function ExtractTextAusXML(ByVal inhalt As String) As String
    Dim inhaltText As String
    Dim xmlDoc As Xml.XmlDocument = New Xml.XmlDocument

    xmlDoc.LoadXml(inhalt)
    inhaltText = xmlDoc.LastChild.InnerText
    xmlDoc = Nothing
    Return inhaltText
End Function
End Class
```

## VSTO-Lösung von einem Dokument abtrennen

Wie in der Einleitung zu diesem Abschnitt »Das VSTO-Dokument vorbereiten« erwähnt, ist es möglich, den VSTO-Code vom Dokument zu trennen. Da sich dieser Code im Gegensatz zu VBA nicht im Dokument befindet, geht das relativ problemlos.

Das VSTO-Team hat zwei Methoden bereitgestellt:

- Über das `ServerDocument`-Objekt. Diese Methode ermöglicht das Entfernen sowie Anfügen einer VSTO-Lösung aus einem bzw. an ein Word- oder Excel-Dokument. Dabei wird jedoch die Word- oder Excel-Anwendung gestartet, weshalb es – trotz des Namens – für den Gebrauch auf einem Server nicht geeignet ist. Diese Methode ist ausreichend im VSTO-Teil der MSDN-Seite beschrieben und wird hier nicht näher behandelt.
- Durch die Methode `RemoveCustomization`. Diese wird auf das in Word (oder Excel) geöffnete VSTO-Dokument ausgeführt, wie in Listing 10.21 ersichtlich.

`RemoveCustomization` entfernt das VSTO-ActiveX-Steuerelement aus dem Dokument. Wird es in diesem Zustand geschlossen, bleibt der `DocumentActions`-Aufgabenbereich in der Liste der Aufgabenbereiche aufgeführt. Zudem, falls der Aufgabenbereich jemals eingeblendet wurde, enthält das Dokument weiterhin die Verknüpfung zum XML-Schema `ActionsPane`. Mit der `Clear`-Methode wird der Aufgabenbereich aus der Liste entfernt; anschließend wird die Verknüpfung zum Schema (falls vorhanden) gelöscht.

Nach Ausführung dieser Handlungen besteht durch Ereignisse (sowohl der Anwendung als auch der Symbolschaltflächen) immer noch eine Verbindung zur VSTO-Lösung. Durch Sperren aller von der Lösung benutzten Objekte der `CommandBar`-Auflistung können diese Befehle dem Benutzer entzogen werden. Unter Umständen sollen Ereignis-Prozeduren testen, ob `RemoveCustomization` durchgeführt wurde, und wenn ja, keine weitere Handlungen durchführen.

Jedes VSTO-Dokument enthält mindestens zwei benutzerdefinierte Dokumenteigenschaften, die es mit der VSTO-Lösung verbinden: `_AssemblyName` sowie `_AssemblyLocation`. Der Wert der ersten ist lediglich ein Sternchen (in VSTO 2003 enthielt es den Namen der Lösung), das dem »Microsoft Office System loader« mitteilt, dass das Dokument mit einer VSTO-Lösung verbunden ist. Der zweite enthält das GUID für das »Runtime Storage Control« – das ActiveX-Steuerelement, das im Dokument als ein Shape-Objekt eingebettet ist. Fehlen diese Einträge, wurde die Lösung erfolgreich vom Dokument getrennt.

### HINWEIS

Der »Microsoft Office System loader« (auch als »Visual Studio Tools for Office loader« bekannt) ist Bestandteil der Office 2003- und Office 2007-Anwendungen. Er dient als Schnittstelle zwischen dem .NET-Code einer VSTO-Lösung und der COM-Umgebung von Office. Durch ihn wird die VSTO-Lösung geladen und werden die Ereignisse der Anwendung überwacht und an die Lösung weitergeleitet. Mehr Informationen zum Loader finden Sie auf den Microsoft-Webseiten [http://msdn2.microsoft.com/en-us/library/zcfbd2sk\(VS.80\).aspx](http://msdn2.microsoft.com/en-us/library/zcfbd2sk(VS.80).aspx) sowie [http://msdn2.microsoft.com/en-us/library/aa194021\(office.11\).aspx](http://msdn2.microsoft.com/en-us/library/aa194021(office.11).aspx).

Das Dokument einer VSTO-Vorlagenlösung bleibt weiterhin mit seiner Vorlage verbunden. Falls es mit einer anderen Vorlage, wie *Normal.dot*, verbunden werden soll, geschieht dies am besten beim Schließen des Dokuments. Dafür wird das Ereignis `ThisApplication.DocumentBeforeClose` benutzt.

**Listing 10.21** Die VSTO-Lösung von einem Dokument trennen

```

Private Sub InNormalesDokWandeln(ByVal ctrl As Office.CommandBarButton, _
    ByRef Cancel As Boolean)
    Try
        'Entfernt das VSTO ActiveX-Steuerelement aus dem Dokument.
        Me.RemoveCustomization()
        'Den Aufgabenbereich aus der Liste entfernen,
        ActionsPane.Clear()
        'sowie das Schema-Referenz aus dem Dokument,
        'sofern der Aufgabenbereich eingeblendet wurde.
        Dim xmlSchema As Word.XMLSchemaReference
        For Each xmlSchema In Me.XMLSchemaReferences
            If xmlSchema.NamespaceURI = "ActionsPane" Then
                xmlSchema.Delete()
            Exit For
        End If
    Next
    Catch ex As Exception
        MessageBox.Show(ex.Message)
    Finally
        cb.Enabled = False
    End Try
End Sub

Private Sub ThisApplication_DocumentBeforeClose( _
    ByVal Doc As Microsoft.Office.Interop.Word.Document, _
    ByRef Cancel As Boolean) Handles ThisApplication.DocumentBeforeClose
    'Stellt sicher, dass das Dokument die VSTO-Vorlage nicht mehr anpeilt.
    'Gewisse Ereignisse werden jedoch noch ausführbar sein, bis das
    'Dokument geschlossen wurde.

    Dim bVstoDocProp As Boolean = False
    Dim prop As Office.DocumentProperty
    For Each prop In Me.InnerObject.CustomDocumentProperties
        If prop.Name = "_AssemblyName" Or prop.Name = "_AssemblyLocation" Then
            bVstoDocProp = True
        Exit For
    End If
    Next
    If Not bVstoDocProp Then Me.InnerObject.AttachedTemplate = _
        ThisApplication.NormalTemplate
End Sub

```

## VSTO-Dokumentlösungen in Word 2007

Meistens können mit VSTO 2005 erstellte Dokumentlösungen auch in Word 2007 benutzt werden. Dabei ist zu bedenken, dass alle in der Lösung definierten Symbolleisten und -schaltflächen unter der Registerkarte *Add-Ins* erscheinen werden. Es ist nicht möglich, das Ribbon in die Lösung einzubinden; dies wird erst mit der nächsten VSTO-Version möglich sein.

Das Setup-Programm muss die entsprechende VSTO-Runtime verteilen.



**HINWEIS**

Informationen sowie einen Link für das Herunterladen der Runtime finden Sie auf der Microsoft-Website unter [http://msdn2.microsoft.com/en-us/library/ms178739\(vs.80\).aspx](http://msdn2.microsoft.com/en-us/library/ms178739(vs.80).aspx).

Der Entwickler wird eine Dokumentlösung nicht auf einem Rechner mit installiertem Office 2007-Rechner bearbeiten können. VSTO 2005 lässt sich nur auf einem Rechner mit Office 2003 Professional installieren. Parallelinstallationen mehrerer Office-Versionen sind in diesem Zusammenhang nicht zulässig.

## Verteilung von VSTO-Lösungen

Dieses Thema ist sehr komplex und wird hier nicht im Detail behandelt. Dazu stehen seitenlange Beschreibungen im Internet bereit. Wir geben nur einen kurzen Überblick.

**HINWEIS**

Deutschsprachige Informationen zum Thema Verteilung (mit Links zu weiteren Artikeln) finden Sie in Jens Häupels Blog: [http://blogs.msdn.com/jensha/archive/tags/Setup+\\_2600\\_amp\\_3B00\\_+Deployment/default.aspx](http://blogs.msdn.com/jensha/archive/tags/Setup+_2600_amp_3B00_+Deployment/default.aspx).

Da VSTO Teil von .NET Framework ist, folgt es dessen Sicherheitsrichtlinien. Wichtigster Punkt für VSTO-Entwickler ist: *Allen Teilen* einer VSTO-Lösung muss der Sicherheitsstatus *volles Vertrauen* («Full Trust») zugeordnet werden. Das schließt sowohl die *DLL* als auch das Dokument sowie allfällige «Helfer *dll*» ein. Die Zuteilung der Sicherheitsstufe erfolgt entweder manuell über den *Microsoft .Net Framework Konfiguration*-Assistenten oder im Setup programmtechnisch über «CASPOL» (Code Access Security POLicy).

Die .NET 2.0-Sicherheit kennt drei Arten von Beweis, welche Ausführungsrechte .NET-Code zugeteilt wurden: Digitale Signatur, Speicherort (URI) sowie «Strong Name». Die Sicherheit kann auf einer der drei Arten oder einer beliebigen Kombination basieren. Für den Eigenbedarf ist der Speicherort einfach einzurichten, entspricht jedoch der schwächsten Sicherheitsstufe.

**HINWEIS**

VSTO 2005 unterstützt keine «ClickOnce»-Verteilung. Diese Funktionalität ist jedoch für die nächste Version vorgesehen.

## Das Beispiel lauffähig machen

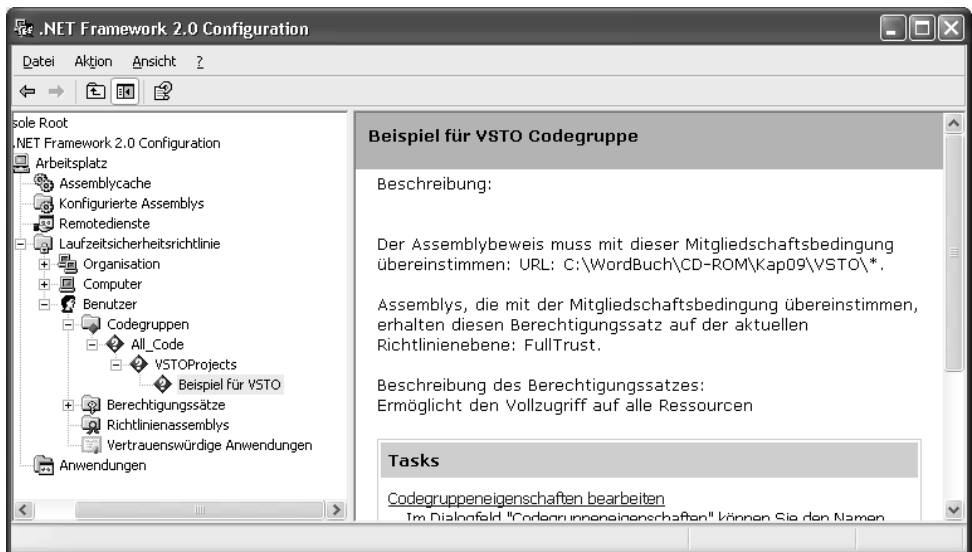
Die folgenden einfachen Schritte sind nur für die Anwendung des Beispiels und als Einführung in die Verteilung gedacht. Visual Studio muss nicht installiert sein, wohl aber das .NET-Framework 2.0 oder 3.0 sowie VSTO Runtime (siehe den Abschnitt »VSTO-Dokumentlösungen in Word 2007«). Sind beide installiert, gehen Sie wie folgt vor:

1. Die Assembly *Bsp\_VSTO\_VB.dll*, die Lösungsvorlage *Bsp\_VSTO\_VB.dot* und die XML-Datei *KundenDaten* müssen sich im selben Ordner befinden. Möchten Sie auch das Daten-Lesen des Zwischenspeichers testen, kopieren Sie die *DatenLesen\_VB.exe* ebenfalls in diesen Ordner.
2. In der Systemsteuerung, unter der Rubrik *Verwaltung*, starten Sie den *Microsoft .NET Framework 2.0-Konfiguration*-Assistenten.
3. Unter *Arbeitsplatz/Laufzeitsicherheitsrichtlinie/Benutzer/Codegruppen* (Abbildung 10.14) wählen Sie den Eintrag *All\_Code*. Falls noch kein Eintrag «VSTOProjects» vorhanden ist, erstellen Sie ihn wie folgt:

- Wählen Sie den Eintrag *All\_Code*.
- Klicken Sie im rechten Fenster *Untergeordnete Codegruppe hinzufügen* an, um den Assistenten zu starten.
- Tragen Sie im ersten Schritt den Namen *VSTOPProjects* ein und klicken Sie auf *Weiter*.
- Im folgenden Schritt akzeptieren Sie den Vorschlag *Gesamter Code* und klicken auf *Weiter*.
- Im dritten Schritt legen Sie *Nothing* als *Berechtigungssatz* fest und klicken auf *Weiter*.
- Im letzten Schritt wählen Sie *Fertig st.*
- Falls dem Assistenten der Eintrag *Kopie von* vorangestellt ist, klicken Sie ihn mit rechts an, wählen im Kontextmenü *Umbenennen* und korrigieren den Namen.

Abbildg. 10.14

Benutzerspezifische Berechtigungen für ein Assembly erstellen



4. Wählen Sie den (neu erstellten) Eintrag *VSTOPProjects* und erstellen Sie einen Eintrag für den Ordner, in dem sich das Assembly befindet:
  - Klicken Sie im rechten Fenster auf *Untergeordnete Codegruppe hinzufügen*, um den Assistenten zu starten.
  - Tragen Sie einen aussagekräftigen Namen ein und klicken Sie auf *Weiter*.
  - Im folgenden Schritt wählen Sie *URL* aus der Liste und geben in das eingblendete Textfeld *URL* die vollständige Pfadangabe zur *.dll* ein, evtl. gefolgt von *\\**. Klicken Sie auf *Weiter*.
  - Im letzten Schritt klicken Sie auf *Fertig st.*
  - Falls dem Assistenten der Eintrag *Kopie von* vorangestellt ist, korrigieren Sie den Namen über den Befehl *Umbenennen* seines Kontextmenüs.



Die vorgestellte Lösung befindet sich in der Datei *Bsp\_VSTO.zip* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap10*. Um damit zu arbeiten, öffnen Sie die Lösung *Bsp\_VSTO\_VB.sln* in Visual Studio 2005 Tools for Office. Drücken Sie **[F5]**, um sie im Debug-Modus auszuführen.

**HINWEIS** Weitere Informationen zu Office-Entwicklerthemen in deutscher Sprache finden Sie im Blog von

Jens Häupel (<http://blogs.msdn.com/jensha/archive/tags/.NET+Dev+mit+Office/default.aspx>) sowie auf den deutschen MSDN Office Developer-Seiten (<http://www.microsoft.com/germany/msdn/office/default.msp>).

Die englischsprachigen MSDN-Seiten zum Thema VSTO starten auf der Seite [http://msdn2.microsoft.com/en-us/library/23cw517s\(VS.80\).aspx](http://msdn2.microsoft.com/en-us/library/23cw517s(VS.80).aspx).

Außerdem gibt es zwei nützliche Bücher in englischer Sprache:

»VSTO for Mere Mortals« von Kathleen McGrath und Paul Stubbs, herausgegeben von Addison-Wesley.

»Visual Studio Tools for Office« von Eric Carter und Eric Lippert, herausgegeben von Addison-Wesley. Je eine Version für VB.NET sowie C# ist erhältlich.

## VSTO COM-Add-Ins

Ende 2006, zeitgleich mit der Marktfreigabe von Office 2007, stellte das VSTO-Team die Zwischenversion VSTO 2005 SE (»Second Edition«) kostenlos zur Verfügung. Sie steht zum Herunterladen auf der Microsoft-Internetseite bereit (siehe den Hinweis in der Einleitung zum Abschnitt »VSTO 2005-Dokumentlösungen«). Damit kann der Entwickler COM-Add-Ins für mehrere Office 2003- und Office 2007-Anwendungen erstellen.

»Managed code« COM-Add-Ins sind nichts Neues. Seit einigen Versionen gehört zum Lieferumfang von Visual Studio die Vorlage *Gemeinsames Add-In* für COM-Add-Ins; sie befindet sich unter *Andere Projekttypen/Erweiterungen*. Diese Art Projekt baut auf der gleichen Schnittstelle auf wie ein COM-Add-In, das in Visual Basic 6.0 entwickelt wird. Ein solches COM-Add-In unterstützt gleichzeitig mehrere Office-Anwendungen sowie -Versionen. Um mit einer spezifischen Anwendung zu arbeiten, muss im Code getestet und ausgewertet werden. In der .NET Umgebung mit seinen starken Datentypen kann sich dies recht umständlich gestalten.

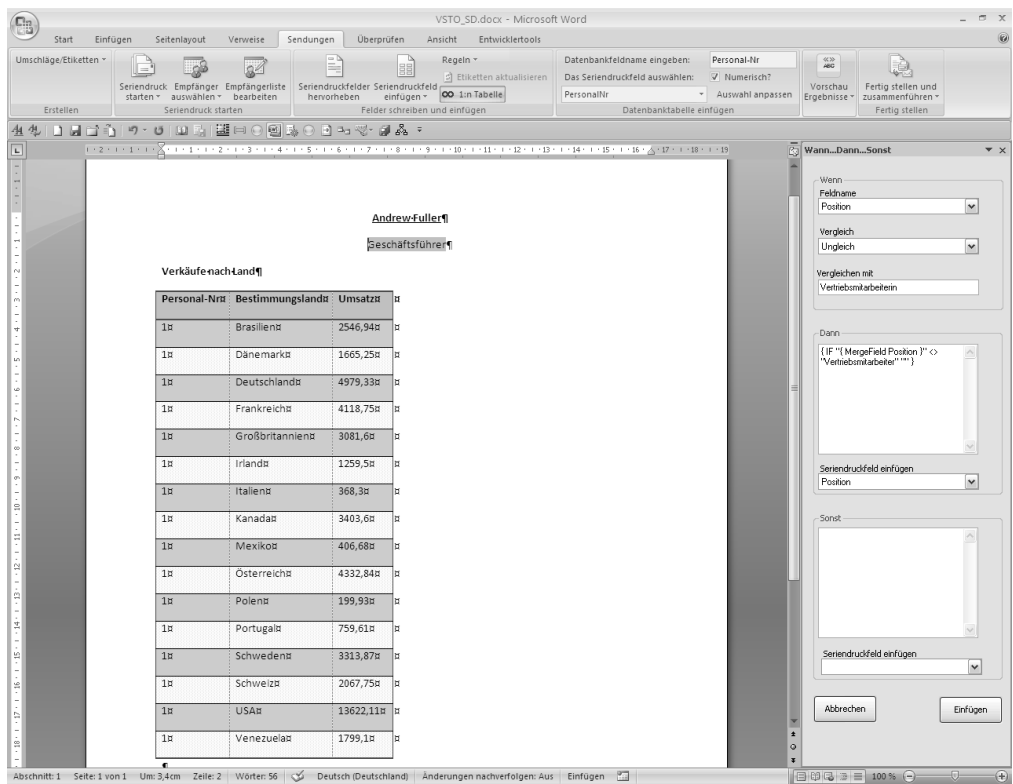
Ein VSTO-COM-Add-In hingegen ist anwendungs- (sowie versionen-)spezifisch. Obwohl es auf der gleichen Schnittstelle wie die allgemeine Vorlage aufbaut, erledigt das VSTO-Werkzeug im Hintergrund viele Details, um die sich der Entwickler sonst kümmern müsste. Statt beispielsweise der fünf Prozeduren eines gemeinsamen Add-Ins, die für das Laden und Entfernen sorgen, hat ein VSTO Add-In derer nur zwei: *ThisAddin\_Startup* sowie *ThisAddin\_Shutdown*.

Die zwei Arten unterscheiden sich auch im Ladeverhalten. Alle gemeinsamen Add-Ins werden in die gleiche »AppDomain« geladen. Hat eines davon ein Problem und muss folglich von der Anwendung gesperrt werden, werden alle Add-Ins in derselben Domain gesperrt. Abhilfe schafft nur ein so genanntes »Shim«, ein mit C++ erstellter Teil, der dafür sorgt, dass das Add-In in eine eigene Domain geladen wird.

VSTO-Add-Ins werden vom »Microsoft Office System loader« (siehe auch den Abschnitt »VSTO-Lösung von einem Dokument abtrennen«) verwaltet und in getrennten Domains geladen. Somit kann kein Add-In ein anderes zum Absturz bringen.

Wir stellen hier ein kleines Beispiel für Word 2007 vor, das veranschaulicht, wie die COM-Add-In-Vorlage aus VSTO den Umgang mit der Ribbon- und Custom Task Pane-Funktionalität (Multifunktionsleiste bzw. frei definierbarer Aufgabenbereich) erleichtert. Ziel dieses Add-Ins ist es, die Seriendruckfunktionalität anzupassen und zu erweitern, wie in Abbildung 10.15 ersichtlich. Dabei werden nicht benötigte Steuerelemente aus der Multifunktionsleiste entfernt, die Wirkung anderer angepasst sowie neue hinzugefügt. Die Funktionalität wird ergänzt durch benutzerfreundlichere Schnittstellen für das Zusammensetzen von *If*-Feldfunktionen sowie das Verbinden einer *Database*-Feldfunktion mit der Seriendruckdatenquelle, um 1:n-Tabellen für die einzelnen Datensätze zu erstellen.

**Abbildg. 10.15** Die Registerkarte *Sendungen* wird angepasst und die Seriendruck-Funktionalität erweitert



#### HINWEIS

Die Verteilung von VSTO 2005 SE-Add-Ins ist komplex und wird hier nicht behandelt. Einen ausführlichen Artikel zum Thema finden Sie unter <http://go.microsoft.com/fwlink/?LinkId=57779>.

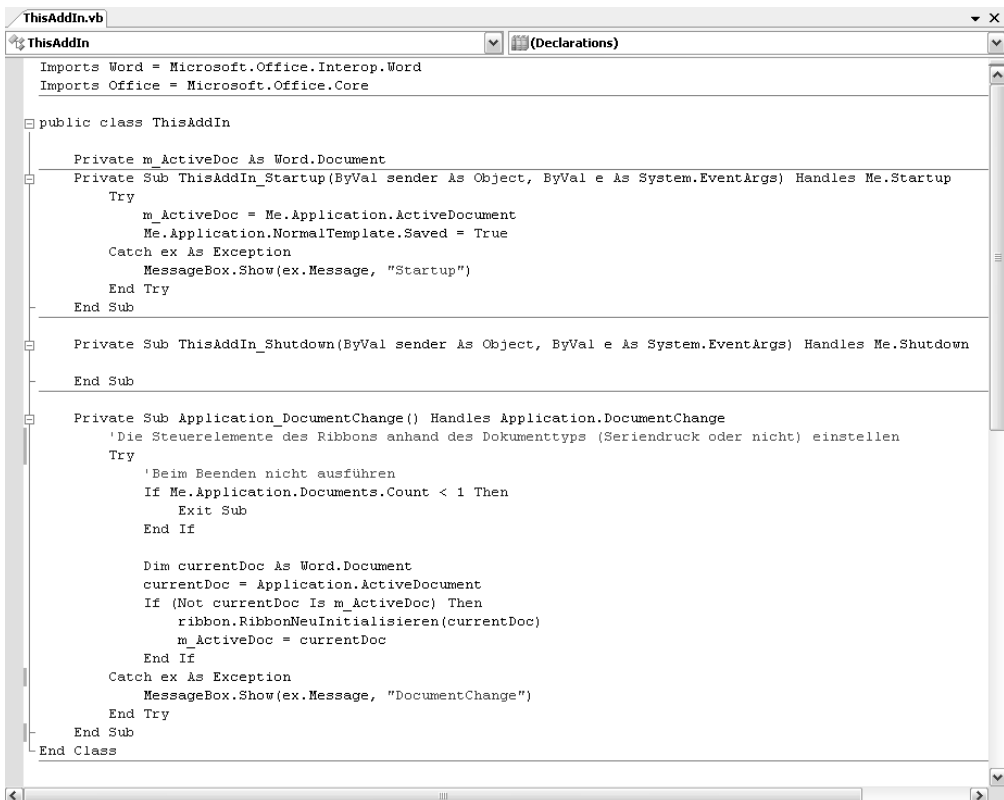
## Das Anlegen eines COM-Add-In Projekts

Um ein VSTO COM-Add-In zu erstellen, wird im Visual Studio-Dialogfeld *Neues Projekt* der Ordner *Visual Basic* (oder *C#*) gewählt. Aus dem Unterordner *Office/2007-Add-Ins* (siehe auch Abbildung 10.4) wird der Eintrag *Word-Add-In* markiert und im unteren Teil des Dialogfelds ein eindeutiger Name für das Projekt eingegeben und der Speicherort festlegt.

Visual Studio zeigt nach einigen Sekunden im Code-Fenster die Klasse *ThisAddIn* an mit den Prozeduren *ThisAddin\_Startup* sowie *ThisAddin\_Shutdown*. Wie üblich für VSTO, wurden die benötigten Verweise zu den Office- sowie Word-PIAs erstellt. Benötigte Imports-Anweisungen werden VB-Entwickler selbst eingeben müssen.

Die *ThisAddIn*-Klasse des Beispielprojekts ist in Abbildung 10.16 ersichtlich. Von allgemeinem Interesse ist die Zeile `Me.Application.NormalTemplate.Saved = True`. Beim Laden eines COM-Add-Ins werden unvermeidliche Änderungen in der Vorlage *Normal.dot* vorgenommen. Diese Anweisung unterdrückt die störende Aufforderung, diese Änderungen zu speichern.

Abbildg. 10.16 Die *ThisAddIn*-Klasse eines COM-Add-In-Projekts



## Word 2007 COM-Add-In-Benutzerschnittstellen

Für Word 2007 wurde die Benutzerschnittstelle grundlegend überarbeitet. Symbolleisten gibt es keine mehr. Stattdessen werden Befehle in der Multifunktionsleiste – auch Ribbon genannt – angeboten. Das Ribbon wird nicht mehr über das Objektmodell `CommandBars` verwaltet, sondern in einer XML-Datei mit Callbacks definiert. Eine ausführliche Beschreibung hierzu finden Sie in Kapitel 17.

Neben dem Ribbon steht dem Entwickler zudem ein frei definierbarer Aufgabenbereich zur Verfügung: der »Custom Task Pane«, auch als CTP bezeichnet. Im Gegensatz zum Aufgabenbereich *Dokumentaktionen* einer VSTO-Dokumentlösung ist der CTP nicht an ein bestimmtes Dokument gebunden, sondern gilt, wie das COM-Add-In, für die gesamte Anwendung.

### WICHTIG

Da Word seit der Version 2002 jedes Dokument in einem unabhängig laufenden Anwendungsfenster anzeigt, gestaltet sich je nach Funktionsvorstellung die Verwaltung der CTPs etwas problematisch. Eingblendete CTPs werden nicht automatisch für jedes Dokumentfenster propagiert. Die Verwaltung der CTPs muss für jedes Projekt sorgfältig ausgearbeitet werden. Eine Behandlung des Themas sprengt leider den Rahmen dieses Buches. Für mehr Informationen wenden Sie sich bitte an die MSDN-Webseiten und -Foren.

In den folgenden Abschnitten werden die Erstellung und der Umgang mit den VSTO-Werkzeugen für diese zwei Schnittstellen näher vorgestellt.

### Das COM-Add-In mit einer Ribbon-Erweiterung ergänzen

In Kapitel 17 ist beschrieben, wie eine Ribbon-Erweiterung in ein Dokument oder eine Vorlage eingebunden wird. Die resultierende Multifunktionsleiste ist begrenzt auf diese Datei (oder, im Fall einer Vorlage, auf die mit der Vorlage verbundenen Dokumente).

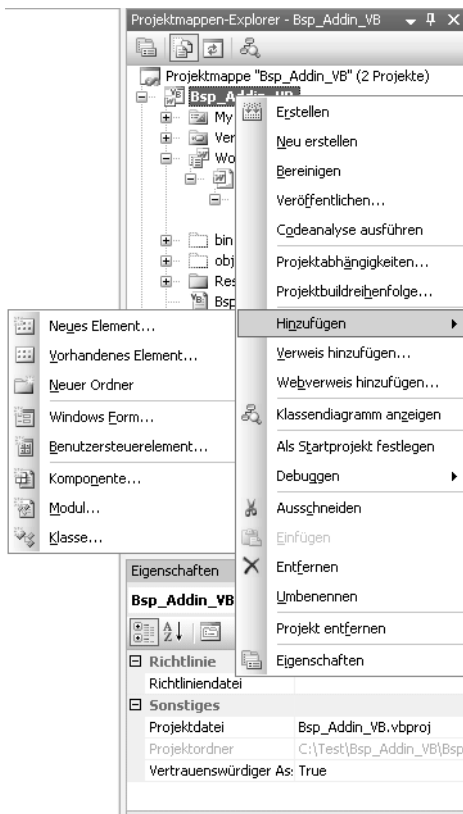
Dagegen steht die Ribbon-Erweiterung eines COM-Add-In der ganzen Anwendung zur Verfügung und wird beim Laden des COM-Add-In in die vorhandene Multifunktionsleiste integriert. Die Datei mit dem definierenden XML ist integrierter Teil des Visual Studio-Projekts.

Um dem VSTO COM-Add-In eine Ribbon-Erweiterung hinzuzufügen, gehen Sie wie folgt vor:

1. Im Fenster *Projektmappen-Explorer* klicken Sie mit rechter Maustaste auf den Projektnamen.
2. Wählen Sie im Kontextmenü den Untermenübefehl *Hinzufügen/Neues Element* (Abbildung 10.17).
3. Im Dialogfeld *Neues Element hinzufügen* wählen Sie den Eintrag *Multifunktionsleisten-Unterstützung* aus und geben unten einen aussagekräftigen Namen ein (Abbildung 10.18).

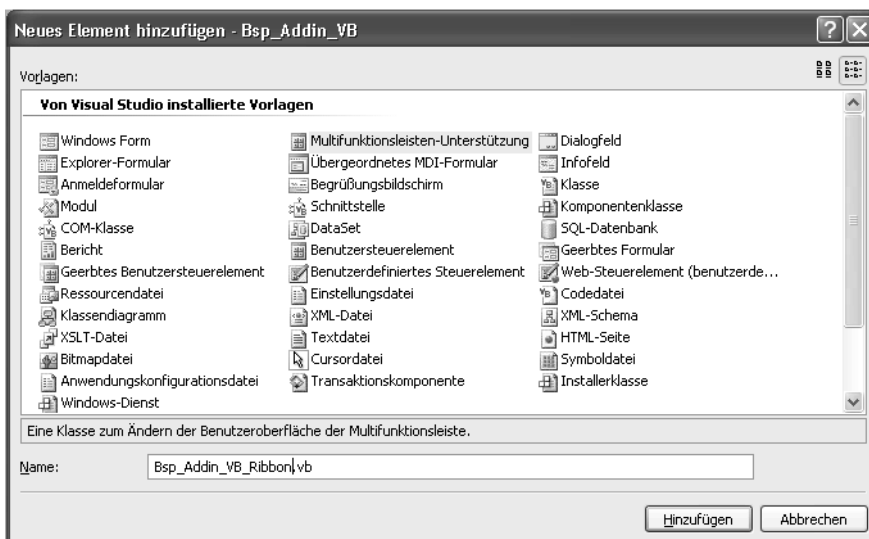
Abbildg. 10.17

Das Projekt um ein neues Element erweitern



Abbildg. 10.18

Dem Projekt eine Klasse für die Ribbon-Erweiterung hinzufügen



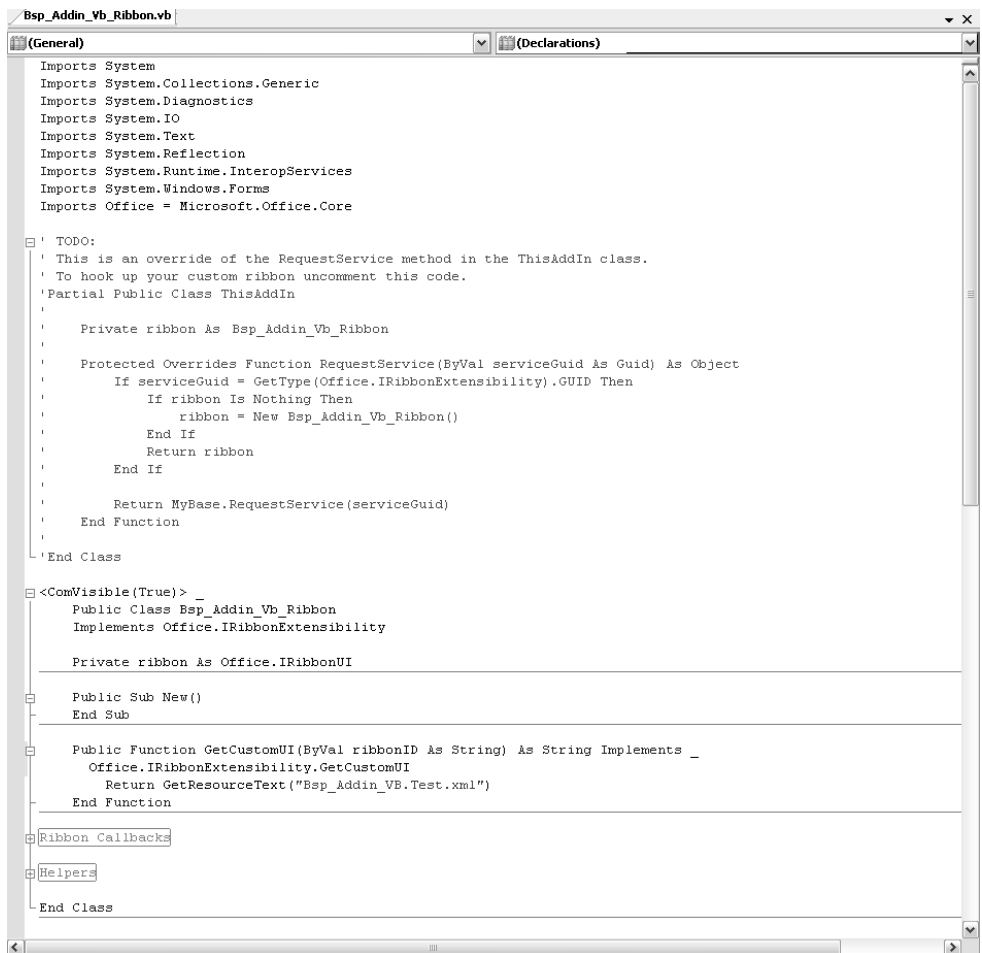
Nach Bestätigung des Dialogfeldes erscheinen im Fenster *Projektmappen-Explorer* zwei neue Einträge mit dem im Dialogfeld eingegebenen Namen: eine Klasse (je nach Programmiersprache mit der Endung *vb* oder *cs*) sowie eine XML-Datei.

Die neu erstellte Klasse ist in Abbildung 10.19 erkennbar. VSTO hat unter anderem die folgenden Handlungen für Sie erledigt:

- Verschiedene, allgemein benötigte *Imports*-Anweisungen wurden geschrieben (Sie werden selber eine für Word eingeben müssen).
- Eine auskommentierte *RequestService*-Funktion wurde als Teil der *ThisAddin*-Klasse eingefügt. Falls im Projekt noch keine Prozedur mit diesem Namen vorhanden ist (wie hier), sollten die Kommentarzeichen entfernt werden. Sonst muss die vorhandene Prozedur um die Informationen ergänzt werden, die das ribbon-Objekt betreffen.

**Abbildg. 10.19**

Die von VSTO erstellte Klasse für die Ribbon-Erweiterung





- Die Klasse für die Ribbon-Erweiterung wurde erstellt. Sie enthält u.a. eine globale Variable für das Ribbon-Objekt, einen Abschnitt *Helpers* mit unterstützenden Funktionen, einen Abschnitt *Ribbon Callbacks* mit Beispielprozeduren für Ribbon-Steuerelemente sowie die Funktion *GetCustomUI*. Diese Prozedur sorgt in einem COM-Add-In für das Laden des Ribbon-Erweiterungs-XML. Normalerweise wird es, wie hier, als Datei übergeben. Es wäre aber auch möglich, eine Zeichenkette mit dem gesamten XML-Code zurückzugeben.
- Die Grundstrukturen des Ribbon-Erweiterungs-XMLs wurden in die XML-Datei geschrieben, mit Beispielen für Steuerelemente.

Das XML für unser Beispiel-Projekt befindet sich in Listing 10.22. Einzelheiten zu den darin verwendeten Steuerelementen und Attributen finden Sie in Kapitel 17. Hier stellen wir lediglich die Lösung sowie Besonderheiten des VSTO-Werkzeugs vor.

**Listing 10.22** Das XML für die Ribbon-Erweiterung des VSTO-Beispielprojekts

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui" onLoad="OnLoad"
loadImage="ribbon_getImages">
  <ribbon>
    <tabs>
      <tab idMso="TabMailings" >
        <group idMso="GroupEnvelopeLabelCreate" visible="false" />
        <group idMso="GroupMailMergeStart" visible="false" />
        <group idMso="GroupMailMergeWriteInsertFields" visible="false" />
        <group id="AddinEnvelopeLabelCreate" label="Erstellen"
          insertBeforeMso="GroupMailMergePreviewResults" >
          <dynamicMenu id="AddinEnvelopesAndLabels" label="Umschläge/Etiketten"
            getContent="menu_GetContent" />
        </group>
        <group id="AddinMailMergeStart" label="Seriendruck starten"
          insertBeforeMso="GroupMailMergePreviewResults" >
          <menu idMso="MailMergeStartMailMergeMenu" size="large" />
          <menu id="AddinMailMergeSelectRecipients" imageMso="MailMergeSelectRecipients"
            label="Empfänger auswählen" size="large" >
            <button id="AddinMailMergeCreateList" imageMso="MailMergeCreateList"
              label="Neue Liste eingeben..." tag="MailMergeCreateList"
              onAction="AddinMailMergeStartMenus_Click"/>
            <button id="AddinMailMergeReceipientsUseExistingList"
              imageMso="MailMergeReceipientsUseExistingList"
              label="Vorhandene Liste verwenden..."
              tag="MailMergeReceipientsUseExistingList"
              onAction="AddinMailMergeStartMenus_Click"/>
            <button id="AddinMailMergeReceipientsUseOutlookContacts"
              imageMso="MailMergeReceipientsUseOutlookContacts"
              label="Aus Outlook-Kontakten auswählen..."
              tag="MailMergeReceipientsUseOutlookContacts"
              onAction="AddinMailMergeStartMenus_Click"/>
          </menu>
          <button idMso="MailMergeRecipientsEditList" size="large" />
        </group>
        <group id="AddinMailMergeWriteInsertFields"
          insertBeforeMso="GroupMailMergePreviewResults"
          label="Felder schreiben und einfügen" >
          <toggleButton idMso="MailMergeHighlightMergeFields" size="large" />
          <splitButton idMso="MailMergeMergeFieldInsertMenu" size="large" />
          <menu id="AddinMailMergeRules" itemSize="normal" label="Regeln" size="normal">
```

**Listing 10.22** Das XML für die Ribbon-Erweiterung des VSTO-Beispielprojekts (Fortsetzung)

```

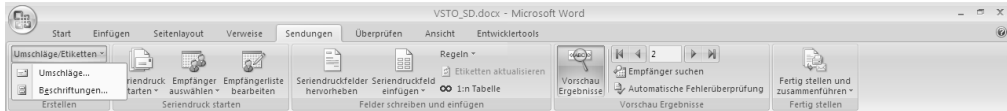
        <button id="AddinInsertIfField" onAction="OnRulesButton_Click"
            label="Wenn... Dann... Sonst..." tag="IfField"
            getEnabled="sdAktiviert_getEnabled" />
        <button id="AddinInsertAskField" onAction="OnRulesButton_Click"
            label="Frage..." tag="AskField"/>
        <button id="AddinInsertFillinField" onAction="OnRulesButton_Click"
            label="Eingeben..." tag="FillinField"/>
        <button id="AddinInsertMergeRecordField" onAction="OnRulesButton_Click"
            label="Datensatz zusammenführen" tag="MergeRecordField"/>
        <button id="AddinInsertMergeSequenceField" onAction="OnRulesButton_Click"
            label="Sequenz zusammenführen" tag="MergeSeqField"/>
        <button id="AddinInsertMergeNextField" onAction="OnRulesButton_Click"
            label="Nächster Datensatz" tag="MergeNextField"/>
        <button id="AddinInsertSetField" onAction="OnRulesButton_Click"
            label="Textmarke festlegen" tag="SetField" />
    </menu>
    <button idMso="MailMergeUpdateLabels" size="normal" />
    <toggleButton id="ShowDatabaseGroup" label="1:n Tabelle" image="database"
        getPressed="ShowDataBaseGroup_GetPressed"
        onAction="ShowDataBaseGroup_Click" />
</group>
<group id="InsertDatabase" label="Datenbanktabelle einfügen"
    insertBeforeMso="GroupMailMergePreviewResults"
    getVisible="DataBaseGroup_getVisible" tag="InsertDatabaseGroup">
    <labelControl id="labelDatabaseFieldName"
        label=" Datenbankfeldname eingeben:" />
    <labelControl id="labelMergeFieldListe"
        label=" Das Seriendruckfeld auswählen:" />
    <dropDown id="dropDownMergeFieldListe" onAction="MergeFieldName_Select"
        getItemCount="dropDownFeldListe_getItemCount"
        getItemID="dropDownFeldListe_getItemLabel"
        getItemLabel="dropDownFeldListe_getItemLabel"
        getEnabled="sdAktiviert_getEnabled"
        sizeString="Das Seriendruckfeld auswählen:" />
    <editBox id="editDatabaseFieldName"
        getEnabled="sdAktiviert_getEnabled" onChange="DatabaseFieldName_OnChange"
        screentip="Datenbankfeldname eingeben"
        supertip="Den Datenbankfeldnamen eingeben, der Datensätze mit dem
            gegenwärtigen Seriendruckdatensatz verbindet. Meistens ein ID-Feld."
        sizeString=" Auswahl anpassen" />
    <checkBox id="checkNumerisch" label="Numerisch?"
        onAction="checkNumerisch_Click" />
    <button id="buttonChangeSELECT" onAction="ChangeSELECT_Click"
        label=" Auswahl anpassen"/>
</group>
</tab>
</tabs>
</ribbon>
</customUI>

```

Aus dem Listing ist zu erkennen, dass die Word-eigenen Gruppen *Erstellen*, *Seriendruck starten* sowie *Felder schreiben und einfügen* ausgeblendet werden. Im Fall der ersten Gruppe soll Platz in der Registerkarte *Sendungen* gewonnen werden. Um den Benutzer nicht zu verwirren, werden die Befehle in einer selbst definierten Gruppe gleichen Namens zur Verfügung gestellt. Wir haben

die Form eines dynamischen Menüs gewählt, um dieses Steuerelement zu veranschaulichen. Der Anfangsstand der Ribbon-Erweiterung mit aufgeklapptem dynamischen Menü ist in Abbildung 10.20 ersichtlich.

**Abbildg. 10.20** Die durch das COM-Add-In angepasste Registerkarte *Sendungen*



Die Gruppe *Seriendruck starten* wird ersetzt, um die Funktionalität der Befehle, die Datenquellen in das Dokument verbinden, zu erweitern. Bestimmte Word-eigene Befehle stehen einzig Seriendruckdokumenten zur Verfügung. Um dies auch für selbst definierte Steuerelemente zu verwirklichen, muss die Ribbon-Erweiterung beim Einbinden einer Datenquelle neu initialisiert werden. Leider funktioniert das Ereignis `MailMergeDataSourceLoad` nicht wunschgemäß, weshalb in der Callback-Prozedur `AddinMailMergeStartMenus_Click` zuerst der Word-Befehl (der anhand des `tag`-Attributs des jeweiligen Steuerelements identifiziert wird) und anschließend die Initialisierung der Ribbon-Erweiterung ausgeführt werden (Listing 10.23).

#### HINWEIS

Die Prozedur *RibbonNeuInitialisieren* wird auch vom Ereignis `Application.DocumentChange` aufgerufen, wie in Abbildung 10.16 ersichtlich. Somit wird die Sperrung der Seriendruck-Befehle dynamisch für das aktuelle Dokument angepasst.

**Listing 10.23** Die Ribbon-Erweiterung nach Einbinden einer Datenquelle neu initialisieren

```
'Auf Klassenebene deklarierte Variable
Private m_MergeFieldListe As ArrayList

Public Sub AddinMailMergeStartMenus_Click(ByVal control As Office.IRibbonControl)
    Globals.ThisAddIn.Application.CommandBars.ExecuteMso(control.Tag)
    RibbonNeuInitialisieren(Globals.ThisAddIn.Application.ActiveDocument)
End Sub

Public Sub RibbonNeuInitialisieren(ByVal doc As Word.Document)
    ribbon.Invalidate()
    Dim docTypNormal As Word.WdMailMergeMainDocType = _
        Word.WdMailMergeMainDocType.wdNotAMergeDocument

    'Falls das aktive Dokument ein Seriendruckdokument ist,
    'die Liste der Seriendruckfelder aktualisieren
    If doc.MailMerge.MainDocumentType <> docTypNormal Then
        If Not m_MergeFieldListe Is Nothing Then m_MergeFieldListe = Nothing
        m_MergeFieldListe = New ArrayList
        m_MergeFieldListe = FieldTools.MergeFieldListeHolen(doc)
    Else
        If Not m_MergeFieldListe Is Nothing Then
            m_MergeFieldListe.Clear()
        End If
    End If
End Sub
```

Die meisten Anpassungen der Registerkarte *Sendungen* wurden in der selbst definierten Gruppe *Felder schreiben und einfügen* vorgenommen, die die gleichnamige, Word-eigene Gruppe ersetzt. Die Befehle *Adressblock*, *Grußzeile* sowie *Übereinstimmende Felder festlegen*, die nicht immer zufrieden stellend funktionieren, wurden ersatzlos entfernt. Das Menü *Regeln* wurde selbst definiert, um die Funktionalität des Befehls *Wenn...Dann...Sonst* zu erweitern (diese Änderung wird im Abschnitt »Ein Custom Task Pane im COM-Add-In definieren« weiter hinten in diesem Kapitel näher vorgestellt). Zudem wurden die Einträge *Nächster Datensatz wenn* sowie *Datensatz überspringen wenn* entfernt, da diese seit Jahren als Auslaufmodelle gelten und vom Benutzer nicht eingesetzt werden sollen.

An Stelle der Schaltfläche *Übereinstimmende Felder festlegen* steht ein button-Steuerelement mit der Beschriftung *1:n Tabelle*. Damit werden das Dialogfeld *Datenbank* und, sofern es erfolgreich bestätigt wird, die selbst definierte Gruppe *Datenbanktabelle einfügen* eingeblendet.

Neben der Beschriftung dieser Schaltfläche befindet sich ein Symbol. Im Gegensatz zu einer in einem Dokument gespeicherten Ribbon-Erweiterung (wie in Kapitel 17 beschrieben) müssen für die Ribbon-Erweiterung eines Add-Ins jedes Mal externe Grafiken geladen werden. Hierfür können das `getImage`-Attribut und sein Callback benutzt werden, um die Grafiken für jedes Steuerelement einzeln zu laden. Effizienter, falls die Grafiken während der Word-Sitzung statisch bleiben, ist das Attribut `loadImage` des `customUI`-Elements.



Die mit dem Attribut verbundene Prozedur wird nur einmal, beim Laden der Ribbon-Erweiterung, ausgeführt. Die Funktion erhält von jedem Steuerelement der Ribbon-Erweiterung, das ein `image`-Attribut enthält, dessen Wert (der Name der Grafik). Die Grafikdatei wird in den Speicher geladen und in der Form einer `stdole IPictureDisp` an die Ribbon-Erweiterung zurückgegeben. Da dieser Datentyp ein COM-Datentyp ist, kann .NET Framework ihn nicht von sich aus zur Verfügung stellen; es muss zuerst eine Konversion stattfinden, wie das Listing 10.24 aufzeigt. (In diesem Beispiel wurde die Grafik dem Projekt als Ressource hinzugefügt und befindet sich im gleichen Ordner wie die anderen Projektdateien.)

**Listing 10.24** Eine von Visual Studio geladene Grafik in einen `stdole IPictureDisp`-Datentyp umwandeln

```
Public Function ribbon_getImages(ByVal imageID As String) As stdole.IPictureDisp
    Return PictureConverter.ImageToPictureDisp(
        My.Resources.ResourceManager.GetObject(imageID))
End Function

Friend Class PictureConverter
    Inherits AxHost

    Public Shared Function ImageToPictureDisp(ByVal image As Image) _
        As stdole.IPictureDisp
        Return CType(GetIPictureDispFromPicture(image), stdole.IPictureDisp)
    End Function
End Class
```

#### HINWEIS

VSTO verweist nicht automatisch auf das PIA für die COM-Bibliothek *stdole*. Um darauf zu verweisen, wählen Sie im Kontextmenü des Projektnamens den Eintrag *Verweis hinzufügen* und in der Registerkarte *COM* den Eintrag *OLE Automation*.

Bei Betätigung der Schaltfläche *1:n Tabelle* wird das Word-Dialogfeld *Datenbank* eingeblendet, das in den Kapiteln 7 und 9 ausführlich beschrieben ist. Damit kann der Benutzer eine Tabelle aus einer

Datenbank einfügen und auch wahlweise dynamisch verknüpfen. Falls der Benutzer dies erfolgreich tut, wird die Gruppe *Datenbanktabelle einfügen* (siehe Abbildung 10.15) eingeblendet.

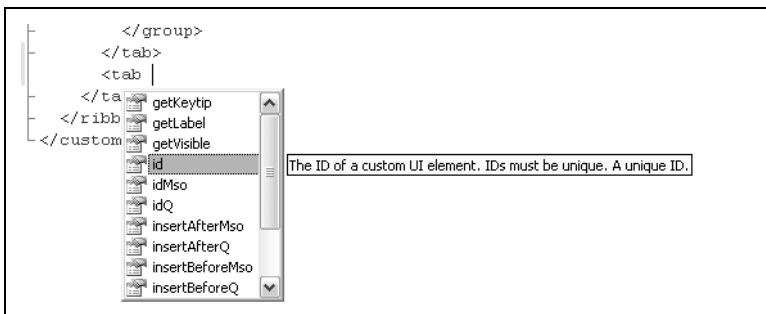
Diese ermöglicht die Verbindung des aktiven Seriendruckdokuments mit der gerade eingefügten, mit einer Datenbank verbundenen Tabelle, um nur Zeilen anzuzeigen, die für den aktuellen Datensatz zutreffen. Hinter der Tabelle befindet sich nämlich eine *Database-Feldfunktion* mit den Angaben für die Datenbankverbindung. Die Schaltfläche *Auswahl anpassen* ergänzt die darin enthaltene Select-Anweisung mit einer Where-Klausel, die das Datensatz-identifizierende Seriendruckfeld mit dem passenden Feld in der Tabellendatenquelle verbindet. Die Namen dieser Felder werden aus dem editBox- sowie dropDown-Steuer-elementen gelesen. Einzelheiten entnehmen Sie bitte dem Beispiel-Projekt auf der CD-ROM zum Buch.

```
"SELECT * FROM 'Personalumsätze nach Land_SD' WHERE [Personal-Nr]={ Mergefeld PersonalNr }"
```

## IntelliSense für die Ribbon-Erweiterung

Wie Sie in Kapitel 17 lesen können, gestaltet sich das Erfassen des Ribbon-XML mühsam. Jede Einzelheit muss stimmen, sonst läuft es nicht. IntelliSense bei der Eingabe wäre eine unschätzbare Hilfe. Die in Visual Studio 2005 enthaltenen Werkzeuge machen es möglich, wie die Abbildung 10.21 veranschaulicht.

Abbildg. 10.21 IntelliSense für Ribbon-XML in Visual Studio 2005

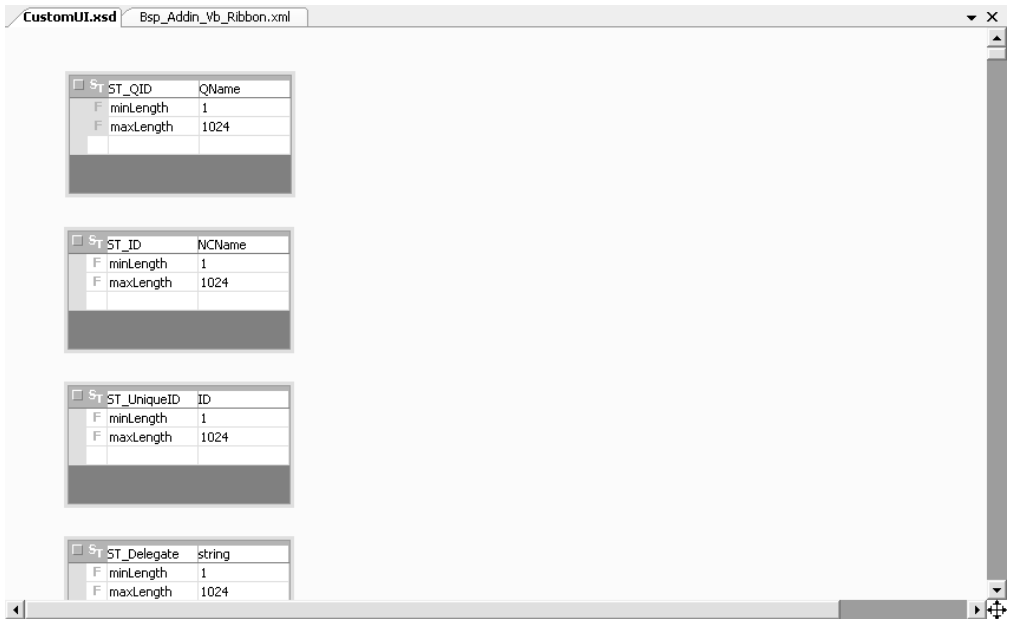


Um IntelliSense beim Erstellen einer XML-Datei zu aktivieren, muss die XML-Datei auf ein Schema verweisen. Im Fall der Ribbon-Erweiterung heißt das Schema *CustomUI.xsd*. Dieses Schema müssen Sie in der Visual Studio-Sitzung laden. Gehen Sie wie folgt vor:

1. Führen Sie in Visual Studio 2005 die Befehlsfolge *Datei/Öffnen/Datei* aus.
2. Navigieren Sie zum Ordner *C:\Programme\Visual Studio 8\Xml\Schemas\1031* und wählen Sie dort die Datei *CustomUI.xsd*.

Das Schema wird in einem eigenen Fenster geöffnet, wie in Abbildung 10.22 dargestellt, und steht fortan dem Projekt zur Verfügung.

Abbildg. 10.22 Das geladene Schema *CustomUI.xsd* bewirkt IntelliSense bei der Eingabe des Ribbon-XML



## Ein Custom Task Pane im COM-Add-In definieren

In diesem Beispiel wird das vom Seriendruck eingesetzte Dialogfeld zum Einfügen einer *If*-Feldfunktion durch einen Aufgabenbereich ersetzt. Im Gegensatz zur Word-Funktionalität können sowohl für die Dann- und Sonst-Klausel als auch für den Vergleich Seriendruckfelder eingefügt werden. Zudem darf der Benutzer Feldfunktionen als Text direkt in die Textfelder eingeben, wie die Abbildung 10.15 zeigt. Die erweiterte Funktionalität setzt die Feldfunktion aus den verschiedenen Steuerelementinhalten zusammen und fügt das Ergebnis mittels der in Kapitel 7 vorgestellten Funktion *FeldCodeEinfuegen* in das Dokument ein.

Das VSTO-Werkzeug vereinfacht den Einsatz des in Office 2007 eingeführten frei definierbaren Aufgabenbereichs. Steuerelemente werden, wie für den Aufgabenbereich *Dokumentaktionen* im Abschnitt »Der Aufgabenbereich *Dokumentaktionen*« beschrieben, hinzugefügt. Auch hier ist es ratsam, ein Benutzersteuerelement als »Behälter« für die einzelnen Steuerelemente zu benutzen. Das für dieses Beispiel eingesetzte Steuerelement ist im Entwurfsmodus in Abbildung 10.23 ersichtlich.

Abbildg. 10.23 Das benutzerdefinierte Steuerelement für das Custom Task Pane des Beispiel-Add-Ins

The image shows a custom task pane dialog box with a light gray background and a dotted border. It is divided into three main sections: 'Wenn', 'Dann', and 'Sonst'. Each section contains a text field, a dropdown menu, and a 'Seriendruckfeld einfügen' button. The 'Wenn' section has a 'Feldname' dropdown. The 'Dann' and 'Sonst' sections have large text areas. At the bottom are 'Abbrechen' and 'Einfügen' buttons.

Bei der Auswahl des Menüpunkts *Wenn...dann...sonst* wird die Prozedur *AddinDialogMailMerge-InsertIf* aus Listing 10.25 ausgeführt. Eine neue Instanz des benutzerdefinierten Steuerelements *IfFieldBauen* wird erstellt und beim Erstellen des *CustomTaskPane*-Objekts *ctp* als Parameter übergeben. (Der zweite Parameter vom Typ *Zeichenkette* legt die Beschriftung des Aufgabenbereichs fest.) Sodann wird der Aufgabenbereich eingeblendet.

Im Gegensatz zum Aufgabenbereich *Dokumentaktionen* können mehrere frei definierbare Aufgabenbereiche erstellt werden. Deshalb gibt es für das Objekt eine *Add*-Methode.

Die Listen im Aufgabenbereich sind noch leer, weshalb die Prozedur *IfFieldInitialisieren* ausführt. Die weiteren mit Steuerelementen im Aufgabenbereich verbundenen Prozeduren sind ebenfalls in Listing 10.25 enthalten.

**Listing 10.25** Code für das Initialisieren eines Custom Task Pane sowie für das benutzerdefinierte Steuerelement

```
'Code, der von der Schaltfläche im Menü "Regeln" ausführt wird.
'Initialisiert ein Custom Task Pane mit
'einer Instanz des Benutzersteuerelements ucIfField.
'Der Aufgabenbereich wird bei der Initialisierung automatisch eingeblendet.
Private Sub AddinDialogMailMergeInsertIf(ByVal doc As Word.Document)
    Dim ctp As Microsoft.Office.Tools.CustomTaskPane
    Me.ucIfField = New IfFieldBauen()
    ctp = Globals.ThisAddIn.CustomTaskPanes.Add(ucIfField, "Wann...Dann...Sonst")
    Me.ucIfField.IfFieldInitialisieren(doc, ctp)
    ctp.Width = 269
    ctp.Visible = True
End Sub

'Code in der Klasse für das benutzerdefinierte Steuerelement ucIfField
Private m_doc As Word.Document
Private m_ctp As Tools.CustomTaskPane
Private wdApp As Word.Application
Private m_FeldNamenListe As System.Collections.ArrayList

'Wird beim Einblenden des Aufgabenbereichs ausgeführt
Public Sub IfFieldInitialisieren(ByVal doc As Word.Document, _
    ByVal ctp As Tools.CustomTaskPane)
    m_doc = doc
    m_ctp = ctp
    wdApp = doc.Application
    m_FeldNamenListe = FieldTools.MergeFieldListeHolen(m_doc)

    Me.cboFeldnamen1.Items.AddRange(m_FeldNamenListe.ToArray())
    Me.cboDann.Items.AddRange(m_FeldNamenListe.ToArray())
    Me.cboSonst.Items.AddRange(m_FeldNamenListe.ToArray())
End Sub

'Wird bei der Auswahl eines Feldnamens aus einer Combobox ausgeführt.
'Schreibt die Mergefeld-Feldfunktion in die passende Textbox,
'an die Stelle der gegenwärtige Markierung.
Private Sub cboFeldnamenListe_SelectedIndexChanged(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles cboSonst.SelectedIndexChanged, _
    cboDann.SelectedIndexChanged
    Dim cboActionControl As Windows.Forms.ComboBox = sender
    Dim FeldText As String = MergeFieldText(cboActionControl)
    Select Case cboActionControl.Name
        Case "cboDann"
            Me.txtDann.Text = TextKlauselZusammenstellen(Me.txtDann, FeldText)
        Case "cboSonst"
            Me.txtSonst.Text = TextKlauselZusammenstellen(Me.txtSonst, FeldText)
        Case Else
    End Select
End Sub

Private Function TextKlauselZusammenstellen(ByVal control As Windows.Forms.TextBox, _
    ByVal FeldText As String) As String
    Dim sel As Integer, allText As String = ""
    sel = control.SelectionStart
    allText = control.Text
```



**Listing 10.25** Code für das Initialisieren eines Custom Task Pane sowie für das benutzerdefinierte Steuerelement (*Fortsetzung*)

```

If sel > 1 Then
    allText = allText.Substring(0, sel - 1) & FeldText & allText.Substring(sel)
Else
    allText = FeldText & allText.Substring(sel)
End If
Return allText
End Function

'Stellt den Text für die MergeField-Feldfunktion zusammen.
Private Function MergeFieldText(ByVal ActionControl As Windows.Forms.Control) As String
    Dim fldText As String = ""

    Try
        fldText = "{ MergeField " & ActionControl.Text & " }"
    Catch ex As Exception
        MessageBox.Show(ex.Message)
    End Try
    Return fldText
End Function

Private Sub btnOK_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles btnOK.Click
    IfFunktionEinfügen()
    ReleaseTaskPane()
End Sub

Private Sub btnCancel_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles btnCancel.Click
    ReleaseTaskPane()
End Sub

Private Sub ReleaseTaskPane()
    m_ctp.Visible = False
    m_ctp = Nothing
    m_doc = Nothing
    wdApp = Nothing
End Sub

```




Das Beispielprojekt befindet sich in der Datei *Bsp\_Addin\_VB.zip* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap10*.

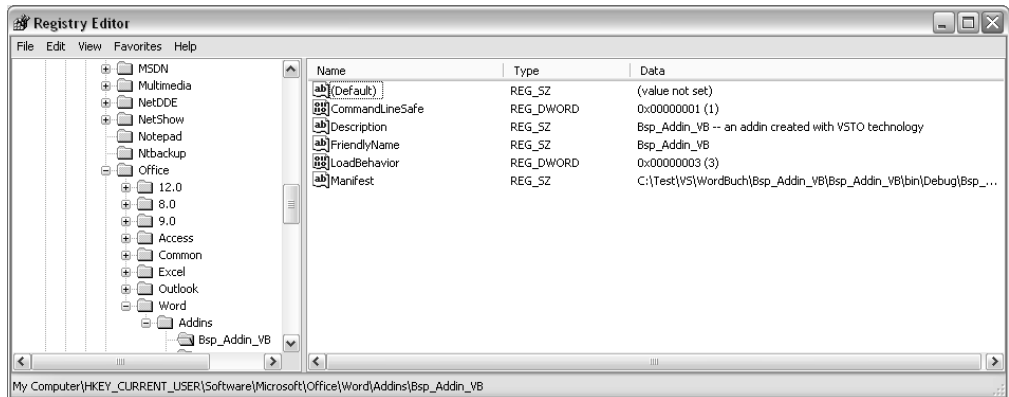
## Das Beispiel ausprobieren

Wie schon erwähnt, wird hier auf die Verteilung nicht eingegangen. Da einem COM-Add-In nicht nur vertraut, sondern es zudem korrekt in der Windows-Registry eingetragen werden muss, werden wir auch keine »Notlösung« beschreiben.

Falls Sie Visual Studio 2005 Professional oder höher sowie VSTO 2005 SE installiert haben, öffnen Sie das Beispielprojekt in Visual Studio. Durch Drücken der Taste **[F5]** wird es im Debug-Modus gestartet, wodurch das Projekt auf Ihrem Rechner als vertrauenswürdig eingestuft und gleichzeitig registriert wird. Die Word-Anwendung wird ebenfalls gestartet und das Add-In darin geladen.

Ab diesem Zeitpunkt bleibt das Add-In auf Ihrem Rechner installiert und aktiv, sofern Sie die Ordner nicht löschen. Falls Sie es entfernen wollen, sollten auch die Einträge in der Windows-Registry gelöscht werden. Im *Start*-Menü wählen Sie *Ausführen* und führen den Befehl *regedit* aus (unter Windows Vista drücken Sie die Tastenkombination  + [R], um das *Ausführen*-Dialogfeld zu öffnen). Im Registry-Editor suchen Sie die Einträge für das Add-In unter *HKEY\_CURRENT\_USER/Software/Microsoft/Office/Word/Addins* und löschen diese (Abbildung 10.24).

**Abbildg. 10.24** Der Registry-Eintrag eines VSTO COM-Add-Ins befindet sich unter *HKEY\_CURRENT\_USER*



Um das Add-In in Word lediglich auszuschalten, wechseln Sie in den *Word-Optionen* zur Kategorie *Add-Ins*. In der Dropdownliste *Verwalten* wählen Sie den Eintrag *COM-Add-Ins* aus, klicken auf *Gehe zu* und deaktivieren im darauf folgenden Dialogfeld das Kontrollkästchen für das Add-In. Über den gleichen Weg kann es später wieder aktiviert werden.

## Zusammenfassung

In diesem Kapitel wurde gezeigt, wie Word von außen ferngesteuert werden kann. In einem zweiten Schwerpunkt wurde erläutert, wie die Prozeduren eines Vorlagen-Add-Ins angesprochen werden. Zudem wurde die VSTO 2005-Technologie kurz vorgestellt.

- Dieses Kapitel hat Ihnen gezeigt, wie Makros von außen mittels der Run-Methode angestoßen werden (Seite 500) und wie Word effektiv ferngesteuert werden kann (Seite 504).
- Als weiterer Schwerpunkt wurde dargestellt, wie bereits erstellte Programmsequenzen gemeinsam genutzt werden können (Seite 508) 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 510).
- Zudem wurde veranschaulicht, wie Prozeduren in globalen Add-Ins von anderen Programmierumgebungen aus aufgerufen werden können (Seite 512).
- Es wurden die Grundlagen der Fernsteuerung der Word-Anwendung durch eine Visual Studio 2005 .NET-Anwendung erläutert (Seite 513).
- Darauf folgte eine Diskussion zum Thema Visual Studio 2005 Tools for Office (VSTO). Ab Seite 519 wurde an zwei Beispielen die Funktionalität der dokumentspezifischen sowie Add-In-Lösungen aufgezeigt.

## Kapitel 11

# Andere Programme von Word aus steuern

### In diesem Kapitel:

Microsoft Excel fernsteuern	560
Microsoft PowerPoint fernsteuern	563
Microsoft Visio fernsteuern	568
Microsoft Outlook fernsteuern	570
Microsoft Access fernsteuern	578
Auf Datenbanken zugreifen	580
Zusammenfassung	588

Dieses Kapitel widmet sich der Fernsteuerung anderer Applikationen aus Word heraus. Dabei stehen insbesondere 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 10 wurde ein kurzes Szenario erarbeitet, welches hier bei allen Applikationen als Beispiel umgesetzt wird. Anhand dieses Beispiels wird gezeigt, 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 legende Beispiel (die erste Seite einer Datei ausdrucken) für Word ist in Kapitel 10 aufgeführt.

Die meisten 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 9 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 9 den Abschnitt »Verweise auf externe Bibliotheken im VB-Editor«).

---

---

**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\Kap11\Bsp_Demo.xls"
```

---

## Microsoft Excel fernsteuern



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 12.

## 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 11.1** Ausdrucken eines Arbeitsblatts mittels Fernsteuerung von Excel

```
Sub Demo_ExcelFernsteuern()
    Const strDATEI_NAME As String = "C:\WordBuch\Beispiele\Kap11\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 11.1**    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 11.1 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 12.0 Object Library*) werden solche Berechnungsfunktionen zur Verfügung gestellt und können somit auch von einem Word-Makro genutzt werden.

In Listing 11.2 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 12 aufgeführt.

**Listing 11.2** 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 = CDBl(InputBox("Anzahl Elemente eingeben", "Kombinationen berechnen", "64"))
    dblBasis = CDBl(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 *Bsp11\_01.doc* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap11*.

## Microsoft PowerPoint fernsteuern



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 11.3    Ausdrucken einer Folie mittels Fernsteuerung von PowerPoint**

```

Sub Demo_PowerpointFernsteuern()
    Const strDATEI_NAME As String = "C:\WordBuch\Beispiele\Kap11\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
        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

```



Listing 11.3 Ausdrucken einer Folie mittels Fernsteuerung von PowerPoint (Fortsetzung)

```
Set pptApp = Nothing
System.Cursor = wdCursorNormal
End Sub
```

Anhand der eingefügten Kommentarzeilen sollte das Listing 11.3 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 5 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. 11.1 Der Quelltext auf der Folie wird in das Dokument übernommen



Das Listing 11.4 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 11.4 Einlesen der Texte aus den Platzhaltern und übertragen in ein Dokument

```
Sub Demo_PowerpointStruktur_Übertragen()
    Const strDATEI_NAME As String = "C:\WordBuch\Beispiele\Kap11\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
    End If
End Sub
```

Listing 11.4 Einlesen der Texte aus den Platzhaltern und übertragen in ein Dokument (Fortsetzung)

```

    Dim pptPara As Object

    'Neue Powerpoint-Instanz erzeugen
    Set pptApp = CreateObject("Powerpoint.Application")
#End If

    Dim doc As Word.Document

    Set doc = Documents.Add

    '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 *Bsp11\_01.doc* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap11*.

## Microsoft Visio fernsteuern



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 11.5**    Ausdrucken eines Zeichnungsblatts mittels Fernsteuerung von Visio

```
Sub Demo_VisioFernsteuern()  
    Const strDATEI_NAME As String = "C:\WordBuch\Beispiele\Kap11\Bsp_Demo.vsd"  
  
    #Const EARLYBINDING = False    'oder True  
  
    System.Cursor = wdCursorWait  
  
    'Variablen und Objekte anlegen  
    #If EARLYBINDING Then
```

Listing 11.5 Ausdrucken eines Zeichnungsblatts mittels Fernsteuerung von Visio (Fortsetzung)

```

'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
#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 11.5 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 *Bsp11\_01.doc* auf der CD-ROM zum Buch im Ordner `\Beispiele\Kap11`.

## Microsoft Outlook fernsteuern



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 11.6**    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
```

Listing 11.6 Ausdrucken eines Kalendereintrags mittels Fernsteuerung von Outlook (Fortsetzung)

```

    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

'Kalender-Ordner setzen
        Set olOrdner = olApp.Session.GetDefaultFolder(olFolderCalendar)
        Set olKalenderEintrag = olOrdner.Items(1)

'Kalendereintrag ausdrucken
        olKalenderEintrag.PrintOut
        .Quit
    End With
Else
    MsgBox "Neue Instanz von Outlook konnte nicht erzeugt werden."
End If

Set olKalenderEintrag = Nothing
Set olOrdner = Nothing
Set olApp = Nothing
End Sub

```

Anhand der eingefügten Kommentarzeilen sollte das Listing 11.6 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 olKalenderEintrag = olOrdner.Items(1)

```



Das obige Beispiel finden Sie in der Beispieldatei *Bsp11\_01.doc* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap11*.

## 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 11.2 ersichtlich, können zur gewählten Adresse zusätzliche Eigenschaften des neuen Dokuments im gleichen Dialogfeld erfasst und übertragen werden.

Abbildg. 11.2 Eigenschaften des neuen Dokuments bestimmen

Die Dokumentvorlage *Bsp11\_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 8 beschrieben.

Der Zugriff auf Outlook ist in vier eigenständige Prozeduren unterteilt. Diese werden anschließend kurz besprochen.

Listing 11.7 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
```



Listing 11.7 Deklaration und Funktion zum Starten von Outlook (Fortsetzung)

```

    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 ...
    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 erläutert, 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 11.7 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 11.8 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 11.8 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 11.9** Funktion zum Übertragen aller Kontakte in das Dialogfeld

```
Function funcOL_KontakteEinlesen()
    If Not (olAnw Is Nothing) Then

        System.Cursor = wdCursorWait

        'Kontakte-Ordner setzen und sortieren
        Set olOrdner = olAnw.Session.GetDefaultFolder(olFolderContacts)
        Set olItem = olOrdner.Items
        olItem.Sort "[FileAs]"

        'Alle Kontakte einlesen (nur das Feld »Speichern unter«)
        For Each olKontakt In olItem
            With olKontakt
                If olKontakt.Class = olContact Then 'nur Kontakte
                    If Not Len(Trim$(.FileAs)) = 0 Then
                        frmBrief.lstKontakteKürzel.AddItem .FileAs
                    End If
                Else
                    'Bei allen anderen nichts machen
                End If
            End With
        Next olKontakt
    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 11.2). In einem ersten Schritt werden nur die Kurzbezeichnungen eingelesen. Dies sind, wie in Abbildung 11.3 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. 11.3 Die Kurzbezeichnungen des Kontakts wird im Feld *Speichern unter* abgelegt

Listing 11.10 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)
```

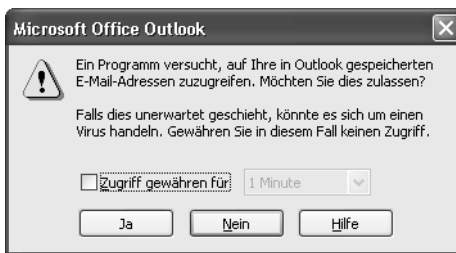
**HINWEIS**

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 (oItemFilter(1)) aus gefilterten Datensätzen eingelesen.

Im aktuellen Beispiel taucht die Sicherheitswarnung (vgl. Abbildung 11.4) von Outlook nicht auf. Sie wird nur eingeblendet, wenn ein anderes Programm auf die E-Mail-Adressen von Outlook zugreifen möchte.

Abbildg. 11.4 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 *Bsp11\_02a.dot* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap11*.

## 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 oEmailItem = oApp.CreateItem(oMailItem)
```

Ist das neue MailItem-Objekt erstellt, wird es, wie in Listing 11.11 beschrieben, mit allen notwendigen Angaben wie Empfänger, Betreff und Mitteilungstext ausgefüllt. Zum Schluss wird die aktuelle Word-Datei als Anhang der Mail zugewiesen.

Listing 11.11 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 Sie das geänderte Dokument. Bitte prüfen!" & vbCrLf & "MfG"
    ' 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 11.12 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

```

**Listing 11.12**    Prüfen, ob das aktuelle Dokument gespeichert ist (*Fortsetzung*)

```

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 speichern werden" & vbCrLf & _
        "Jetzt speichern?", vbInformation + vbYesNo, c_Title)
    If intRet = vbYes Then
        ActiveDocument.Save
    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 *Bsp11\_02b.doc* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap11*.

## Microsoft Access fernsteuern



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 11.13 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 11.13**    Die Access-Anwendung öffnen und ein Formular anzeigen

```

Sub AccessFormularEinblenden()
    Dim accApp As Access.Application
    Dim strDB As String

```

Listing 11.13 Die Access-Anwendung öffnen und ein Formular anzeigen (Fortsetzung)

```

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 11.14 wird der Bericht direkt gedruckt und die Access-Instanz anschließend wieder geschlossen.

Listing 11.14 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 *Bsp11\_03.doc* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap11*. 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, stellt Office ab Version 2002 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 11.14 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 Word 2003 und früher wird die Tabelle in der Benutzerschnittstelle über die Schaltfläche *Datenbank einfügen* aus der Symbolleiste *Datenbank* eingefügt.





In der Multifunktionsleiste von Word 2007 steht der Befehl *Datenbank einfügen* standardmäßig nicht zur Verfügung. Dieser kann jedoch der Symbolleiste für den Schnellzugriff hinzugefügt werden. Wie diese angepasst werden kann, wurde bereits in Kapitel 1 erläutert.

#### 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 7 zusammengefasst.

Das Einfügen einer Datenbanktabelle benötigt vier Schritte. Diese sind in Abbildung 11.5 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.

Abbildg. 11.5 Daten aus einer Access-Datenbank mit der Feldfunktion *Database* in Word als Tabelle verknüpfen

Kategorie-Nr	Kategorienname	Beschreibung	Abbildung
1	Getränke	Alkoholfreie Getränke, Kaffee, Tee, Bier	151C2F00
2	Gewürze	Süße und saure Soßen, Gewürze	151C2F00
3	Süßwaren	Desserts und Süßigkeiten	151C2F00
4	Milchprodukte	Käsesorten	151C2F00
5	Getreideprodukte	Brot, Kracker, Nudeln, Müsli	151C2F00
6	Fleischprodukte	Fleisch-Fertiggerichte	151C2F00
7	Naturprodukte	Getrocknete Früchte, Tofu usw.	151C2F00
8	Meeresfrüchte	Meerespflanzen und -früchte, Fisch	151C2F00

**Datenbank**

Datenquelle  
  
 Daten: C:\WordBuch\CD-ROM\...\Wordwind.mdb

Datenoptionen

Einfügen der Daten in das Dokument

**Daten einfügen**

Datensätze einfügen  
☒ Alle  
☐ Von:  Bis:

☒ Daten als Feld einfügen

```
{DATABASE \d "C:\WordBuch\CD-ROM\Wordwind.mdb" \c
"DSN=Microsoft Access-Datenbank;DBQ=C:\WordBuch\CD-
ROM\Wordwind.mdb;DriverId=25;FIL=MS
Access;MaxBufferSize=2048;PageTimeout=5;UID=admin;" \s "SELECT * FROM
'Kategorien' \i "27" \b "63" \h}
```

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.



Die Beispieldatei *Bsp11\_03.doc* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap11*. Die Beispieldatenbank *Nordwind.mdb* befindet sich im Ordner *\Datenbank*.

### Daten direkt ansprechen

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 anwendungspezifischen ODBC-Treiber umgesetzt. Dies besorgt bei ADO ein OLE DB-Provider für die Datenschnittstelle. Das Verbindungsprotokoll (»Connection string«), das die Verbindung herstellt, 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>.

#### HINWEIS

Während des Verfassens dieses Buches war die erwähnte Seite mit den Verbindungsinformationen zum neuen Access 2007-Dateiformat (\*.accdb) noch nicht aktualisiert. Der »Connection string« für \*.accdb-Datenbanken müsste jedoch wie folgt aufgebaut werden (vergleiche dazu Listing 11.15).

```
objConnection.Open "Provider = Microsoft.ACE.OLEDB.12.0; " &
    "Data Source = C:\Datenbanken\Test.accdb;"
```

Für unser Beispiel verwenden wir als Datenquelle die Tabelle *Personal* (Abbildung 11.6) der Beispieldatenbank *Nordwind.mdb*. Diese Datei ist im Lieferumfang von Office Professional enthalten. Der Code in der Dokumentvorlage (*Bsp11\_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. 11.6** 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 11.7. 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 16 beschrieben. Das Datum bleibt ebenfalls statisch, es wird anhand einer *CreateDate*-Feldfunktion generiert.

Abbildg. 11.7 Die Daten aus der Tabelle wurden zusammengestellt und in die Textmarken geschrieben

Northwind GmbH  
Personalabteilung

München, den 11. September 2008

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

1

Der Code für die beschriebene Aufgabe befindet sich in Listing 11.15. 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.

In *AutoNew* ist der nächste Schritt der Aufruf der Funktion *SymbolleisteMitComboErstellen*. Diese erstellt die temporäre Symbolleiste *Briefschreiben* 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 11.15**    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 Dropdownliste 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.
```

Listing 11.15 Daten über ADO-Verbindungen und Recordsets lesen und in ein Word-Dokument schreiben (Fortsetzung)

```

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
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 & vbCr & RS.Fields("Straße").Value & _
    vbCr & RS.Fields("Ort").Value & " " & RS.Fields("PLZ").Value & _
    vbCr & 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.

```

**Listing 11.15**    Daten über ADO-Verbindungen und Recordsets lesen und in ein Word-Dokument schreiben (*Fortsetzung*)

```
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 *Bsp11\_04.dot* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap11*.

## 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 7 im Abschnitt zu den Feldfunktionen, aufgezeigt. In Kapitel 12 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 Fel-

der eines nach dem andern mit Daten zu bestücken, werden die Daten in Zeichenketten gesammelt und mit der SQL-Anweisung `INSERT INTO` der Datentabelle hinzugefügt.

Die Ausgangslage ist in Abbildung 11.8 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 11.16 aus.

Abbildg. 11.8 Die Adressenangaben aus dem Formular werden einer Excel-Tabelle hinzugefügt

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 11.8 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 7 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 11.16 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
```

**Listing 11.16**    Daten aus Word-Formularfeldern über eine ADO-Verbindung als neue Zeile einer Excel-Tabelle hinzufügen (*Fortsetzung*)

```

Dim strSQL As String

Set doc = ActiveDocument
strSQL = ""
strFeldListe = "("
strWertListe = "("
For Each ffld In doc.FormFields
    strFeldListe = strFeldListe & ffld.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 *Bsp11\_05.dot* finden Sie auf der CD-ROM zum Buch im Ordner \Beispiele\Kap11. 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 Zugriff auf Datenbanken gewidmet:

- In diesem Kapitel wurde zunächst gezeigt, wie Excel ferngesteuert (Seite 560 ff.) oder zur Berechnung von komplexen Funktionen (Seite 562) herangezogen werden kann.
- Beim Steuern von PowerPoint (Seite 563 ff.) wurde erläutert, wie der Inhalt einer Präsentation in ein Dokument eingelesen (Seite 566) werden kann.
- Ein einfaches Beispiel zum Steuern von Visio wurde auf Seite 568 behandelt.
- Beim Zugriff auf Outlook (Seite 570 ff.) wurde vermittelt, wie auf die Kontakte zugegriffen (Seite 572) oder ein Dokument als E-Mail versendet (Seite 576) werden kann.
- Das Fernsteuern von Access (Seite 578 ff.) und der Zugriff auf Datenbanken (Seite 580 ff.) wurde ebenfalls behandelt.



## Kapitel 12

# Eingebettete OLE-Objekte

### In diesem Kapitel:

Excel-Tabellenobjekte	593
Excel-Diagramme	600
MS Graph-Diagramme	606
WordArt	610
ActiveX-Steuerelemente	615
Zusammenfassung	624

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 ist möglich, 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, dieser wurde in Kapitel 1 vorgestellt.

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 zugewiesen. Die folgende Codezeile wurde bei der Erstellung eines Excel-Objekts aufgezeichnet:

```
Selection.InlineShapes.AddOLEObject ClassType:="Excel.Sheet.8", LinkToFile:=False, _
    DisplayAsIcon:=False
```



2007

Die Automatisierung von Excel-Tabellen in Word 2007 unterscheidet sich, trotz des neuen Dateiformats, nicht wesentlich von früheren Versionen. Tabellen im alten binären Dokumentformat sowie aus konvertierten Arbeitsmappen werden abgearbeitet. Arbeitsmappen mit gemischten Excel-Tabellen aus der aktuellen sowie Vorgängerversionen können erfolgreich automatisiert werden (siehe das Listing 12.5). Die Syntax, um eine Excel 2007-Kalkulationstabelle in Word 2007 zu erstellen, lautet folgendermaßen:

```
Selection.InlineShapes.AddOLEObject ClassType:="Excel.Sheet.12", 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 7 im Abschnitt zu Grafiken.

OLE-  
Format

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

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 nachfolgend 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.

Automati-  
sierung

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 12.1.

**Listing 12.1** 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
    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
```



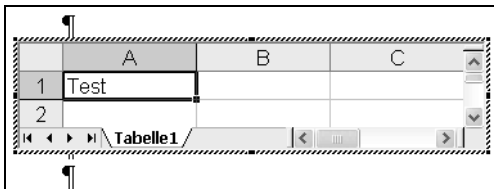
Die Beispieldatei *Bsp12\_01.doc* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap12*.

Danach muss mit dem Objektmodell des OLE-Servers gearbeitet werden, wie die letzten Codezeilen veranschaulichen. Das Resultat sehen Sie in Abbildung 12.1.

**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. 12.1** 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.

**Bildschirm** Die programmiertechnische Bearbeitung von `OLE`-Objekten ist auf dem Bildschirm sichtbar. Bislang 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 7 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 12.2. Es veranschaulicht den Gebrauch von `Late Binding` – alle Objektvariablen für das Excel-Objekt sind 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 12.2 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. 12.2** 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							

Steuern und gesteuert werden

**Listing 12.2** 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

    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
    'Syntax für Excel 2007
    'Set o = ActiveDocument.InlineShapes.AddOLEObject( _
        "Excel.Sheet.12", 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 12.3 zeigt, dass die Prozedur aus Listing 12.3 beim Erstellen des Excel-Objekts auch dessen sicht- baren Bereich bestimmt.

**Abbildg. 12.3** 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 12.2 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.

**HINWEIS**

Mehr zum Thema Early sowie Late Binding lesen Sie im Kapitel 9.

**Listing 12.3** Eine Excel-Mappe erstellen und anschließend 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 & "\Bsp12_01.xls"
    'Syntax für Excel 2007
    'strArbeitsmappenPfad = ActiveDocument.Path & "\Bsp12_01.xlsx"
    '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
End Sub
```

**Listing 12.3** Eine Excel-Mappe erstellen und anschließend in Word importieren (Fortsetzung)

```
'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 12.4** 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\\Kap12\\Bsp12_01.xls";
        //Syntax für Excel 2007
        strArbeitsmappenPfad = "C:\\WordBuch\\Beispiele\\Kap12\\Bsp12_01.xlsx";
        //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";
    }
}
```



**Listing 12.4** 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)

```
x1Blatt.get_Range("A2", Type.Missing).Select();
//Überflüssige Arbeitsblätter entfernen.
for (int zaehler = x1Mappe.Sheets.Count; zaehler > 1; zaehler --)
{
    x1.Worksheet x1VorigesBlatt = (x1.Worksheet) x1Mappe.Worksheets[zaehler];
    x1VorigesBlatt.Delete();
    x1VorigesBlatt = null;
}
//Die Mappe speichern, schließen und Excel beenden.
x1Mappe.Close(true, arbeitsmappenPfad, Type.Missing);
x1App.Quit();
x1Blatt = null;
x1Mappe = null;
wdMarshal.ReleaseComObject(x1App);
x1App = 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");
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 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 *Bsp12\_01.doc* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap12*.

Bearbeiten

In Listing 12.1 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 12.5 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` oder `Excel.Sheet.12` (Excel-Arbeitsblatt), wird jeweils die Prozedur *Excel-ObjektBearbeiten* 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 12.4).

**Abbildg. 12.4** 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 12.5** 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
            Select Case olef.ClassType
                Case "Excel.Sheet.8", "Excel.Sheet.12"
                    '...wird es weiter bearbeitet
                    ExcelObjektBearbeiten olef, bExcel
            Case Else
            End Select
        End If
    End For
End Sub

```

Listing 12.5 Alle Excel-Tabellenobjekte eines Dokuments in einem Excel-Fenster öffnen und formatieren (Fortsetzung)

```

    End If
Next
'Anschliessend durch alle Shapes Schleifen.
For Each shp In ActiveDocument.Shapes
    If shp.Type = msoEmbeddedOLEObject Then
        Set olef = shp.OLEFormat
        Select Case olef.ClassType
            Case "Excel.Sheet.8", "Excel.Sheet.12"
                ExcelObjektBearbeiten olef, bExcel
            Case Else
                End Select
        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

```

**Listing 12.5** Alle Excel-Tabellenobjekte eines Dokuments in einem Excel-Fenster öffnen und formatieren (Fortsetzung)

```
If Not bExcel Then
    xApp.Quit
End If
Set xApp = 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 *Bsp12\_01.doc* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap12*.

## 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.

**PROFITTIPP**

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.



2007

In Excel 2007 befindet sich der Befehl *Als Bild* im Dropdownmenü zur Schaltfläche *Einfügen* auf der Registerkarte *Start*.

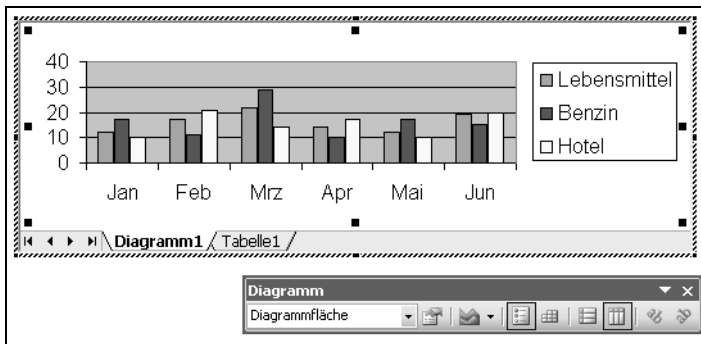
Der Makrorekorder verrät folgende Syntax, um ein Excel-Diagramm einzufügen, wobei der Klassentyp »Excel.Chart.8« verwendet wird:

```
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 12.5), welches die Daten für das Diagramm enthält. Der Automatisierungscode hat demzufolge zwei Aufgaben zu erledigen:

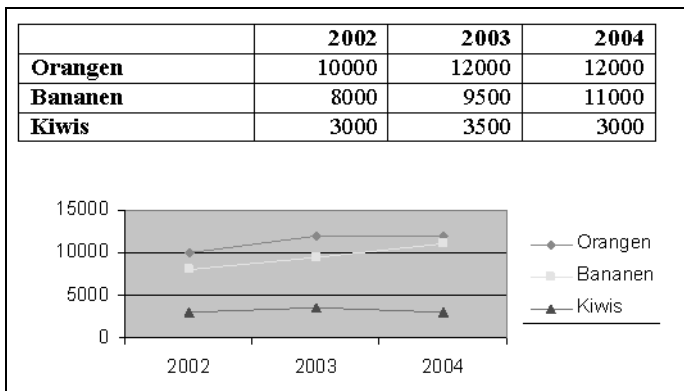
- Die im Tabellenblatt hinterlegten Beispieldaten durch die benötigten zu ersetzen
- Das Diagramm nach Wunsch zu formatieren

Abbildg. 12.5 Standardmäßig umfasst ein neues Diagramm-Objekt, ein Diagramm- sowie ein Tabellenblatt



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 12.6, den Beispielcode in Listing 12.6.

Abbildg. 12.6 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 12.6** 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
End Sub
```

Listing 12.6 Ein Excel-Diagramm anhand einer Tabelle im Word-Dokument erstellen (Fortsetzung)

```

'Das Diagramm formatieren
DiagrammFormatieren xlDiagramm, xlBlatt

'Aufräumungsarbeiten
rs.Close
Set rs = Nothing
Set xlDiagramm = Nothing
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

```

**Listing 12.6** Ein Excel-Diagramm anhand einer Tabelle im Word-Dokument erstellen (Fortsetzung)

```

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
        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))
        .SetSourceData 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
        'Nur ab Excel 2007
        '.Format.Glow.Color = RGB(240, 0, 0)
        '.Format.Glow.Radius = 3
    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 Diagramm-Funktionalität wurde für Office 2007 überarbeitet. Die Aufzeichnung mit dem Makrorekorder liefert eine leere Prozedur, da für Office 2007-Diagramme kein OLE-Server vorhanden ist. Es kann jedoch der Code für ein »Excel.Chart.8« eingesetzt werden. Das Ergebnis ist ein Excel 2007-Diagramm, das alle Objekte des neuen Objektmodells unterstützt. Weiterhin besteht die Möglichkeit, zuerst eine Excel 2007-Kalkulationstabelle zu erstellen und anschließend ein Diagrammblatt einzufügen, wie das Listing 12.7 veranschaulicht.

**Listing 12.7** Der Weg zu einem Excel 2007-Diagramm führt über das Erstellen eines Excel-Tabellen-Objekts

```
Sub Excel2007DiagrammErstellen()
    Dim rng As Word.Range
    Dim doc As Word.Document
    Dim wb As Excel.Workbook
    Dim oSheet As Excel.Worksheet
    Dim xlRng As Excel.Range
    Dim o As Object 'Das Diagramm

    Set doc = ActiveDocument
    Set rng = doc.Content
    'Das Tabellen-Objekt wird am Dokumentende eingefügt.
    rng.Collapse wdCollapseEnd
    Set wb = ActiveDocument.InlineShapes.AddOLEObject( _
        "Excel.Sheet.12", Range:=rng).OLEFormat.Object
    Set oSheet = wb.Sheets(1)
    oSheet.Name = "Testdaten"
    'Die Daten in die Tabelle schreiben.
    oSheet.Range("B1").Value = "2004"
    oSheet.Range("C1").Value = "2005"
    oSheet.Range("D1").Value = "2006"
    oSheet.Range("E1").Value = "2007"
    oSheet.UsedRange.Font.Bold = True
    oSheet.Range("A2").Value = "Orangen"
    oSheet.Range("B2").Value = 10000
    oSheet.Range("C2").Value = 12000
    oSheet.Range("D2").Value = 12000
    oSheet.Range("E2").Value = 15000
    oSheet.Range("A3").Value = "Bananen"
    oSheet.Range("B3").Value = 8000
    oSheet.Range("C3").Value = 9500
    oSheet.Range("D3").Value = 11000
    oSheet.Range("E3").Value = 10000
    oSheet.Range("A4").Value = "Kiwis"
    oSheet.Range("B4").Value = 3000
    oSheet.Range("C4").Value = 3500
    oSheet.Range("D4").Value = 3000
    oSheet.Range("E4").Value = 4000
    Set xlRng = oSheet.UsedRange
    'Ein Diagrammblatt einfügen...
    Set o = wb.Charts.Add
    '...und mit dem Datenbereich verbinden.
    o.SetSourceData xlRng
    'Als Linien-Diagramm festlegen
    o.ChartType = xlLine
    DiagrammFormatieren o, oSheet 'Ruft die Prozedur in Listing 12.6
End Sub
```



Die Beispieldateien *Bsp12\_02.doc* und *Bsp12\_02\_2007.docm* finden Sie auf der CD-ROM zum Buch im Ordner `\Beispiele\Kap12`.

## MS Graph-Diagramme

Die Funktionalität hinter MS Graph ist die gleiche wie in Excel 2003 und früher. Nur die Schnittstelle für die Daten ist anders als in Excel, zudem bietet MS Graph keine Unterstützung der Farbpalette. 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 positionieren.

Der Makrorekorder ergibt in allen Word-Versionen 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 12.8. Wo die gleichen Prozeduren in beiden Listings gebraucht werden, sind sie nur in Listing 12.6 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 auch 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 12.8** 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
```

Listing 12.8 Ein MS Graph-Diagramm einfügen und formatieren (Fortsetzung)

```

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
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 vbCr
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

'Aufräumungsarbeiten
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
    End With
End Sub

```

**Listing 12.8** Ein MS Graph-Diagramm einfügen und formatieren (Fortsetzung)

```

.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.
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 12.9 stellvertretend auch für Excel-Diagramme.

**Listing 12.9** 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);
    }
}

```

Listing 12.9 Ein MS Graph-Diagramm einfügen und formatieren (C#-Code) (Fortsetzung)

```

//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();
//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,

```

**Listing 12.9** Ein MS Graph-Diagramm einfügen und formatieren (C#-Code) (Fortsetzung)

```

        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);
            // 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 *Bsp12\_03.doc* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap12*.

## 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 für Office 2003 und früher von einer Drittfirma – MathType – lizenziert hat.



WordArt steht in Word 2007 unverändert zur Verfügung, wurde jedoch in den anderen Office 2007-Anwendungen durch eine neue Grafikfunktionalität ersetzt. Bisheriger Code funktioniert weiterhin, liefert jedoch andere Ergebnisse. Der Formel-Editor wurde vollständig durch eine neue Funktionalität ersetzt. Diese ist im Objektmodell unter der Bezeichnung »OMath« zu finden. Dies bedeutet, Formeln können neu, in Office 2007, programmatisch erstellt und bearbeitet werden. Dieser Teil des Objektmodells wird im vorliegenden Buch nicht behandelt.

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 7 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:

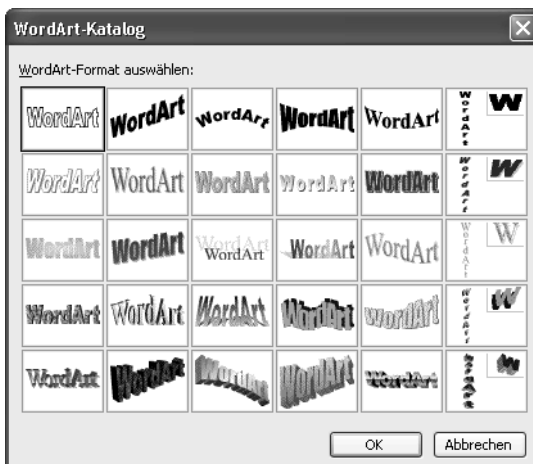
```
AddTextEffect(PresetTextEffect, Text, FontName, FontSize, FontBold, FontItalic, Left, Top,
    [Anchor])
```

#### HINWEIS

Bekanntlich kann in Word 2002 und 2003 unter *Extras/Optionen* auf der Registerkarte *Bearbeiten* (Word 2007: in der Kategorie *Erweitert* der *Word-Optionen* im Abschnitt *Ausschneiden, Kopieren und Einfügen*) 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 12.7). 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.

Abbildg. 12.7 Im ersten Schritt wird die Grundform des WordArt-Objekts aus dem *WordArt-Katalog* gewählt

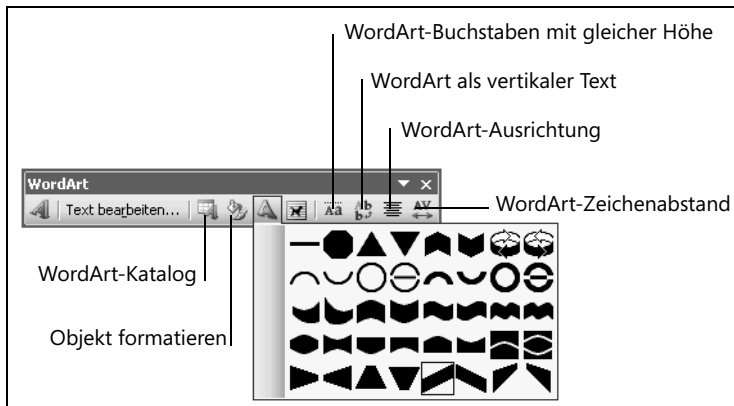


PresetTextEffect erwartet einen MsoPreSetTextEffect-Konstantwert. Dieser ist von 0 bis 29 durchnummeriert, beschreibt den Effekt also nicht. Er entspricht jedoch den Abbildungen im Dialogfeld *WordArt-Katalog*, reihenweise von links oben nach rechts unten.

Diese Grundformen sind vordefinierte Zusammenstellungen der erweiterten Formatierungsmöglichkeiten, die die Zeichnungs- und WordArt-Symbolleisten (Abbildung 12.8) 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.

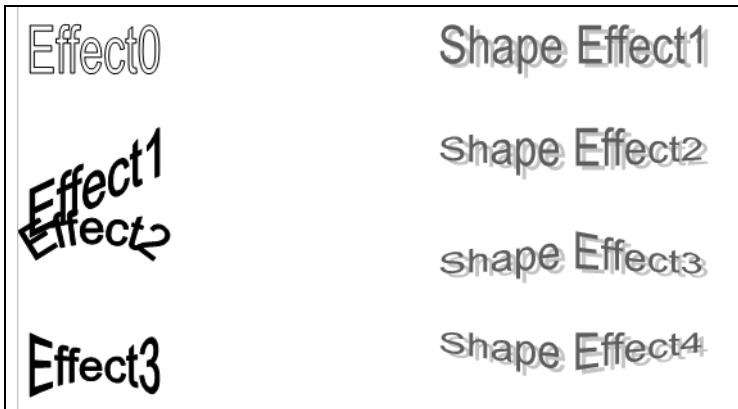
Im Objektmodell entsprechen die Formen im Dropdownfeld der PresetShape Eigenschaft des Shape-Objekts. Sie erwartet einen von 40 MsoPresetTextEffectShape-Konstantwerten. Die Konstantwertbezeichnungen sind in diesem Fall beschreibend und in der Hilfe sowie dem Objektkatalog aufgelistet. Die numerischen Werte entsprechen auch hier reihenweise den Abbildungen des Dropdownfeldes von oben links nach unten rechts.

**Abbildg. 12.8** 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 12.10. Es wird durch alle Konstantwerte geschleift und für jeden wird 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 12.9).



Abbildg. 12.9 Wirkung der *MsoPresetTextEffect*- und *MsoPresetTextEffectShape*-KonstantwerteListing 12.10 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 vbCrLf
    Next

    'Dafür sorgen, dass genügend Absätze für alle MsoPresetTextEffectShape
    'Variationen vorhanden sind.
    rng.InsertAfter String(40 - doc.Paragraphs.Count, vbCrLf)
    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 12.11** 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");
    }

    //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 Benutzerschnittstelle kein Gegenstück mehr. Sie können es jedoch mit der Prozedur in Listing 12.12 anbieten.

Listing 12.12 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

```



Die Beispieldatei *Bsp12\_04.doc* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap12*.

**HINWEIS**

Ein weiteres Code-Beispiel für die Automatisierung von WordArt finden Sie im Bonuskapitel VIII auf der CD-ROM zum Buch im Ordner *\Bonus\KapVIII*.

## ActiveX-Steuerelemente

In Kapitel 7 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.



2007

Für Word 2007 sind die neuen Inhaltssteuerelemente (»Content Controls«, siehe Kapitel 7) ActiveX-Steuerelementen vorzuziehen. ActiveX-Steuerelemente stehen in Word 2007 jedoch weiterhin zur Verfügung und werden in der gleichen Liste mit den Formularfeldern aufgeführt. Die Beispiele in diesem Kapitel funktionieren auch unter Word 2007.

**WICHTIG**

Weil ActiveX-Steuerelemente über eine VBA-Schnittstelle verfügen, lösen sie die Makrosicherheitswarnung aus (mehr darüber steht in Kapitel 1 beschrieben).

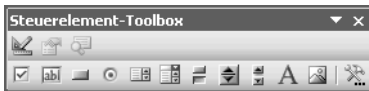
Wie andere Objekte auch, werden ActiveX-Steuerelemente von Word als grafische Objekte betrachtet. Entsprechend können sie mit Textfluss (Shapes-Auflistung) versehen oder *mit Text in Zeile* (InlineShapes-Auflistung) positioniert sein. ActiveX-Steuerelemente lassen sich in Word mit oder ohne Formularschutz 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 vier-eckiger Block.
- ActiveX-Steuerelemente, die sich automatisch dem Inhalt anpassen, erkennen die Dokument-ränder 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 Seiten-ansicht 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 12.10). 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.

**Abbildung. 12.10** 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

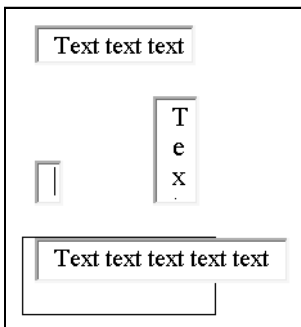



Ein Textfeld-Steuerelement kann eine genaue Größe haben oder sich dem Inhalt anpassen. Ferner ist es möglich, den Inhalt auf eine einzige Zeile zu begrenzen oder mehrere Zeilen zuzulassen. Solche Einstellungen werden im Eigenschaftenfenster (das gleiche, das auch in einem VBA-Projekt zur Verfügung steht) 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 12.11 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 12.11 zeigt.

Abbildg. 12.11 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 angegebenen Zeilenanzahl angepasst, wobei eine Zeile durch Drücken der -Taste definiert ist. Das Listing 12.13 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 12.13 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
```

**Listing 12.13** Die Höhe eines Textfeldes der Zeilenanzahl im Feld anpassen (Fortsetzung)

```
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 12.13 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 12.14 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 `GetFocus` 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 12.14** Navigieren der Steuerelemente mit Hilfe der *LostFocus*- und *GetFocus*-Ereignisse

```
Private b_LOSTFOCUS As Boolean
Private o_GotFocus As Object

Private Sub xTextBox1_LostFocus()
    If o_GotFocus Is Nothing Then
        Set o_GotFocus = Me.xTextBox2
    End If
    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

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 *Bsp12\_05.doc* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap12*.

## 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 12.12 ist ein Beispiel dargestellt, das sich vom Aussehen her zwar nicht von einem Formular mit Formularfeldern (in Kapitel 7 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 12.15 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 Briefftext frei redigieren.

Abbildg. 12.12 Ein Formular, das mit ActiveX-Steuerelementen erstellt wurde

Nordwind GmbH München	München, den 17. September 2007
<div style="display: flex; justify-content: space-between; align-items: flex-end;"> <div style="width: 40%;"> <p>[Empfänger eingeben]</p> <p>[Adresse eingeben]</p> <p>[PLZ und Ort eingeben]</p> <p>[Land eingeben oder löschen]</p> </div> <div style="width: 5%; border-left: 1px solid black; height: 60px; margin-left: 10px;"></div> </div> <p>[Betreff eingeben]</p> <p>Sehr geehrt [Anrede eingeben]</p> <p> </p> <p>Mit freundlichen Grüßen,</p>	



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 12.15 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 12.15** 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
    bDATENGÜLTIG = 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
```

**Listing 12.15** Datengültigkeit eines Formulars mit ActiveX-Steuerlementen prüfen (Fortsetzung)

```
Set doc = ActiveDocument
Set DataValidation 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
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.SelStart = 0
ctl.SelLength = 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 aufwändiger, 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 *Bsp12\_06.dot* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap12*.

---



---

**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 21 beschrieben.

---

# Zusammenfassung

In diesem Kapitel wurde auf OLE-Objekte zugegriffen, die in einem Dokument eingebettet sind. Diese Objekte wurden aus Word heraus ferngesteuert.

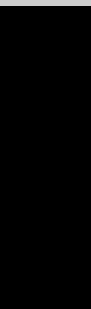
- In diesem Kapitel wurde zunächst gezeigt, wie eingebettete Excel-Tabellen (Seite 593) und Excel-Diagramme (Seite 600) bearbeitet werden können.
- Im Weiteren wurde erläutert, wie der Zugriff auf MS Graph-Diagramme (Seite 606) und Word-Art-Objekte (Seite 610) erfolgen kann.
- Ein dritter und letzter Schwerpunkt in diesem Kapitel bildet das Einbinden von ActiveX-Steuer-elementen (Seite 615) in ein Dokument.

# Teil D

## Optimierung der Benutzerschnittstelle

### In diesem Teil:

<b>Kapitel 13</b>	Anwendungsoptionen	627
<b>Kapitel 14</b>	Speicherort der Anpassungen	655
<b>Kapitel 15</b>	Mit Dialogfeldern arbeiten	665
<b>Kapitel 16</b>	Menüs und Symbolleisten	707
<b>Kapitel 17</b>	Multifunktionsleisten (Ribbons)	723
<b>Kapitel 18</b>	Tastaturbelegungen	761
<b>Kapitel 19</b>	Aufgabenbereiche	779
<b>Kapitel 20</b>	Interne Word-Befehle übersteuern	807
<b>Kapitel 21</b>	Zugriff auf den Visual Basic-Editor (VBE)	813



## Kapitel 13

# Anwendungsoptionen

### In diesem Kapitel:

Dokumentvorlage <i>Normal.dot</i> konfigurieren	628
Benutzereinstellungen abspeichern	641
Dokumenteinstellungen abspeichern	649
Zusammenfassung	654

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.
- Die Einstellungen für das *Zeichnungsraster* sowie der Standard für die Autoformen werden im Abschnitt »Zeichnungselemente optimieren« den eigenen Bedürfnissen entsprechend angepasst.
- 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 Programmweiterungen, 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 10 und in Kapitel 14.

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 13.1 in der Titelleiste erscheint.

**Abbildg. 13.1**

 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.

In Word 2007 befinden sich diese Einstellungen auf der Registerkarte *Seitenlayout*.



2007

## 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.



2007

Um das Dialogfeld für alle Dokumenteigenschaften einzublenden, muss über die *Office*-Schaltfläche der Befehl *Vorbereiten/Eigenschaften* angewählt werden. Word 2007 blendet eine Leiste mit den meistgebrauchten Dokumenteigenschaften ein. Um das Dialogfeld einzublenden, muss auf den Pfeil neben der Beschriftung *Dokumenteigenschaften* geklickt und im Dropdownmenü der Eintrag *Erweiterte Eigenschaften* ausgewählt werden, wie in Abbildung 13.2 dargestellt.

Abbildg. 13.2 Die Schnittstelle für Dokumenteigenschaften in Word 2007

**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 vorbelegt werden. Im Weiteren ist es möglich, auf der Registerkarte *Anpassen* des Eigenschaftenfensters 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.

In Word 2007 befinden sich diese Einstellungen auf der Registerkarte *Ansicht*.



2007

## AutoTexte entfernen

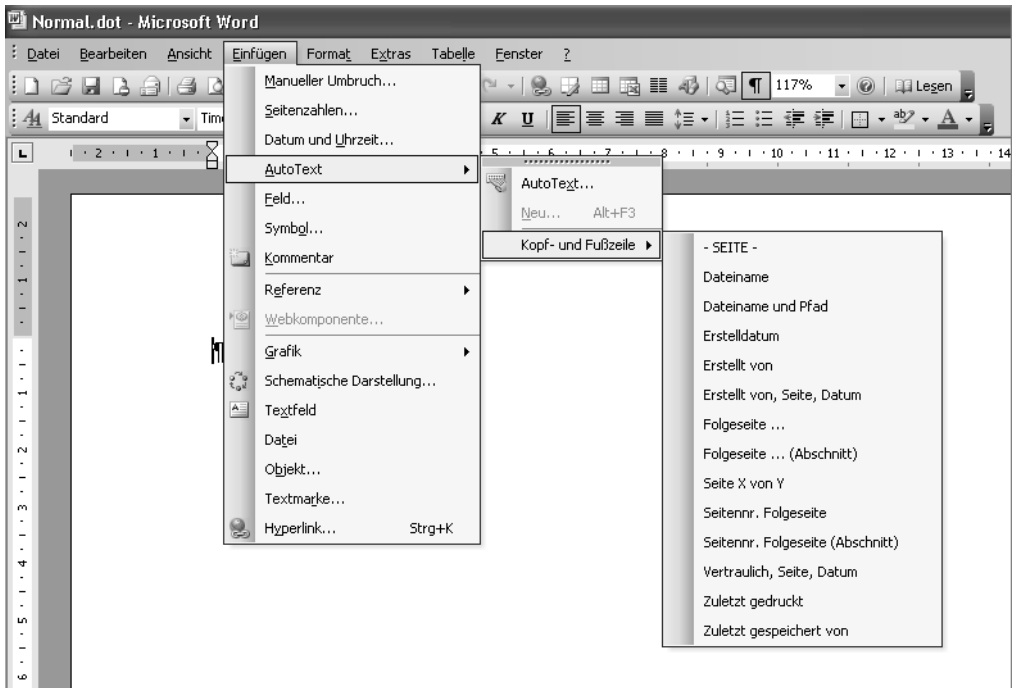
Der vierte Arbeitsschritt widmet sich den vordefinierten AutoTexten. Aus Sicht der Autoren hat der Anwender, mit Ausnahme jener AutoTexte im Bereich *Kopf- und Fußzeile*, keinen direkten Nutzen. Im Gegenteil, die Übersicht über die angebotenen AutoTexte wird erschwert.

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 13.3.

Abbildg. 13.3 Liste der nützlichen AutoTexte nach dem Löschen der überzähligen Einträge



2007

In Word 2007 entfällt dieser Arbeitsschritt, da die mitgelieferten Bausteine in einer getrennten Vorlage gespeichert sind.

#### 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 10 und in Kapitel 14. Für Word 2007 verweisen wir auf die Diskussion zu Bausteinen in dem bei Microsoft Press veröffentlichten Buch »Office 2007 – Das Profibuch«.

## Formatvorlagen definieren

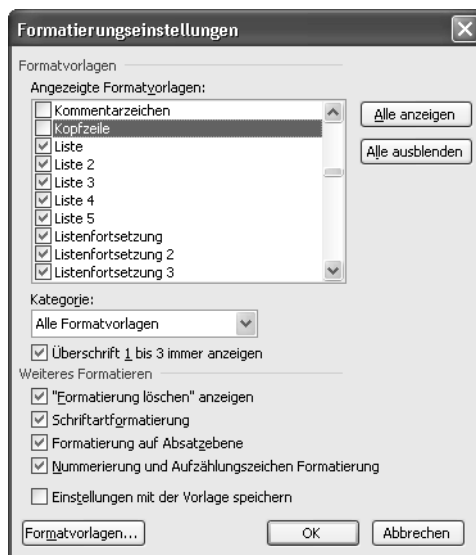
Im fünften Arbeitsschritt werden die benötigten Formatvorlagen definiert. Sie werden verstehen, dass wir hier keine Anleitung geben können, wie Sie die Formatvorlagen aufbauen müssen, da die Anforderung sehr unterschiedlich sind. Grundsätzlich sollten Sie sich jedoch zu den nachstehenden Punkten Gedanken 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 BesuchterHyperlink
- 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. 13.4

Sichtbarkeit der Formatvorlagen im Listenfeld *Angezeigte Formatvorlagen* festlegen



2007

Word 2007 stellt zu diesem Zweck ein verfeinertes Werkzeug zur Verfügung. Wir verweisen an dieser Stellen ebenfalls auf »Office 2007 – Das Profibuch«.

## 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*.



2007

In Word 2007 befinden sich die Einstellungen für die Silbentrennung auf der Registerkarte *Seitenlayout* in der Gruppe *Seite einrichten*.

## 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 Anwen-

dern 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.

In Word 2007 wählen Sie in den *Word-Optionen* die Kategorie *Dokumentprüfung* aus.



Abbildg. 13.5 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.

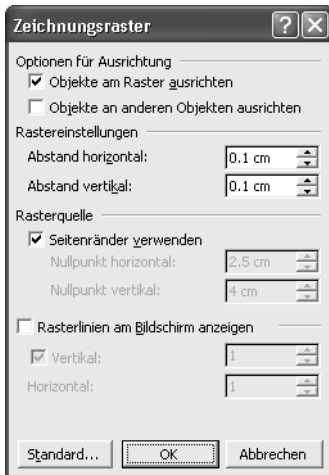
### Zeichnungselemente optimieren

Der achte Punkt, der beachtet werden sollte, widmet sich den Einstellungen für die Zeichnungselemente. Hier liegen uns die Konfiguration für das Zeichnungsraster und die Vorgabewerte für die Autoformen am Herzen.

Mit dem Zeichnungsraster wird je nach Einstellung ein sichtbares oder unsichtbares Netz über das Dokument gelegt. Die einzelnen Zeichnungselemente rasten an diesem Gitternetz automatisch ein. In unseren Breitengraden sind wir an ein metrisches System gewöhnt. Deshalb ist es sinnvoll, wenn das Gitternetz mit einer Maschenweite von 1 mm oder 5 mm definiert wird.

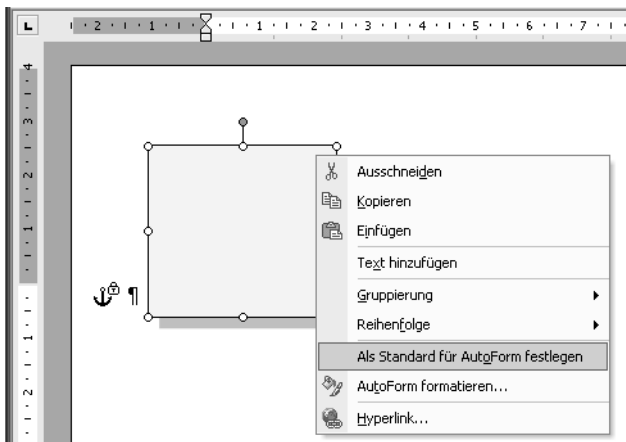
Die Einstellungen für das Zeichnungsraster können anhand der Symbolleiste *Zeichnen* festgelegt werden. Rufen Sie dort den Befehl *Zeichnen/Gitternetz* auf und tragen Sie die gewünschten Einstellungen ein.

Abbildg. 13.6 Einstellungen für das Zeichnungsraster festlegen



Ebenso sinnvoll kann es sein, dass die Vorgabewerte für die Autoformen auf die eigenen Bedürfnisse hin konfiguriert werden. Zeichnen Sie dazu eine beliebige Autoform und formatieren Sie diese entsprechend den Anforderungen (Linienart und -Farbe, Füllfarbe, Schatten usw.). Wählen Sie anschließend im Kontextmenü der Autoform den Befehl *Als Standard für Autoform festlegen* an. Die temporäre Autoform kann jetzt wieder entfernt werden.

Abbildg. 13.7 Standardwerte für Autoformen festlegen





In Word 2007 steht das Dialogfeld *Zeichnungsraster* nur dann zur Verfügung, wenn ein Zeichnungsobjekt im Dokument aktiviert ist. In der Gruppe *Anordnen* der Registerkarte *Zeichentools/Format* blenden Sie das Dropdownmenü zur Schaltfläche *Ausrichten* ein und wählen darin den Eintrag *Rastereinstellungen* aus. Das Kontextmenü für die Standardeinstellungen für AutoFormen befindet sich am gleichen Ort wie in Word 2003.

## Optionen bearbeiten

Im neunten 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.

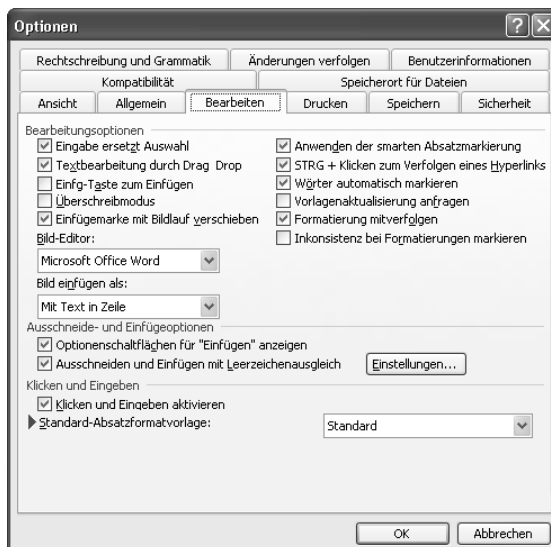
Beachten Sie die nebenstehend gezeigte Markierung in den nachfolgenden Abbildungen!

Wir verzichten darauf, jede einzelne Einstellung explizit mit den zugehörigen Werten aufzulisten. Stattdessen versuchen wir die wichtigsten Punkte kurz zu erläutern. Alle eingestellten Optionen und die zugehörigen Vorschlagswerte sind in Abbildung 13.8 bis Abbildung 13.13 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. 13.8 Festlegen der nötigen Optionen auf der Registerkarte *Bearbeiten*







2007

In Word 2007 befindet sich diese Option in den *Word-Optionen* in der Kategorie *Erweitert* im Abschnitt *Bearbeitungsoptionen*.

### 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 Schriftarten in die einzelnen Dokumente eingebunden werden. So ist sichergestellt, dass jedes Dokument auch auf firmenfremden Arbeitsgeräten über das gewünschte Layout verfügt.

#### HINWEIS

Beachten Sie jedoch, dass das Einbetten der Schriftarten zusätzlichen Speicherplatz benötigt. Dokumente mit eingebetteten Schriften sind entsprechend größer.

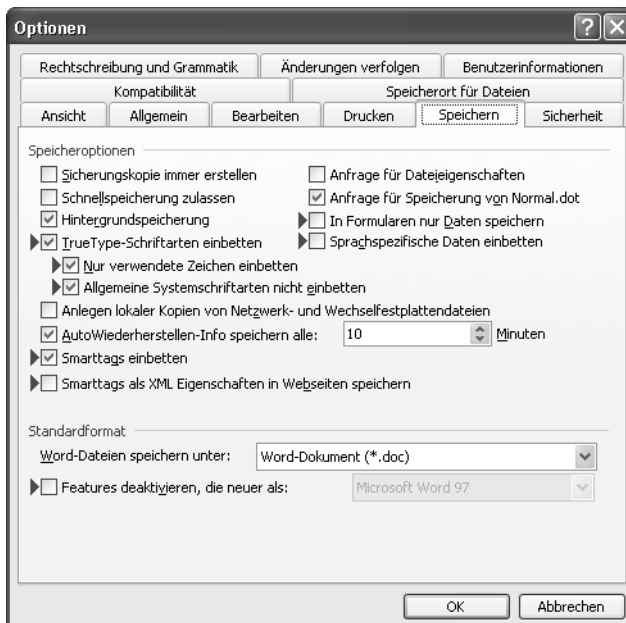
Das Aktivieren des Kontrollkästchens *Sprachspezifische Daten einbetten* scheint von der Bezeichnung her sinnvoll zu sein. Ein direkter Nutzen ist jedoch nicht vorhanden, da die entsprechende Schnittstelle (Erkennung von Spracheingabe und Handschrift) in der deutschen Version von Word nicht implementiert ist.



2007

Die Option für TrueType-Schriftarten befindet sich auf der Registerkarte *Speichern*. Jene für Formulare sowie Smarttags sind auf der Registerkarte *Erweitert* im Abschnitt *Genauigkeit beim Freigeben dieses Dokuments beibehalten* hinterlegt. Um Features auszuschalten, welche in früheren Versionen noch nicht vorhanden waren, muss der standardmäßige Dateityp auf der Registerkarte *Speichern* auf *Word 97-2003-Dokument (\*.doc)* eingestellt werden.

Abbildg. 13.9 Festlegen der nötigen Optionen auf der Registerkarte *Speichern*





TrueType.ttf

## 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.

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. 13.10 Die Eigenschaften der TrueType-Schrift prüfen



Die Datei *setup.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.msp>.

## 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.

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.

Abbildg. 13.11 Festlegen der nötigen Optionen auf der Registerkarte *Sicherheit*



Die dargestellten Einstellungen wurden in Word 2007 aus den Optionen entfernt und stehen erst beim Fertigstellen eines Dokuments unter dem Eintrag *Vorbereiten* im Menü der *Office*-Schaltfläche zur Verfügung.

### 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.

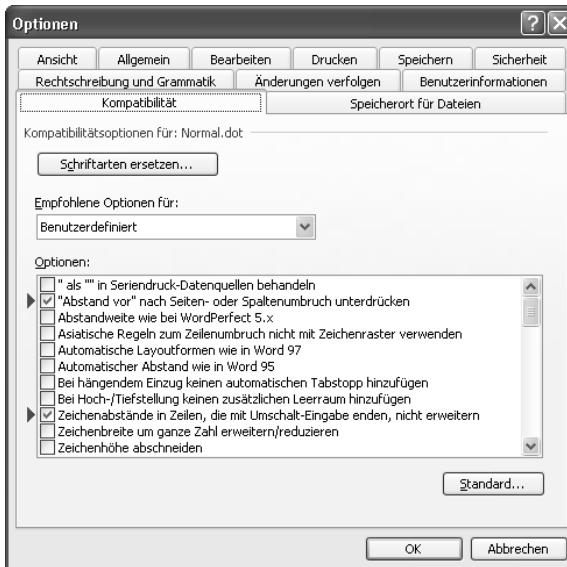


Die Kompatibilitätsoptionen befinden sich zuunterst in der Kategorie *Erweitert* (Schaltfläche Word-Optionen im Menü der *Office*-Schaltfläche).

#### 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. 13.12 Festlegen der notwendigen Optionen auf der Registerkarte *Kompatibilität*



### Registerkarte *Rechtschreibung und Grammatik*

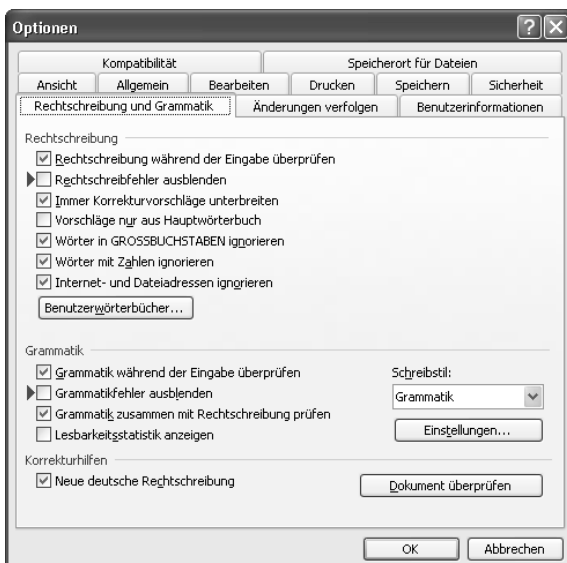
Aus Sicht der Autoren ist es angebracht, wenn dem Anwender die beiden Hilfsmittel als Standard zur Verfügung gestellt werden. Deshalb werden diese beiden Kontrollkästchen nicht aktiviert.



2007

In Word 2007 befinden sich diese Einstellungen in der Kategorie *Dokumentprüfung* (Schaltfläche *Word-Optionen* im Menü der *Office*-Schaltfläche).

Abbildg. 13.13 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 Dialogfeldes).
- 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.



Das Handbuch.ini

## 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 13.4 wird ebenfalls eine *.ini*-Datei aufgebaut. Das Resultat der entsprechenden Programmsequenz ist in Abbildung 13.14 dargestellt.

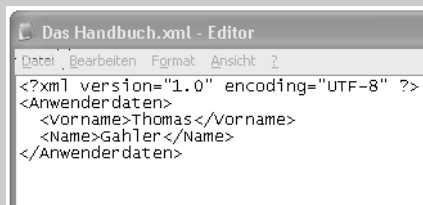
Abbildg. 13.14 Der Aufbau einer *.ini*-Datei



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 dem Programmierer die zentrale Windows-Registrierung 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.

Abbildg. 13.15 Der Aufbau einer XML-Konfigurationsdatei



## 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 13.1** 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 13.2** 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")
```

**Listing 13.2**    Auslesen der Anwenderdaten mittels *GetSetting* aus der Windows-Registrierung (Fortsetzung)

```

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 13.3 bis Listing 13.6 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 3 mit entsprechenden Beispielen aufgezeigt.

**Listing 13.3**    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

    '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 13.4

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
```

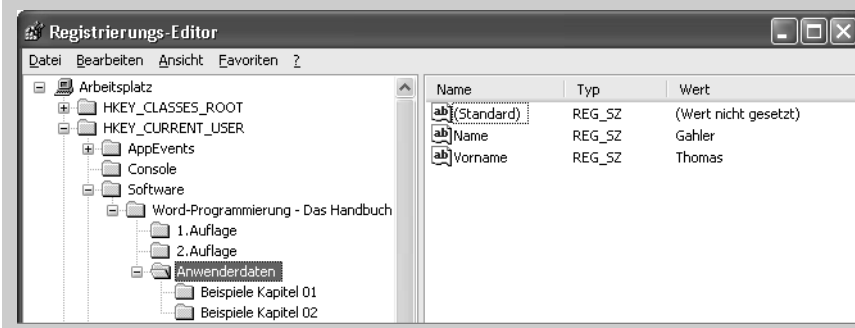
**Daten in der Windows-Registrierung abspeichern**

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.

Abbildg. 13.16 Strukturierte Datenablage in der Windows-Registrierung aufbauen



Mit der `PrivateProfileString`-Eigenschaft können die gespeicherten Werte auf die gleiche Art vom Speicherort ausgelesen werden.

**Listing 13.5** 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 13.6** 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

    '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 & _
```

Listing 13.6 Auslesen der Anwenderdaten mittels *PrivateProfileString* aus einer Konfigurationsdatei (Fortsetzung)

```

    "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 13.7 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 13.7 *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")

    '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

```

**Listing 13.7** *PrivateProfileString*-Eigenschaft als eigene Funktion mit Standardrückgabewert erweitert (Fortsetzung)

```
'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
```

## Informationen mit MSXML schreiben und lesen

Allgemeine Informationen zu XML sind in Kapitel 22 beschrieben. Mit den nachstehenden Programmzeilen soll erläutert werden, wie Anwenderdaten in die XML-Datei der Abbildung 13.15 geschrieben (Listing 13.8) und von dort wieder ausgelesen werden (Listing 13.9).

**Listing 13.8** Daten in eine XML-Konfigurationsdatei mit der MSXML-Bibliothek (XMLDOM) schreiben

```
Sub XmlKonfigSchreiben()
    Const cstrFileName As String = "C:\Das Handbuch.xml"
    Const strFehlerDateiNichtGefunden As String =
        "Die XML-Konfigurationsdatei konnte nicht gefunden werden."
    Dim xmlDoc As MSXML2.DOMDocument
    Dim strVorname As String, strName As String

    strVorname = InputBox("Bitte Vorname eingeben.")
    strName = InputBox("Bitte Familienname eingeben.")

    Set xmlDoc = New MSXML2.DOMDocument
    xmlDoc.async = False
    If Not xmlDoc.Load(cstrFileName) Then
        MsgBox strFehlerDateiNichtGefunden
    Else
        If xmlDoc.parseError <> 0 Then
            MsgBox "Parse error in XML-Datei: " & xmlDoc.parseError.reason
        Else
            xmlDoc.SelectSingleNode("Anwenderdaten/Vorname").Text = strVorname
            xmlDoc.SelectSingleNode("Anwenderdaten/Name").Text = strName
            xmlDoc.Save cstrFileName
        End If
    End If
    Set xmlDoc = Nothing
End Sub
```

Listing 13.9 Daten aus einer XML-Konfigurationsdatei lesen

```

Sub XmlKonfigLesen()
    Const cstrFileName As String = "C:\Das Handbuch.xml"
    Const strFehlerDateiNichtGefunden As String = _
        "Die XML-Konfigurationsdatei konnte nicht gefunden werden."
    Dim xmlDoc As MSXML2.DOMDocument
    Dim strVorname As String, strName As String

    Set xmlDoc = New MSXML2.DOMDocument
    xmlDoc.async = False
    If Not xmlDoc.Load(cstrFileName) Then
        MsgBox strFehlerDateiNichtGefunden
    Else
        If xmlDoc.parseError <> 0 Then
            MsgBox "Parse error in XML-Datei: " & xmlDoc.parseError.reason
        Else
            strVorname = xmlDoc.SelectSingleNode("Anwenderdaten/Vorname").Text
            strName = xmlDoc.SelectSingleNode("Anwenderdaten/Name").Text
            MsgBox "Vorname" & vbTab & strVorname & vbCr & "Name" & vbTab & strName
        End If
    End If
    Set xmlDoc = Nothing
End Sub

```



Das dargestellte Beispiel finden Sie in der Beispieldatei *Bsp13\_01.doc* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap13*.

## 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 einem Schwarzweißdrucker bzw. auf 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.

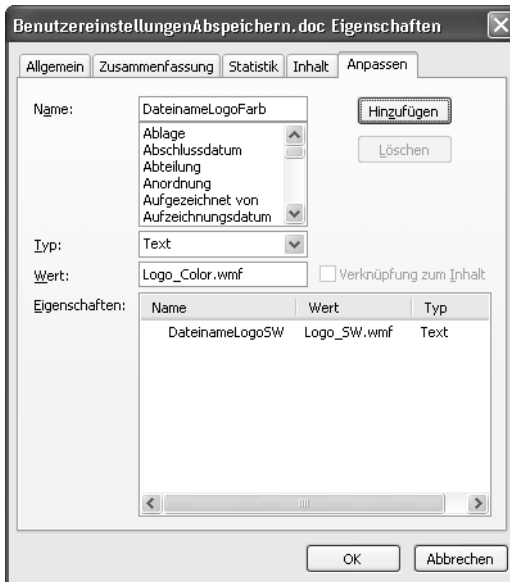
Die erste Möglichkeit ist die Verwendung der *Dokumenteigenschaften*. Diese können sehr einfach auf die persönlichen Bedürfnisse hin angepasst und erweitert werden:

- 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. 13.17 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* sind eine Auflistung. Diese kann mit den Methoden *Add*, *Delete* usw. bearbeitet werden.

Listing 13.10 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.

In Listing 13.10 wurde das Dokument als bearbeitet gekennzeichnet. Dies wurde mit der Zeile `docDemo.Saved = False` erreicht.

Wird wie in Listing 13.10 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 13.11 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 13.11**   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 die Bearbeitung der Eigenschaften von Office-Dokumenten aus Visual Basic .NET 2003 und Visual Basic .NET« 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>.



**HINWEIS** Für .NET-Entwickler

Dokumenteigenschaften sind Mitglieder des Office-Objektmodells und werden, ähnlich den Word-Dialogfeldern und WordBasic-Befehlen, den einzelnen Office-Anwendungen über Late Binding bereitgestellt. Aus diesem Grund kann C# `BuiltinDocumentProperties` und `CustomDocumentProperties` nicht direkt über die Word-Anwendung ansprechen. Ein Beispielprojekt, das wir hier aus Platzgründen nicht vorstellen, liegt auf der CD bei. Zudem finden Sie mehr Informationen im Knowledge-Base-Artikel »So wird's gemacht: Verwenden der Automatisierung zum Abrufen und Festlegen von Office-Dokumenteigenschaften mit Visual C# .NET« auf <http://support.microsoft.com/kb/303296/de>.

## 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 13.12** 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.



Das dargestellte Beispiel finden Sie in der Beispieldatei *Bsp13\_01.doc* auf der CD-ROM zum Buch im Ordner `\Beispiele\Kap13`. Das Beispiel für C#-Entwickler befindet sich in der Datei *Kap13\_CS.zip* im gleichen Ordner.

## Zusammenfassung

In diesem Kapitel wurden die unterschiedlichen Möglichkeiten zum Abspeichern von Anwendungsoptionen aufgezeigt:

- Im ersten Abschnitt des Kapitels wurde dargestellt, weshalb es sinnvoll ist, die allgemeine Dokumentvorlage *Normal.dot* explizit zu konfigurieren und welche Schritte dazu nötig sind (Seite 628 ff.).
- Im Weiteren wurde erklärt, wie Benutzereinstellungen innerhalb einer Konfigurationsdatei oder der Windows-Registrierung abgespeichert und von dort wieder gelesen werden können (Seite 641).
- Im dritten und letzten Abschnitt wurde erläutert, wie Dokumenteinstellungen abgespeichert und wieder gelesen werden können (Seite 649).

## Kapitel 14

# Speicherort der Anpassungen

### In diesem Kapitel:

Der Speicherort einer Anpassung	656
Kontext für COM-Add-Ins	661
Hierarchie der Anpassungen	662
Zusammenfassung	663

Heutzutage stellen alle Office-Anwendungen ähnliche Benutzerschnittstellen zur Verfügung. Dazu gehören beispielsweise Dialogfelder, Symbolleisten (inkl. Menüs) bzw. die Multifunktionsleiste in Office 2007 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 Möglichkeiten.

Bis einschließlich Office 2003 unterstützen alle Module der Office-Familie das Erstellen mehrerer Symbolleisten und deren Bestückung mit Menübefehlen und Makros. In diesen Versionen stellt einzig Word *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 16 vorgestellt.



Ab Office 2007 ersetzt die Multifunktionsleiste (»Ribbon«), zusammen mit der Symbolleiste für den Schnellzugriff (»QAT«), die Symbolleisten. Ein unauffälliger aber dennoch wichtiger Teil der neuen Funktionalität stellt das Command-Element dar, das dem Office-Entwickler alle internen Befehle zur direkten Anwendung bereitstellt. Einzelheiten zu dieser neuen Benutzerschnittstelle lesen Sie in Kapitel 17.

Des Weiteren kann der Word-Entwickler einer VBA-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 Word-Befehlen ist in Kapitel 20 beschrieben.

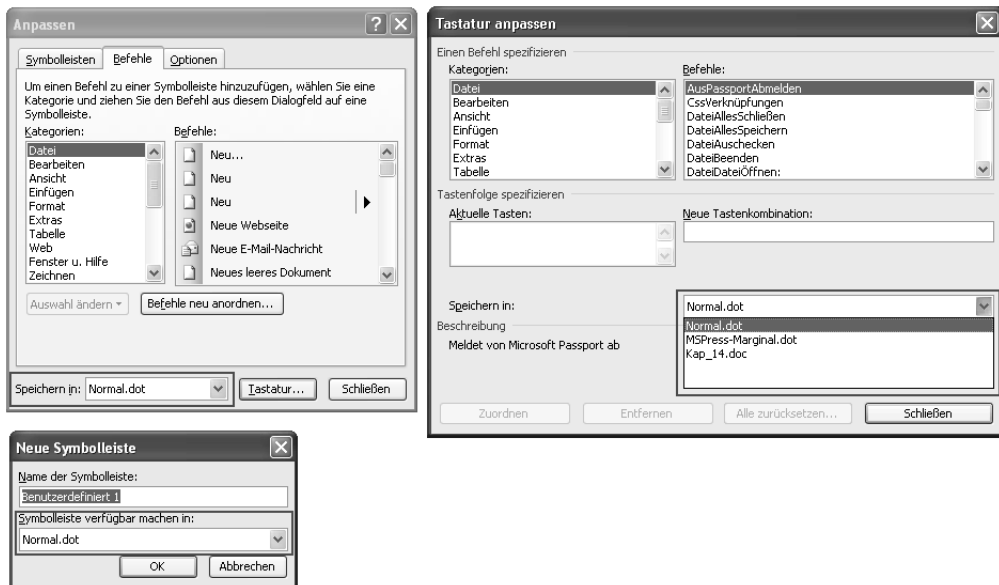
Zusätzlich können die internen Dialogfelder von Word eingeblendet werden. Viele Einstellungen lassen sich vor dem Aktivieren festlegen und anschließend wieder auslesen. Diese Möglichkeiten werden in Kapitel 15 erläutert.

Schließlich kann jeder Menübefehl oder jedes Makro (Public Sub, ohne Argumente) einer beliebigen Tastenkombination zugewiesen werden (Menübefehl *Extras/Anpassen/Tastatur* in Word oder über die KeyBindings-Auflistung des Objektmodells, dies wird in Kapitel 18 detailliert beschrieben).

Dieses Kapitel zeigt, 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. Die Dialogfelder in *Extras/Anpassen* (bzw. in *Word-Optionen/Anpassen* für Word 2007) verfügen über eine *Speichern in*-Dropdownliste (Abbildung 14.1), 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. 14.1 Bei Anpassungen sollte die Auswahl für *Speichern in* immer genau kontrolliert werden

Makros, Symbolleisten und -Schaltflächen, und Tastenkombinationen können sowohl in einem Dokument als auch einer Dokumentvorlage gespeichert werden. AutoText bzw. Baustein-Einträge dagegen lassen sich ausschließlich in Dokumentvorlagen ablegen. Der Speicherort, also die eigentliche Datei, dieser vier Elemente stellt den *Kontext* der Anpassung dar.

## Auswahl des Speicherorts

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

- Im aktuellen Dokument
- In einer beliebigen Dokumentvorlage
- In der Standardvorlage *Normal.dot*
- In einem Add-In

Jeder Einzelne dieser Speicherorte hat seine Vor- und Nachteile, die zur Veranschaulichung in Tabelle 14.1 kurz zusammengefasst sind.

**Tabelle 14.1** Speicherorte für Makros und deren Vor- bzw. Nachteile

Speicherort	Vorteil	Nachteil
Dokument	Die Anpassungen (Makros, Befehlsschaltflächen sowie Tastenkombinationen) 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 »VBA-Projekte mit einer Signatur versehen« in Kapitel 1).
Dokumentvorlage	Die Anpassungen stehen nur für Dokumente zur Verfügung, die auf der betreffenden Dokumentvorlage basieren.	Der Zugriff auf die Dokumentvorlage muss stets gewährleistet sein.
Normal.dot	Die Anpassungen stehen für alle Dokumente, auch jene aus fremder Quelle, zur Verfügung.	Die Anpassungen stehen nur einem einzelnen Anwender zur Verfügung.
Add-In	Die Anpassungen werden über eine geschlossene Programmerweiterung eingesetzt.	Das Aktualisieren der Makros innerhalb einer Firmenumgebung benötigt normalerweise die Unterstützung des Netzwerkadministrators.

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. Tabelle 14.2 bietet eine Übersicht.

**Tabelle 14.2** 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

Grundsätzlich gilt:

- 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.
- Die *Normal.dot* gehört grundsätzlich dem Anwender. Hier haben Sie, als »Fremder«, nichts zu suchen. Mehr zum Thema *Normal.dot* lesen Sie in Kapitel 13. 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.

**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.

## Den Speicherort programmtechnisch festlegen

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 Programmierumgebung 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
```



# Kontext für COM-Add-Ins



Beim Erstellen eines COM-Add-Ins für Word 2003 und früher muss der Kontext für die Symbolleisten und Symbolschaltflächen ebenfalls festgelegt werden.

In Word 2007 ist diese Entscheidung hinfällig, da die angepasste Benutzerschnittstelle als dokument-unabhängiges XML definiert ist. Sie wird beim Laden des Add-Ins dynamisch in die Umgebung eingebunden. Mehr darüber lesen Sie in Kapitel 10 im Abschnitt über VSTO-COM-Add-Ins.

Allgemein empfehlen die Autoren, eine eigens für diesen Zweck bestimmte Dokumentvorlage in die Gesamtlösung einzubinden. Anpassungen werden in dieser Vorlage gespeichert; das COM-Add-In lädt diese Vorlage als Vorlagen-Add-In. Alles, was sich in der Benutzerschnittstelle nicht dynamisch ändern muss, kann in der Vorlage vordefiniert werden, was einige Codezeilen erspart.

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, so könnte argumentiert werden, dass es wäre besser, wenn nur diese oder jene Symbolleisten oder Menüpunkte sichtbar wären. Dies ist 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.

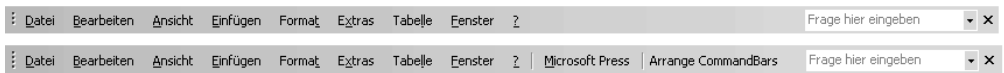
Höchste Priorität haben die Anpassungen im aktuellen Dokument. Wurde beispielsweise die Tastenkombination **Alt+I** in der *Normal.dot* sowie im aktuellen Dokument zwei verschiedenen Befehlen zugewiesen, hat die Zuweisung im aktuellen Dokument Vorrang.

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 14.2 veranschaulicht dieses Prinzip. Oben steht die unveränderte Menüleiste der *Normal.dot*. Zum Vergleich befinden 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. 14.2** 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 ein Menüeintrag in der Vorlage vorhanden, auf einer bestimmten Installation jedoch nicht sichtbar ist, so könnte ein Konflikt mit einer Anpassung in der *Normal.dot* die Ursache sein.

Wurde auf dieser Installation irgendwann eine Symbolschaltfläche mit der gleichen Beschriftung erstellt und danach ausdrücklich gelöscht, so »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 (Dokumentvorlage) 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 (Seite 656).
- Die bewusste Wahl des Speicherorts legt die Sichtbarkeit der Anpassung fest (Seite 657).
- Die Eigenschaft `CustomizationContext` wird gebraucht, um den Speicherort programmtechnisch festzulegen (Seite 659).
- 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 661).
- Sind Anpassungen aus mehreren Kontexten gleichzeitig vorhanden, wird hierarchisch entschieden, welche sichtbar sind (Seite 662).



## Kapitel 15

# Mit Dialogfeldern arbeiten

### In diesem Kapitel:

Benutzerdefinierte Dialogfelder	666
Interne Dialogfelder	690
<i>FileDialog</i> -Objekt	696
Zusammenfassung	706

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 Ihnen zeigen, 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

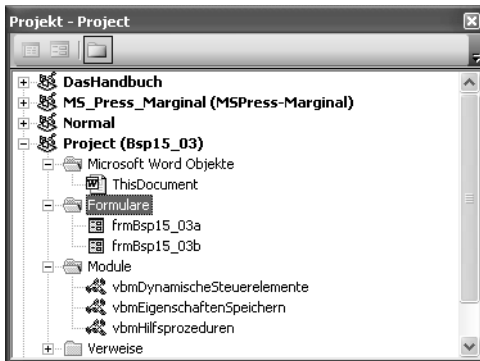


Für benutzerdefinierte Dialogfelder steht die Klasse `UserForm` aus der Bibliothek `MSForms` zur Verfügung. Mit dieser Klasse können Dialogfelder erstellt werden, welche die Schnittstelle zwischen dem eigentlichen Programm und dem Anwender bilden. Alle benötigten Eingabewerte für ein Makro werden in 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 15.1 ersichtlich ist, wie der eigentliche Programmcode selber, zusammen als Projekt in einem Dokument oder einer Dokumentvorlage abgespeichert.

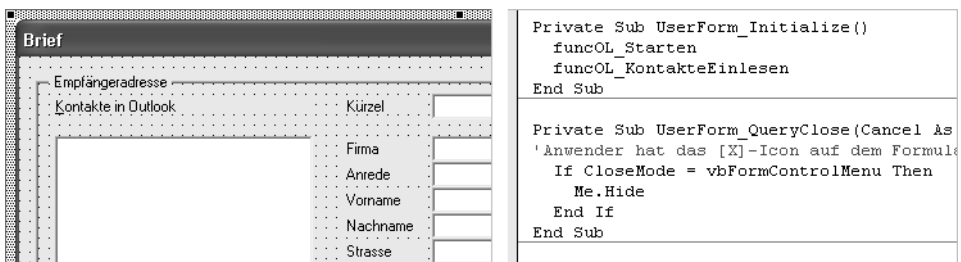
Abbildg. 15.1 Das UserForm-Objekt wird im Zweig *Formulare* zusammen mit dem Projekt gespeichert

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.

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. 15.2 Das UserForm-Objekt im Entwicklungsmodus (links) und das zugehörige Code-Fenster (rechts)



## UserForm in verschiedenen Projekten nutzen

Das Erstellen eines benutzerdefiniertes Dialogfeldes 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 1 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 Programmzeilen 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 10 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 gemeinsam genutzten Objekten gearbeitet wird, wurde ebenfalls in Kapitel 10 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. 15.3** 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 2 erwähnt, sollten 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  
Top  
Height  
Width** 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.

**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  
Posi-  
tion** Mit der StartUpPosition-Eigenschaft kann festgelegt werden, an welcher Position eine neue Instanz des UserForm-Objekts am Bildschirm angezeigt wird.

**Tabelle 15.1** 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 15.1 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  
Unload

Um ein UserForm am Bildschirm darzustellen, muss diese vorab in den Speicher geladen werden. Dieser Schritt kann (wie in Listing 15.1 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 einzublenden (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 15.1**

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 15.1 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 15.2 entsprechend angepasst.

## Listing 15.2 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

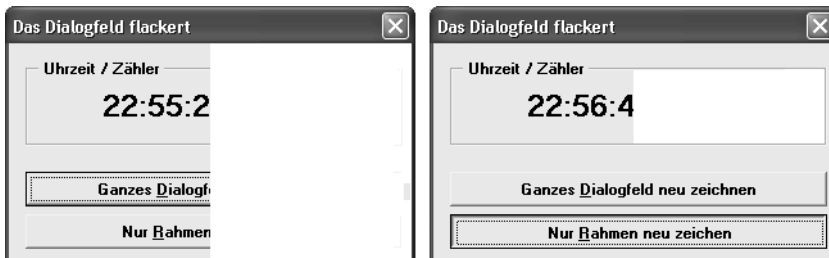
Durch den Aufruf der Repaint-Methode wird das benutzerdefinierte Dialogfeld am Bildschirm neu gezeichnet. Dieser Schritt wird dann benötigt, wenn zur Laufzeit ein am Bildschirm aktives Dialogfeld in der Darstellung modifiziert wird (beispielsweise die Caption-Eigenschaft eines Label-Objekts wird geändert).

**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 Dialogfeldes einschränken.

**Abbildg. 15.4** 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 *Bsp15\_01.doc* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap15*.

## Ereignisse

Die einzelnen Programmschritte, die beim Auftreten eines Ereignisses ausgeführt werden, müssen in jeden Fall manuell beim entsprechenden Ereignis hinterlegt werden.

Initiali-  
ze  
Termini-  
rate

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 Dialogfeldes 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 »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.

Acti-  
vate  
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

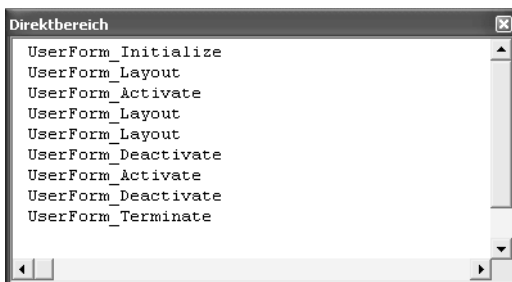
Die einfachste Art, um herauszufinden, in welcher Reihenfolge die einzelnen Ereignisse ausgeführt werden, ist in Listing 15.3 dargestellt. Hier wurde jedem Ereignis, das von Interesse war, eine kleine Programmsequenz hinterlegt. Die Debug.Print-Anweisung protokolliert jedes Auftreten des Ereignisses im Direktbereich der Programmierungsumgebung, wie dies in Abbildung 15.5 ersichtlich ist.

Um den Direktbereich als Fenster innerhalb der Programmierungsumgebung anzuzeigen, rufen Sie den Menübefehl *Ansicht/Direktfenster* auf.

Listing 15.3 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. 15.5 Das Resultat der Protokollierung kann im Direktbereich analysiert werden



**QueryClose** Das QueryClose-Ereignis tritt auf, bevor das Dialogfeld geschossen 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 15.2 zusammengestellt.

**Tabelle 15.2** Zusammenstellung der möglichen Werte des *CloseMode*-Arguments

Bezeichnung	Wert	Bedeutung
vbFormControlMenu	0	Der Benutzer hat auf dem UserForm im Systemmenü den Befehl <i>Schließen</i> gewählt.
vbFormCode	1	Innerhalb des Programms wurde die <b>Unload</b> -Anweisung aufgerufen.
vbAppWindows	2	Die aktuelle Windows-Umgebung wird beendet.
vbAppTaskManager	3	Die Anwendung wird vom Windows Task-Manager geschlossen.

In Listing 15.4 wird als mögliches Beispiel gezeigt, wie verhindert werden kann, dass der Anwender durch Betätigen des in der Titelleiste vorhandenen Schließen-Symbols das benutzerdefinierte Dialogfeld schließt.

**Listing 15.4** 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 *Bsp15\_01.doc* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap15*.

### 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 befindlichen UserForm-Objekte angesprochen werden.

Anhand des Beispiels in Listing 15.5 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 15.5 Entladen aller *UserForm*-Objekte aus dem Arbeitsspeicher.

```

Sub Demo_UserformAlleEntladen()
    Dim intZähler As Integer

    For intZähler = UserForms.Count - 1 To 0 Step -1
        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 15.6 ist ein Fragment aus dem allgemeinen Programm und die Prozedur zur Bearbeitung einer bestimmten Instanz des *UserForm*-Objekts dargestellt.

Listing 15.6 *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 15.6 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 2 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 *Bsp15\_01.doc* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap15*.

## 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-Komponenten zur Verfügung stehen.

### Microsoft Forms 2.0 Object Library

Wie in Abbildung 15.6 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. 15.6 Die Standardsteuerelemente aus der Microsoft Forms 2.0 Object Library



Das *Bezeichnungsfeld* (Label) dient zum Beschriften von einzelnen Steuerelementen, die über keine eigene Bezeichnung (Caption) verfügen, oder zur Ausgabe allgemeiner Informationen auf dem Dialogfeld.



Das *Textfeld* (TextBox) stellt dem Anwender eine Schnittstelle zur Verfügung, welche das Erfassen von freien Texten ermöglicht. Das Textfeld kann als einzelzeiliges Eingabefeld oder als mehrzeiliges Feld definiert werden.



Im *Kombinationsfeld* (ComboBox) kann aus einer vorgegebenen 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 vorgegebenen Menge von Werten ein Eintrag oder mehrere Einträge ausgewählt werden. Es kann definiert werden, ob nur ein Wert oder mehrere Werte 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.



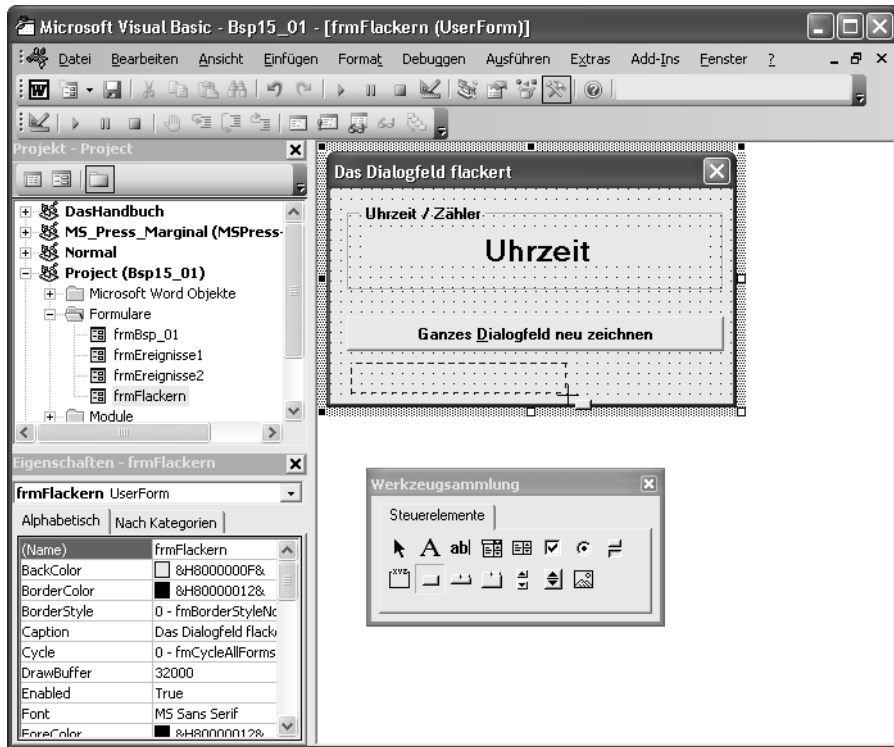
Die *Anzeige* (Image) dient zum Darstellen eines Bildes bzw. einer Grafik auf dem Dialogfeld. Stimmt das Größenverhältnis der Grafik nicht mit jenem des Steuerelements überein, kann die Grafik in der Originalgröße, im gleichen Seitenverhältnis oder auch gestaucht dargestellt werden.

### Steuerelement einfügen

Um ein Steuerelement auf dem UserForm-Objekt zu platzieren, muss dieses (wie in Abbildung 15.7 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. 15.7 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 21 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.

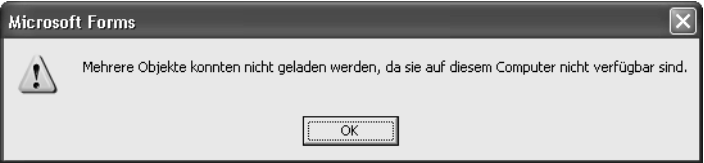
Das eigentliche Hinzufügen von zusätzlichen Steuerelementen wurde bereits in Kapitel 9 besprochen.

### 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 15.8 dargestellt, weist auf dieses Problem hin.

**Abbildg. 15.8** 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

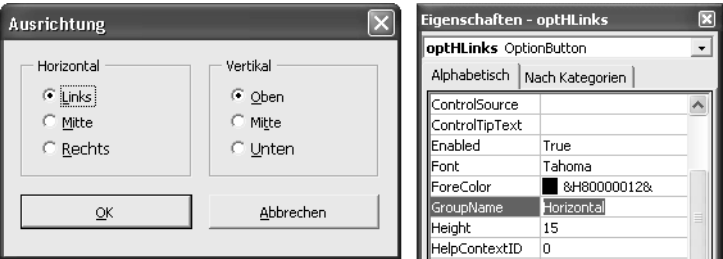
Das Verhalten von einzelnen Steuerelementen aus der »Microsoft Forms 2.0 Object Library« unterscheidet sich teilweise vom bekannten Verhalten aus anderen Entwicklungsumgebungen. Aus diesem Grunde werden eine paar dieser Besonderheiten hier erläutert.

## Option (OptionButton)

Auf einem UserForm-Objekt können mehrere *OptionButtons* eingefügt werden. Zusammengehörende Optionen werden, wie in Abbildung 15.9 ersichtlich, oftmals in einen Rahmen gesetzt, damit dieser Umstand für den Anwender sofort ersichtlich ist.

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. 15.9** Zusammengehörende Optionen werden in Rahmen zusammengefasst



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 15.7 dargestellt, die Auswertung der gewählten Option einzeln erfolgen und kann beispielsweise mit einer *Select Case*-Anweisung erfolgen.

**Listing 15.7** 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 15.10 die verschiedenen Eigenschaften und deren Abhängigkeiten dargestellt. Grundsätzlich stehen drei Eigenschaften zur Verfügung um den gewählten Eintrag im Listenfeld zu ermitteln.

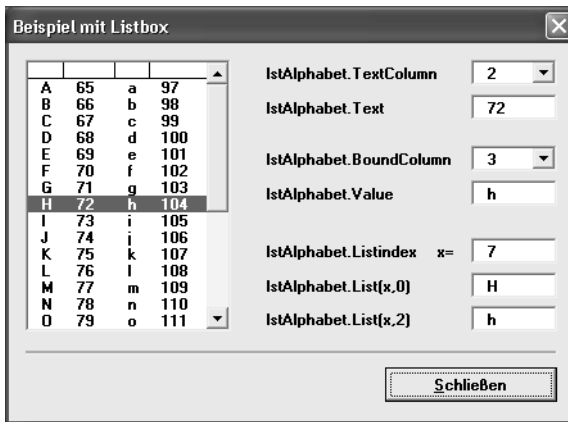
**Text** In der Text-Eigenschaft wird der Wert aus der in der TextColumn-Eigenschaft eingetragenen Spalte ausgegeben. Die Nummerierung der Spalten beginnt bei 1. Wird in der TextColumn-Eigenschaft der Wert 0 eingetragen, werden keine Werte ausgegeben, sondern die Position innerhalb der Liste (ListIndex). Wird ein Wert von -1 eingetragen, so wird der Wert aus der ersten sichtbaren Spalte (ColumnWidth größer 0) ausgegeben.

**Value Bound-Column** 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. 15.10** 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 vergessen zu implementieren (vgl. Abbildung 15.10).



Die dargestellten Beispiele finden Sie in der Beispieldatei *Bsp15\_02.doc* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap15*.

## 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 **[Tab]**-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 (**[Alt]**+Buchstabe) direkt erreicht werden können (Accelerator-Eigenschaft).
- Die Schaltfläche **OK** kann durch Betätigen der **[Enter]**-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 Tastaturkombinationen zugewiesen. Diese sollten möglichst mit jenen aus den internen Dialogfeldern korrespondieren.

Abbildg. 15.11

Komplexes benutzerdefiniertes Dialogfeld, das die Anforderungen an ein Dialogfeld abdeckt

**Drucken mit Papierschlachtsteuerung**

**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 erfolgtem 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 Eigenschaftenfenster den einzelnen Eigenschaften alle benötigten Standardwerte zu.
- Fügen Sie ein erstes Steuerelement dem Dialogfeld hinzu. Weisen Sie dem Steuerelement im entsprechenden Eigenschaftenfenster 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 13 wurde erläutert, 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 15.8 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 15.12).

**Listing 15.8** Abspeichern der Dialogfeld-Eigenschaften während der Terminierung des *UserForm*-Objekts

```
Const REG_APP As String = "Word-Programmierung - Das Handbuch"
Const REG_SEC As String = "Bsp15_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

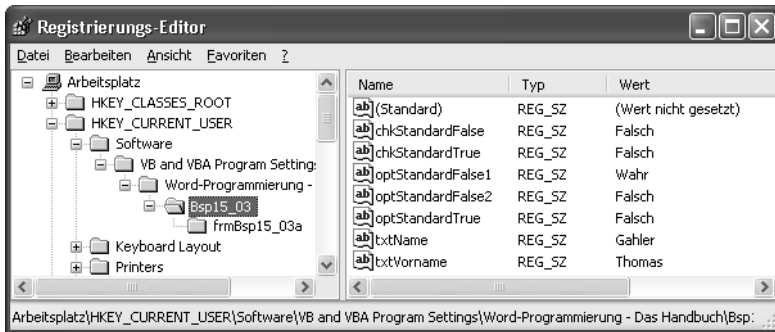
SaveSetting REG_APP, REG_SEC, "chkStandardTrue", chkStandardTrue.Value
SaveSetting REG_APP, REG_SEC, "chkStandardFalse", chkStandardFalse.Value

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. 15.12** 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. Frühestens zu diesem Zeitpunkt können die gewünschten Werte eingetragen werden.

In Listing 15.9 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 15.9** Einlesen der Dialogfeld-Eigenschaften während der Initialisierung des *UserForm*-Objekts

```
Const REG_APP As String = "Word-Programmierung - Das Handbuch"
Const REG_SEC As String = "Bsp15_03"

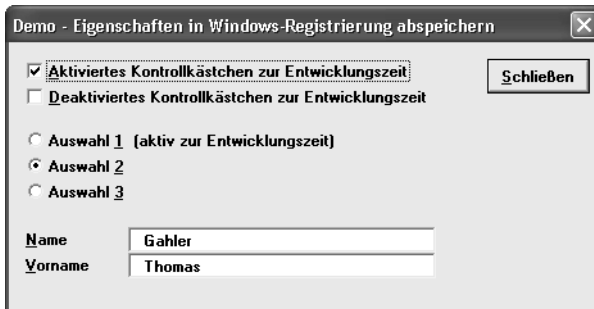
Private Sub UserForm_Initialize()
    'Werte aus Windows-Registrierung auslesen
    Me.Top = CInt(GetSetting(REG_APP, REG_SEC & "\" & Me.Name, "Top", 100))
    Me.Left = CInt(GetSetting(REG_APP, REG_SEC & "\" & Me.Name, "Links", 100))

    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. 15.13 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 ähnlichem 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*-, *.txt* oder *.xml*-Dateien. In Kapitel 13 wurde bereits erläutert, wie einzelne Werte in Konfigurationsdateien abgespeichert und von dort wieder eingelesen werden können.

### Eigenschaften in *.ini*-Dateien hinterlegen

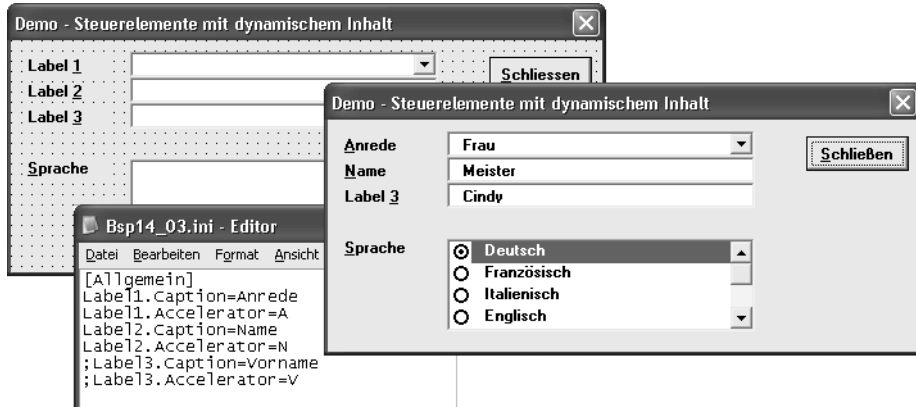
In Listing 15.10 ist eine Programmsequenz aus dem Initialize-Ereignis des UserForm-Objekts aufgeführt. Hier werden die Bezeichnungen der Steuerelemente aus einer *.ini*-Datei eingelesen und auf das Dialogfeld übertragen.

Listing 15.10 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 13 vorgestellt.

Abbildg. 15.14 Benutzerdefiniertes Dialogfeld zur Entwicklungszeit und zur Laufzeit mit zugehöriger `.ini`-Datei



**HINWEIS** In Abbildung 15.14 ist ersichtlich, wie in der `.ini`-Datei `Bsp15_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.

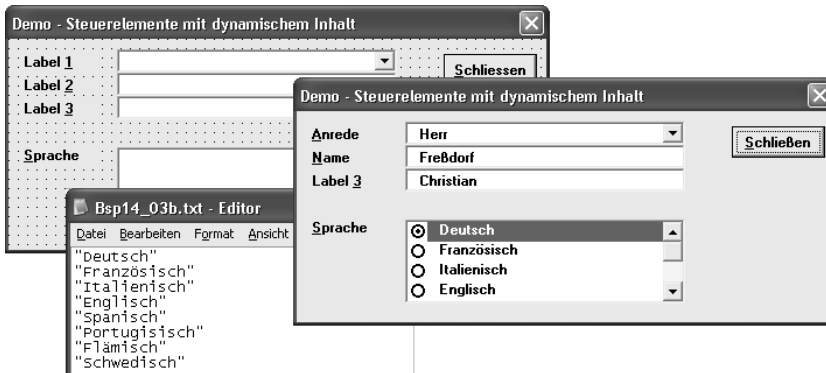
### Eigenschaften in `.txt`-Dateien auslagern

In Listing 15.11 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 15.11 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. 15.15 Benutzerdefiniertes Dialogfeld zur Entwicklungszeit und zur Laufzeit mit zugehöriger .txt-Datei



### Daten in .xml-Dateien auslagern und einlesen

Analog zum Speichern und Lesen von Dokumenteigenschaften kann auch der Inhalt eines Dialogfelds für die Weiterverwendung extern gespeichert werden. Das Beispiel in Listing 15.12 und Listing 15.13 zeigt, wie dies mit einer XML-Datei gemacht wird.

Statt die Prozeduren im Klassenmodul des Formulars zu speichern, wird diese durch das Click-Ereignis der Schaltfläche *Schließen* aufgerufen. Dabei wird veranschaulicht, wie das Formular-Objekt als Argument an eine Prozedur weitergegeben wird.

Wenn es gilt, den Wert einer Liste festzuhalten, stellt sich die Frage, ob der Index- oder der Textwert von Interesse ist. Für den Mensch ist der Textwert aussagekräftiger, für das Dialogfeld ist der Indexwert maßgebend. Das Beispiel speichert beide Werte, wobei der Indexwert in ein Attribut eingetragen wird.

Listing 15.12 Daten in eine xml-Datei speichern

```
Sub DatenSchreiben(frm As frmBsp15_03b)
    Dim xmlDoc As MSXML2.DOMDocument
    Const FILE_XML_B As String = "Bsp15_03.xml"

    Set xmlDoc = New MSXML2.DOMDocument
    xmlDoc.async = False
    xmlDoc.Load(ThisDocument.Path & "\" & FILE_XML_B)
    If xmlDoc Is Nothing Then
        MsgBox "Die XML-Datei konnte nicht gefunden werden."
    End If
    If xmlDoc.parseError <> 0 Then
        MsgBox "Fehler beim Laden der XML-Datei: " & xmlDoc.parseError.reason
    Else
        xmlDoc.SelectSingleNode("//Anrede").Text = frm.cboAnrede.Text
        xmlDoc.SelectSingleNode("//Anrede/@index").Text = frm.cboAnrede.ListIndex
        xmlDoc.SelectSingleNode("//Vorname").Text = frm.txtVorname.Text
        xmlDoc.SelectSingleNode("//Name").Text = frm.txtName
        xmlDoc.SelectSingleNode("//Sprache").Text = frm.lstSprache.Text
        xmlDoc.SelectSingleNode("//Sprache/@index").Text = frm.lstSprache.ListIndex
    End If
    xmlDoc.Save ThisDocument.Path & "\" & FILE_XML_B
    Set xmlDoc = Nothing
End Sub
```

**Listing 15.13** Daten aus einer *xml*-Datei lesen

```

Sub DatenLesen(frm As frmBsp15_03b)
    Dim xmlDoc As MSXML2.DOMDocument
    Const FILE_XML_B As String = "Bsp15_03.xml"

    Set xmlDoc = New MSXML2.DOMDocument
    xmlDoc.async = False
    xmlDoc.Load (ThisDocument.Path & "\" & FILE_XML_B)
    If xmlDoc Is Nothing Then
        MsgBox "Die XML-Datei konnte nicht gefunden werden."
    End If
    If xmlDoc.parseError <> 0 Then
        MsgBox "Fehler beim Laden der XML-Datei: " & xmlDoc.parseError.reason
    Else
        frm.cboAnrede.ListIndex = xmlDoc.SelectSingleNode("//Anrede/@index").Text
        frm.txtVorname.Text = xmlDoc.SelectSingleNode("//Vorname").Text
        frm.txtName = xmlDoc.SelectSingleNode("//Name").Text
        frm.lstSprache.ListIndex = xmlDoc.SelectSingleNode("//Sprache/@index").Text
    End If
    Set xmlDoc = Nothing
End Sub

```



Das dargestellte Beispiel finden Sie in der Beispieldatei *Bsp15\_03.doc* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap15*.

## Interne Dialogfelder

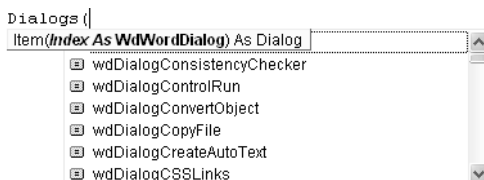
Bei der täglichen Arbeit mit Word begegnen Ihnen an verschiedenen Stellen Dialogfelder, in denen Sie Einstellungen oder Änderungen vornehmen oder Informationen auslesen können. Bei diesen Dialogfeldern handelt es sich um Word-interne (integrierte) Dialogfelder, die sich meist hinter Menübefehlen verbergen 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 rund 230 und in Word 2007 sogar fast 300 Dialogfelder zur Verfügung.

Alle benannten Konstanten werden Ihnen im Visual Basic-Editor nach Eingabe der ersten Klammer »(« in einer Auswahlliste angezeigt.

**Abbildg. 15.16** 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 15.3 aufgeführten Rückgabewerte.

**Tabelle 15.3** 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 OK 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.

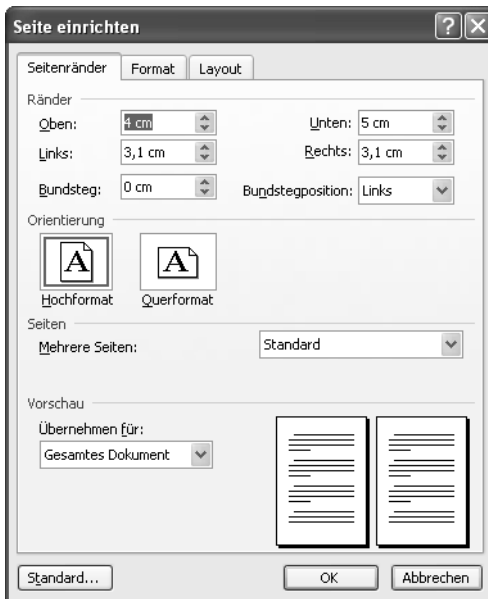
Update

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. 15.17** Anzeige des Dialogfeldes zur Seiteneinrichtung *wdDialogFileDocumentLayout*





Wenn die Dokumentseite wie in Abbildung 15.17 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 15.14** 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 ihrer 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`.

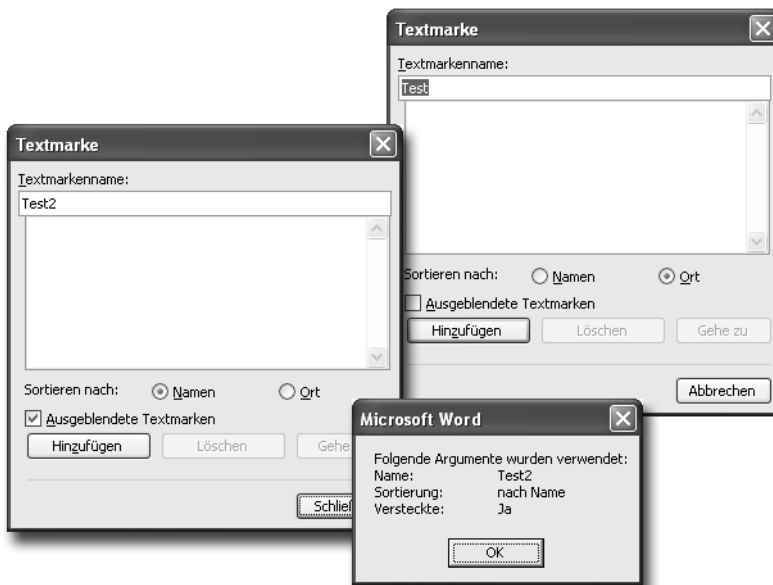
In Listing 15.15 wird das Dialogfeld zum Einfügen von Textmarken so eingestellt, dass ein Name für die Textmarke vorgegeben, die Anzeige versteckter Textmarken ausgeschaltet und die Sortierung nach Ort eingeschaltet wird. Da Sie diese Einstellungen aber im Dialogfeld ändern können, müssen Sie anschließend 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 15.15** Konfigurieren des Dialogfeldes *wdDialogInsertBookmark* mittels Argumenten 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. 15.18** 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 Dialogfeldern 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 15.16 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 15.16** 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 *Bsp15\_04.doc* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap15* 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\Kap15\wdDialogsÜbersicht\_Word2003.doc*. Eine Übersicht über die *wdWordDialog*-Konstanten finden Sie in der Datei *\Beispiele\Kap15\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 *Bsp15\_04.doc* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap15*.

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:

- `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 15.4 sind die verfügbaren Eigenschaften aufgeführt.

Tabelle 15.4 Übersicht über die Eigenschaften der *FileDialog*-Objekte

Eigenschaft	Parameter	Zugriff	Beschreibung
AllowMultiSelect	True/False	Lesen Schreiben	Legt fest, ob mehrere Dateien ausgewählt werden können
Application		Lesen	Gibt die Containeranwendung, in der das <b>FileDialog</b> -Objekt aufgerufen wurde, zurück
ButtonName	»Bezeichnung«	Lesen Schreiben	Legt den Beschriftungstext der auslösenden Aktionsschaltfläche fest oder gibt ihn zurück
Creator		Lesen	Eine 32-Bit-Zahl, die die Containeranwendung repräsentiert
DialogType	<code>msoFileDialogOpen</code> <code>msoFileDialogSaveAs</code> <code>msoFileDialogFilePicker</code> <code>msoFileDialogFolderPicker</code>	Lesen Schreiben	Der durch die <code>MsoFileDialogType</code> -Konstante festgelegte <b>FileDialog</b> -Typ
Filters	<code>FileDialogFilters</code> -Objekt		Gibt eine <code>FileDialogFilters</code> -Auflistung zurück, über die Sie Dateiauswahlfiler definieren können (siehe den Abschnitt »Definieren von Dateiauswahlfiler« in diesem Kapitel)

Optimierung der  
Benutzerschnittstelle

**Tabelle 15.4** Übersicht über die Eigenschaften der *FileDialog*-Objekte (Fortsetzung)

Eigenschaft	Parameter	Zugriff	Beschreibung
FilterIndex	»Index«	Lesen Schreiben	Legt den zu verwendenden Datei- filter 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 15.5 aufgeführten Rückgabewerte.

Tabelle 15.5 Übersicht über die Rückgabewerte der *FileDialog*-Dialogfelder

Rückgabewert	Beschreibung
-1	Das Dialogfeld wurde über die Aktionsschaltfläche beendet. Bei Verwendung der <b>Show</b> -Methode wird die Aktion dabei nicht ausgeführt.
0	Das Dialogfeld wurde über die Schaltfläche Abbrechen, über das Schließen-Symbol in der Titelleiste oder über die Tastenkombination <b>Alt</b> + <b>F4</b> beendet.

## Definieren von Dateiauswahlfilter

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.

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 15.17 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 & vbTab & "(" & dlgFlt.Extensions & ")" & _  
            vbCrLf  
    Next dlgFlt  
    MsgBox strMSG, vbInformation, "Übersicht über die verfügbaren Dateiauswahlfilter " & _  
        "des 'FileDialogOpen'-Dialogfeldes"  
End Sub
```

Abbildg. 15.19 Übersicht über die Dateiauswahlfilter des *FileDialogOpen*-Objektes



Optimierung der  
Benutzerschnittstelle

**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 15.18 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 15.18** 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
    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 15.19 wird die Mehrfachauswahl durch die Angabe der Eigenschaft erlaubt:

Allow-  
Multi-  
Select

```
.AllowMultiSelect = True
```

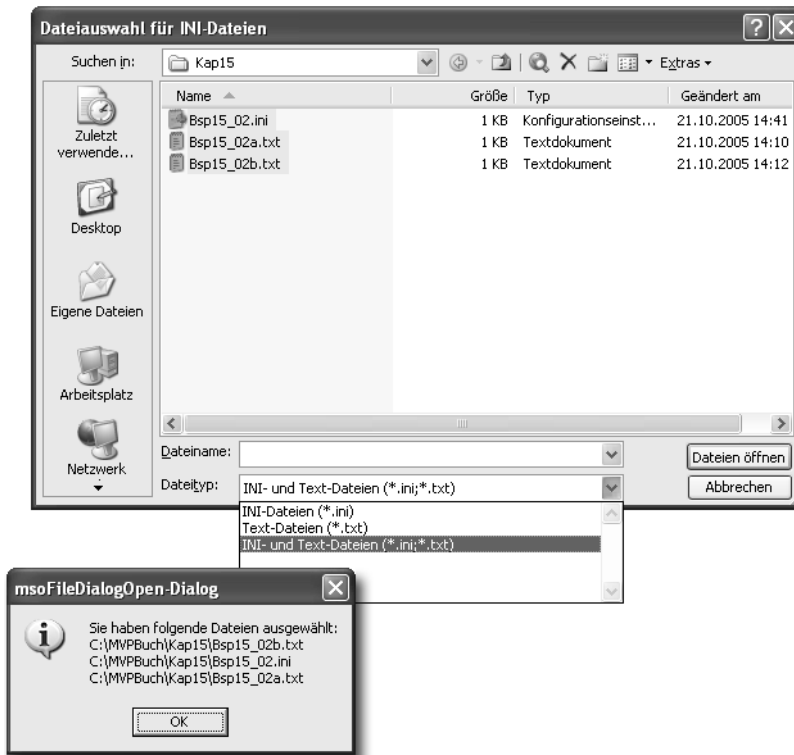
Als nächster Punkt müssen Sie die notwendigen Filter definieren, wenn Sie nicht auf die Standardfilter zurückgreifen möchten. Das Beispiel verwendet drei Filter für die Auswahl von INI-Dateien, Text-Dateien und zur Auswahl beider Dateitypen in einer Auswahl. Alle weiteren Filter werden vorher durch die Funktion *fktStoreFilters* in einem Array gespeichert und anschließend über die Funktion *fktRestoreFilters* 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 15.19** 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)
fktStoreFilters dlg
With dlg
    .ButtonName = "Dateien öffnen"
    .Title = "Dateiauswahl für INI-Dateien"
    .InitialView = msoFileDialogViewDetails
    .AllowMultiSelect = True
    .InitialFileName = "C:\Beispiele\Kap15\"
    .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. 15.20** 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 15.20 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 3) 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 Dateierdung ».doc« im Dateinamen vorkommt. Ist dies der Fall, wird diese durch die Endung ».xml« ersetzt, andernfalls wird die Endung an den Dateinamen angehängt.

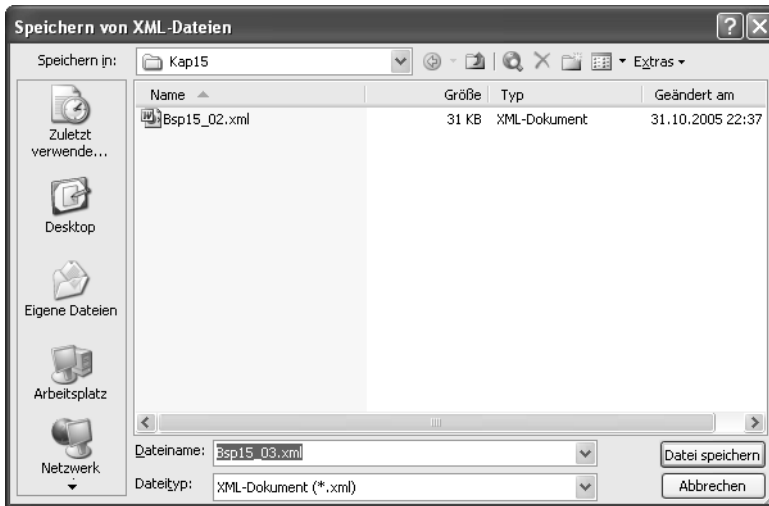
Im nächsten Schritt werden die vorhandenen Dateifilter dahingehend überprüft, ob ein geeigneter Filter für XML-Dateien vorhanden ist. Ist dies der Fall, wird er als verwendeter Dateityp festgelegt, andernfalls wird der Speicher-Vorgang beendet.

Wenn ein entsprechender Dateifilter existiert, wird das Dialogfeld mit den Einstellungen angezeigt. Wird eine Datei angegeben, was auch in einem anderen als dem vorgegebene 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 15.20** 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")
    .InitialFileName = "C:\Beispiele\Kap15\" & 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. 15.21 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 Ordernamen 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 15.21 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.

Dazu durchläuft die Funktion `fktReadDirFiles` mit Hilfe der `Dir`-Funktion (siehe Kapitel 2) alle Dateien des ausgewählten Ordners, ermittelt die Dateiattribute und speichert die Daten in einem

Dir-  
Funktion

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 15.21** 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 & "\")
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

```

**Listing 15.21** Auswahl eines Ordners und Anzeige aller im Ordner enthaltenen Dateien mit Attributen (Fortsetzung)

```

        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. 15.22** Anzeige aller Dateien im ausgewählten Ordner mit ihren Dateiattributen



Die dargestellten Beispiele finden Sie in der Beispieldatei *Bsp15\_04.doc* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap15*.

## Zusammenfassung

In diesem Kapitel wurde der Umgang mit Dialogfeldern erläutert. Dazu gehören nebst den benutzerdefinierten Dialogfeldern (UserForm-Objekt) auch die internen Dialogfelder von Word:

- Im Kapitel wurde gezeigt, wie ein benutzerdefiniertes Dialogfeld angelegt und in verschiedenen Projekten genutzt werden kann (Seite 667 ff.).
- Es wurden die wichtigsten Eigenschaften (Seite 668 ff.), Methoden (Seite 670 ff.) und Ereignisse (Seite 672 ff.) des UserForm-Objekts vorgestellt.
- Es wurden die allgemeinen Steuerelemente aus der »Microsoft Forms 2.0 Object Library« vorgestellt (Seite 676 ff.) und erklärt, wie zusätzliche Steuerelemente eingebunden werden können (Seite 678 ff.).
- Eine Zusammenfassung, welche Anforderungen der Anwender an ein Dialogfeld hat (Seite 679 ff.) und wie ein benutzerdefiniertes Dialogfeld zur Laufzeit beeinflusst werden kann (Seite 684 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 690 ff.) und über die zugehörigen Argumente konfigurieren können (Seite 693). Dabei wurde hervorgehoben, welche Unterschiede in den verschiedenen Aufrufmethoden bestehen.
- Der letzte Abschnitt stellte die verschiedenen FileDialog-Dialogfelder vor (Seite 697). Anhand von Beispielen wurden die Einstellmöglichkeiten und Einschränkungen aufgezeigt (Seite 701 ff.).

## Kapitel 16

# Menüs und Symbolleisten

### In diesem Kapitel:

CommandBars	708
Symbolschaltflächen	713
Symbolleisten erstellen	719
Zusammenfassung	721

In allen Microsoft Office-Versionen bis einschließlich der Version 2003 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 eigentlich Symbolleisten mit Symbolschaltflächen.

Word unterstützt, wie in Kapitel 14 beschrieben, die Erstellung von Symbolleisten für den globalen Gebrauch, oder für Dokument-spezifische 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 eine 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 von Grund auf programmtechnisch erstellt wird.



Ab Office 2007 werden die Menüs und Symbolleisten durch die Multifunktionsleiste abgelöst. Wie diese bearbeitet wird, ist in Kapitel 17 beschrieben. Alle Symbolleisten, die in Dokumenten oder Dokumentvorlagen hinterlegt sind, sowie die dynamisch erstellten, werden in Word 2007 auf der Registerkarte *Add-Ins* angezeigt. Automatisierungscode, der nicht unterstützte Befehle und Funktionen aufruft (Listing 16.2 beispielsweise), läuft ergebnislos und ohne Fehlermeldungen ab. Das in diesem Kapitel beschriebene *CommandBars*-Objektmodell ist einzig für das Erstellen und Verwalten von Kontextmenüs in Word 2007 von Interesse (siehe den Abschnitt »Symbolleisten erstellen« sowie das Listing 16.3 in diesem Kapitel).

Aus Zeitgründen wird es wohl nicht möglich sein, alle in früheren Versionen erstellten Werkzeuge sofort an die neue Benutzerschnittstelle anzupassen. Sie können Befehle aus der Registerkarte *Add-Ins* besser sichtbar machen, indem diese der *Symbolleiste für den Schnellzugriff* zugewiesen werden:

1. Öffnen Sie die bestehende Dokumentvorlage und speichern Sie diese im neuen Dateiformat ab (.dotm).
2. In den *Word-Optionen* aktivieren Sie die Kategorie *Anpassen*.
3. Wählen Sie in der Liste *Symbolleiste für den Schnellzugriff anpassen* den Speicherort aus.
4. Wählen Sie in der Liste *Befehle auswählen* die Registerkarte *Add-Ins* aus.
5. Wählen Sie den Eintrag *Benutzerdefinierte Symbolleisten* aus, klicken Sie auf *Hinzufügen* und bestätigen Sie das Dialogfeld *Word-Optionen* mit OK.

## 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 16.1 erstellt.

#### HINWEIS

Eine entsprechende Liste für die standardmäßige Dokumentvorlage *Normal.dot* finden Sie im Anhang C.

#### Listing 16.1

Eine Tabelle der verfügbaren Symbolleisten generieren, die die englischen und deutschen Bezeichnungen gegenüberstellt

```
Sub SymbolleistenAuflisten()
    Dim cb As Office.CommandBar
    Dim s As String
    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 *Bsp16\_01.doc* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap16*.

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.

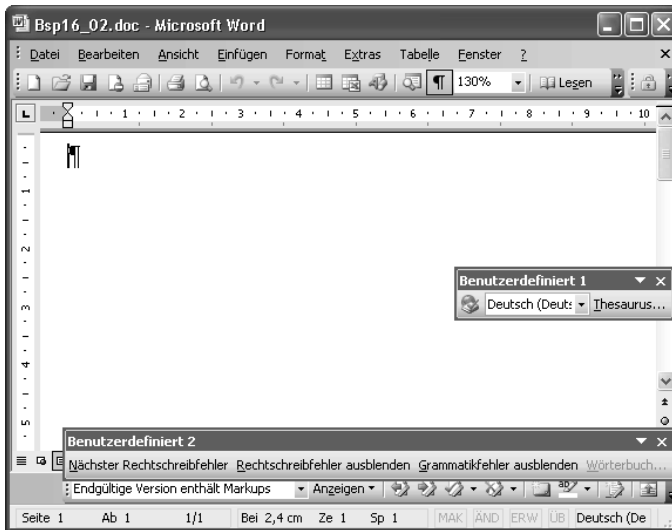
Positionierung

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*

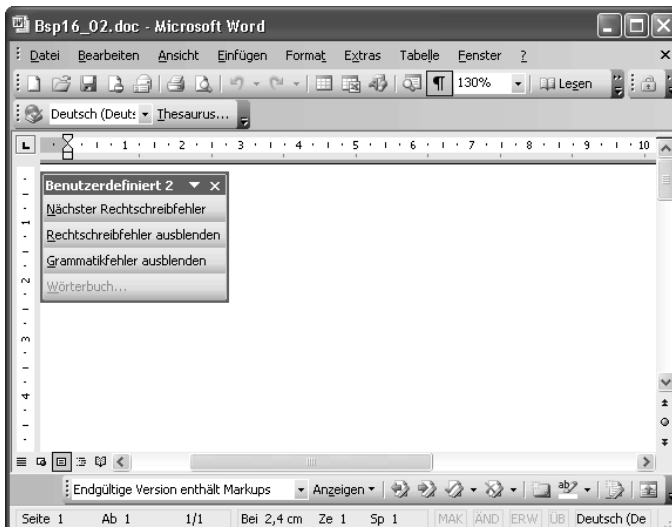
(Höhe). Zusätzlich wird für Symbolleisten, die andockbar werden, mit `RowIndex` die jeweilige Zeile festgelegt, wie die Abbildung 16.1, die Abbildung 16.2 sowie das Listing 16.2 verdeutlichen.

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 andockbaren Symbolleisten (`RowIndex`) andockt. Die zweite Symbolleiste bleibt freistehend (`msoBarFloating`), über der linken oberen Ecke des Dokuments. Ihre Symbolschaltflächen werden, durch Festlegung der Breite, untereinander aufgereiht.

**Abbildg. 16.1** Beliebige Ausgangslage: die Symbolleisten befinden sich »irgendwo«



**Abbildg. 16.2** 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 16.2** 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 16.1 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 16.1** 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 eingeblendet 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.

**Disable Customize** 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* ausgeblendet 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 *Bsp16\_02.doc* finden Sie auf der CD-ROM zum Buch im Ordner `\Beispiele\Kap16`.

Adap- tive- Menus Adap- tiveMenu	Die Eigenschaft <code>AdaptiveMenus</code> gehört der Auflistung <code>CommandBars</code> des <code>Application</code> -Objekts und entspricht der Option <i>Menüs immer vollständig anzeigen</i> unter <i>Extras/Anpassen/Optionen</i> . Ist die Einstellung <code>True</code> , werden nur die zuletzt oder am häufigsten benutzten Befehle in den Symbolleisten angezeigt.
Priority	Die Eigenschaft <code>AdaptiveMenu</code> bezieht sich auf eine einzelne Symbolleiste ( <code>CommandBar</code> ) und legt dieses Verhalten dafür fest. Die Eigenschaft kommt jedoch nur zur Geltung, wenn <code>AdaptiveMenus</code> für die ganze Umgebung <code>True</code> ist.
Disable AskaQue- stionDr- opdown	Damit verwandt ist die Symbolschaltflächen- ( <code>Control</code> ) Eigenschaft <code>Priority</code> . 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 <i>alle</i> Symbolschaltflächen einer Symbolleiste zwingend eingeblendet werden können, auch wenn dadurch eine angedockte Symbolleiste über mehrere Zeilen umbrochen wird.
	Auch interessant ist die Eigenschaft <code>DisableAskAQuestionDropDown</code> der <code>CommandBars</code> -Auflistung. Wird ihr der Wert <code>True</code> zugewiesen, verschwindet das Eingabefeld <i>Frage hier eingeben</i> aus der Menüleiste. Im Gegensatz zu <code>DisableCustomize</code> 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 vom Typ `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.)

Während die internen Symbolleisten von Word über die englische Bezeichnung angesprochen werden müssen, ist der beschreibende Indexwert einer Symbolschaltfläche 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, dieses 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 hat ihrerseits 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.

FindControl  
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 16.3) in die Symbolleiste oder das 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`).

Abbildg. 16.3 Symbolleisten in der Benutzeroberfläche definieren



**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 ein Popup-Menü gezogen.

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://msdn2.microsoft.com/en-us/library/Aa165301\(office.10\).aspx](http://msdn2.microsoft.com/en-us/library/Aa165301(office.10).aspx).

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 16.3 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 16.3

Eine Schaltfläche im Kontextmenü erstellen, die mit einem Makro verbunden ist

```
Sub SymbolschaltflächeDefinieren1()
    Dim ctl As Office.CommandBarButton

    Application.CustomizationContext = ActiveDocument
```

**Listing 16.3** Eine Schaltfläche im Kontextmenü erstellen, die mit einem Makro verbunden ist (Fortsetzung)

```
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 16.4 werden mehrere Symbolschaltflächen einer Symbolleiste hinzugefügt. Als Parameter wird ein Farbname festgelegt. Der `OnAction`-Eigenschaft aller Objekte wird das gleiche Makro `MarkierungHervorheben` zugewiesen. Klickt der Anwender auf eine dieser vier Symbolschaltflächen (in Abbildung 16.4 abgebildet), wird diese durch `ActionControl` ermittelt. Je nach Parameter (dem Farbennamen), wird der Wert für die Hervorhebungsfarbe festgelegt.

**Listing 16.4** Mehrere Schaltflächen erstellen, die mit dem gleichen Makro verbunden sind. Die *Parameter*-Eigenschaft entscheidet, welche Handlung auszuführen ist.

```
Sub SymbolschaltflächeDefinieren2()
    Dim cb As Office.CommandBar

    Application.CustomizationContext = ActiveDocument
    Set cb = CommandBars("Meine Befehle")
    HervorhebungsSchaltflächen cb, "Gelb"
    HervorhebungsSchaltflächen cb, "Grün"
    HervorhebungsSchaltflächen cb, "Rosa"
    HervorhebungsSchaltflächen cb, "Keine"
End Sub

Sub HervorhebungsSchaltflächen(ByVal cb As Office.CommandBar, ByRef sFarbe As String)
    Dim ctl As Office.CommandBarButton

    Set ctl = cb.Controls.Add(Type:=msoControlButton, Parameter:=sFarbe)
    ctl.Caption = sFarbe & " hervorheben"
    ctl.OnAction = "MarkierungHervorheben"
    ctl.DescriptionText = "Hervorhebung der gegenwärtigen Markierung " & sFarbe
```



**Listing 16.4** Mehrere Schaltflächen erstellen, die mit dem gleichen Makro verbunden sind. Die *Parameter*-Eigenschaft entscheidet, welche Handlung auszuführen ist. (Fortsetzung)

```

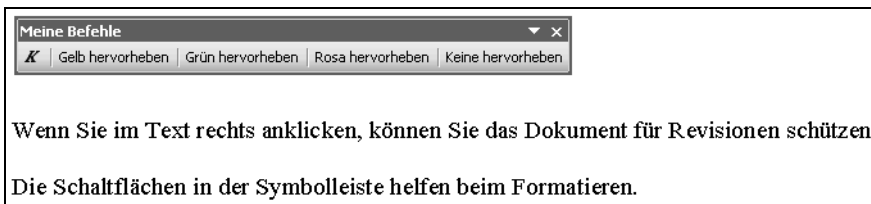
ctl.Style = msoButtonCaption
ctl.ToolTipText = "Markierung " & sFarbe & " hervorheben"
ctl.BeginGroup = True
Set ctl = Nothing
End Sub

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. 16.4** Schaltflächen, die über die Automatisierungsschnittstelle erstellt wurden




Die Beispieldatei *Bsp16\_03.doc* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap16*.

## Bearbeitungsfelder und Dropdownlisten

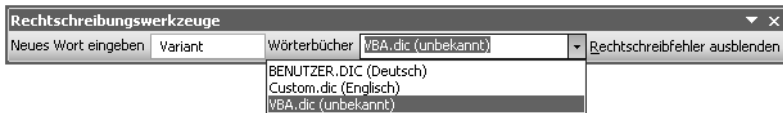
Obwohl der Word-Umgebung vier Befehlsleisten-Steuerelementtypen zur Verfügung stehen, können über die Benutzerschnittstelle nur zwei davon definiert werden: Schaltflächen (*msoControlButton*) sowie Popup-Menüs (*msoControlPopup*). Um Bearbeitungsfelder oder Dropdownlisten einer Symbolleiste hinzuzufügen, braucht es ein Makro.

Diese zwei Control-Typen sind in Abbildung 16.5 ersichtlich. Deren Style-Eigenschaft wurde jeweils auf *msoComboBox* 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. 16.5** Bearbeitungsfelder und Dropdownlisten, wie hier abgebildet, können nur programmtechnisch erstellt werden



Der Code, um diese beiden Symbolschaltflächen zu erstellen, befindet sich in Listing 16.5. Elemente werden einer Dropdownliste durch die AddItem-Methode hinzugefügt.

**Listing 16.5** Edit- und ComboBox-Schaltflächen einer Symbolleiste hinzufügen

```
Sub EditSchaltflächeErstellen()
    Dim ctl As Office.CommandBarControl

    Set ctl = Application.CommandBars("
        RechtschreibungsWerkzeuge").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(
        "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 *Bsp16\_04.doc* mit der gesamten Lösung finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap16*.

## Symbolleisten erstellen

Neue Symbolleisten werden mit der Add-Methode der CommandBars-Auflistung erstellt:

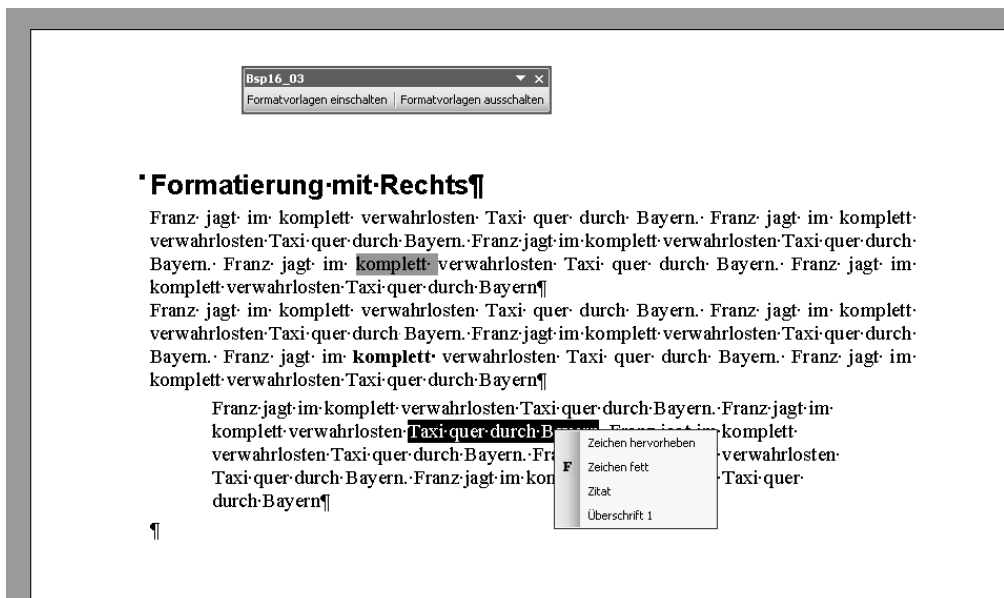
```
Add(Name, Position, MenuBar, Temporary)
```

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 16.6 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 16.6) wird die Aktivierung dieses Menüs ein- und ausgeschaltet (das Word-eigene Kontextmenü wird wieder aktiv).

Abbildg. 16.6 Ein Dokument mit einem eigens dafür erstellten Kontextmenü effizient formatieren



Das Kontextmenü wurde mit der Prozedur *KontextMenuErstellen* in Listing 16.6 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.

**Listing 16.6** 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 16.7 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 8 beschrieben.

Die Ereignis-Prozedur *app\_WindowBeforeRightClick* ruft *ShowKontext* auf, die das Kontextmenü einblendet. Die standardmäßige Handlung des Rechtsklicks wird unterdrückt (*Cancel=True*).

**Listing 16.7** Die Teile des Ereignisses, das das Rechtsklick-Verhalten steuert

```
'Inhalt des normalen Moduls
Public appEvents As New clsBsp16_03

Public Sub EreignisseEinschalten()
    Set appEvents.app = Word.Application
```

Listing 16.7 Die Teile des Ereignisses, das das Rechtsklick-Verhalten steuert (Fortsetzung)

```

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

```



2007

Bis einschließlich Word 2003 werden die Kontextmenüs über das Dialogfeld *Anpassen* (Abbildung 16.3) verwaltet. In Word 2007 gibt es keine ähnliche Möglichkeit – Anpassungen erfolgen zwingend über das Objektmodell. Um ein bestehendes Kontextmenü mit Befehlen zu ergänzen, müssen Sie dessen Namen kennen. Die Beispieldatei *Bsp16\_06.doc* enthält Code, der eine Liste vorhandener Kontextmenüs samt den Steuerelementen erstellt.

**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 Beispieldateien *Bsp16\_05.doc* und *Bsp16\_06.doc* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap16*.

## Zusammenfassung

Dieses Kapitel beschäftigte sich mit der programmtechnischen Erstellung von Symbolleisten. Beschrieben wurde dabei:

- Wie mit den Eigenschaften einer Symbolleiste umzugehen ist (Seite 708)
- Wie sich verschiedene Arten von Bedienelementen für eine Symbolleiste erstellen und mit einem Befehl verbinden lassen (Seite 714)
- Wie Symbolleisten erstellt werden (Seite 719)



## Kapitel 17

# Multifunktionsleisten (Ribbons)

### In diesem Kapitel:

Grundlagen	724
Das Ribbon erweitern	727
Zusammenfassung	760

In einigen Office 2007-Anwendungen – darunter auch Word – hat das Microsoft Office-Team eine neue Benutzerschnittstelle eingeführt. Sie ersetzt die bisher bekannten Menüs und Symbolleisten. In der deutschen Sprache wurde sie Multifunktionsleiste getauft. In Englisch sowie in der Entwicklerdokumentation heißt sie »Ribbon« oder auch »RibbonX« (das »X« steht für Extensibility, was so viel wie Erweiterung bedeutet).

## Grundlagen

Die neue Schnittstelle ist etwas gewöhnungsbedürftig. Beispielsweise bleibt die Multifunktionsleiste statisch am oberen Fensterrand haften und kann nicht verschoben werden. Der Benutzer kann sie auch nicht in der Anwendungsoberfläche anpassen. (Ihm steht aber eine andere, kleinere Leiste zur Verfügung, die im Abschnitt »Die neue Terminologie« in diesem Kapitel kurz vorgestellt wird.)

Statt über ein Objektmodell wird der Inhalt des Ribbon durch XML in der Dokumentstruktur oder in einem Add-In festgelegt. Anpassungen aus mehreren Quellen werden, analog zu den alten Symbolleisten, zusammengeführt. Falls mehrere Quellen genau die gleichen Ribbon-Elemente ansprechen, haben die Anpassungen der zuletzt geladenen Vorrang.

Im Gegensatz zu Symbolleisten ist es mit der Ribbon-Erweiterung nicht möglich, während der Laufzeit Steuerelemente zu erstellen oder diese zu löschen. Alle Steuerelemente müssen bereits in der XML-Datei definiert sein. Während der Laufzeit können sie durch den eigenen Code ein- und ausgeblendet oder gesperrt werden, auch andere Eigenschaften sind dynamisch anpassbar. Außenstehender Code hat aber darauf keinen Zugriff.

Einige Überlegungen stehen hinter der Entscheidung, diese neue Benutzerschnittstelle einzuführen. Dazu gehören unter anderen:

- Die Vielfalt an Funktionalität in Office-Anwendungen wie Word ist inzwischen so reichhaltig geworden, dass dem Benutzer viele Fähigkeiten entgangen sind. Sie sind in der Multifunktionsleiste schneller auffindbar.
- Immer häufiger stören COM Add-Ins von Drittanbietern den Benutzer bei seiner Arbeit: Sie blenden die Symbolleistenanpassungen anderer (inklusive diejenigen des Benutzers) aus oder entfernen ihre Befehle aus den Menüs. Dazu kommen die Speicheraufforderungen für die Vorlage *Normal.dot*, die nicht nur störend sind, sondern den Benutzer auch verunsichern. Die Arbeitsweise der Multifunktionsleiste ist so konzipiert, dass kein Add-In auf die Steuerelemente eines anderen zugreifen kann.
- Ferner haben Studien gezeigt, dass der Anwender eine berechenbare Positionierung der Befehle bevorzugt. Der Entwickler kann in die anwendungseigenen Befehlsgruppen des Ribbon nicht eingreifen; er kann höchstens Gruppen ausblenden oder Befehle sperren.
- Microsofts gegenwärtige Philosophie für Office-Entwickler sieht vor, dass möglichst viele Handlungen außerhalb der Anwendung stattfinden sollen. Ziel ist es, sowohl Dokumente als auch Benutzerschnittstellen erstellen und bearbeiten zu können, ohne dass die Word-Anwendung läuft oder gar installiert sein muss.
- Microsoft will die eigene Identität besser herausheben. Inzwischen stellen viele Anwendungen von Drittanbietern ihre Funktionalität durch Menüs und Symbolleisten zur Verfügung. Ein Konkurrent brüstet sich sogar damit, dass sein Textverarbeitungsprogramm äußerlich kaum von Microsoft Word zu unterscheiden sei. Mit der Ribbon-Technologie werden die Office-Anwendungen wieder eindeutig erkennbar. Sie macht auch über die QuickInfo klar, woher ein Befehl stammt.



Was bedeutet diese Änderung für den Office-Entwickler? Wer sich mit XML noch nicht befasst hat, wird dies jetzt zwangsläufig tun müssen, um mithalten zu können. Das XML der Ribbon-Erweiterung ist jedoch nicht besonders kompliziert. Kenntnisse von XSLT, XPath oder XMLDOM sind nicht erforderlich. Mit den Angaben in diesem Kapitel werden Ihnen die nötigen Puzzle-Teile in die Hände gegeben.

Obwohl der Entwurf einer Ribbon-Erweiterung etwas mühsam ist, da es keinen grafischen Entwurfsmodus gibt, ermöglicht sie sehr elegante Lösungen. Dem Entwickler stehen zusätzliche Steuerelemente zur Verfügung, wovon er für Symbolleisten nur träumen durfte. Allerdings ist etwas Umdenken nötig, um die neuen Vorgaben zu verinnerlichen.

Das Ärgernis mit Add-Ins anderer Anbieter, die Symbolleisten entfernen oder ausblenden, gehört der Vergangenheit an. Auch der Benutzer kann nichts mehr »kaputt machen«. Die ewige Sorge um CustomizationContext und die Unterbindung von lästigen Speichermeldungen wegen Änderungen in den Symbolleisten sind ebenfalls in dieser Beziehung kein Thema mehr.

Zusammenfassend gesagt, ist die Umstellung für den Entwickler einfacher, da dem »Poweruser«, (außer bei der Symbolleiste für den Schnellzugriff) keine Anpassungsmöglichkeiten mehr zur Verfügung stehen. Das Erlernen der Ribbon-Erweiterung ist nicht schwieriger als für das CommandBars-Objektmodell. Und die neue Technologie bringt einige Vorteile mit sich.

#### HINWEIS

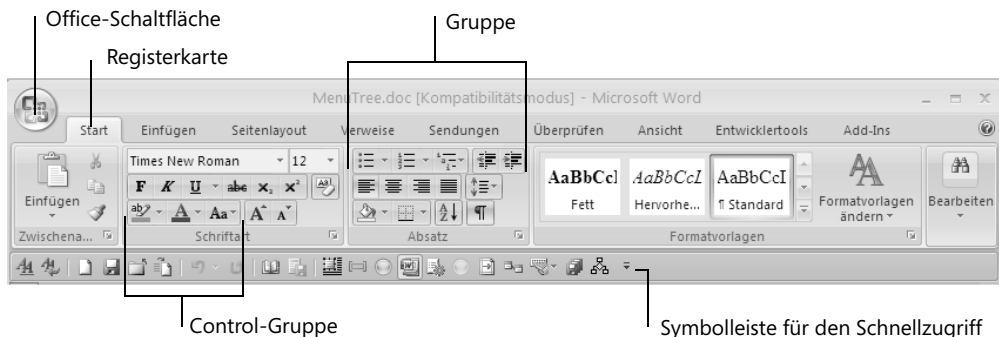
Inzwischen gibt es auf dem Markt einige Werkzeuge für den Benutzer, die ihm die Anpassung der Arbeitsumgebung ohne XML-Kenntnisse ermöglichen. Als Beispiele hierfür erwähnen wir:

- »Toolbar Toggle«, das die Erstellung von eigenen »Symbolleisten« ermöglicht. Diese können überall auf dem Bildschirm positioniert werden und unterstützen sowohl Word-eigene als auch Makro-Befehle. Dieses Werkzeug ist erhältlich unter <http://www.toolbartoggle.com>.
- »RibbonCustomizer«, das ausschließlich Anpassungen in der Multifunktionsleiste unterstützt. Auch dieses Werkzeug unterstützt Word-eigene sowie Makro-Befehle. Es ist in einer deutschen Version erhältlich bei <http://pschmid.net/office2007/ribboncustomizer/indexGerman.php>.

## Die neue Terminologie

Die neue Schnittstelle bringt auch neue Bezeichnungen mit sich. Einige werden in Abbildung 17.1 vorgestellt.

Abbildg. 17.1 Die Multifunktionsleiste als neue Word 2007-Benutzerschnittstelle



- **Ribbon:** Der »Behälter« aller übrigen Elemente. Es umfasst das, was wir früher als die Menüleiste kannten, sowie den Teil darunter, der ungefähr die Höhe drei alter Symbolleisten hat. Der untere Teil kann über das Kontextmenü durch den Befehl *Multifunktionsleiste minimieren* ausgeblendet werden, um mehr Platz auf dem Bildschirm zu erhalten.
- **Registerkarte:** In der englischsprachigen Umgebung auch als Tab bezeichnet. Es umfasst das, was aussieht wie Menüs, sowie die dazu gehörenden Befehle. In Abbildung 17.1 ist die Registerkarte *Start* aktiviert.
- **Gruppe:** Die viereckigen Felder innerhalb einer Registerkarte, die die Befehle gruppieren. Jede Gruppe ist an seinem unteren Rand beschriftet. In der Ecke rechts unten befindet sich manchmal eine kleine Schaltfläche (der so genannte »Dialoglauncher«), die ein Dialogfeld mit erweiterten Befehlen einblendet.
- **Control:** Ein Steuerelement, das einen Befehl ausführt
- **Control-Gruppe:** Eine Gruppe von Steuerelementen
- **Die Office-Schaltfläche:** Sie ersetzt mehr oder weniger das bisherige Menü *Datei*. Hierunter befinden sich zudem die Anwendungsoptionen. Es ist nicht möglich, diese Schaltfläche zu entfernen oder auszublenden.
- **Symbolleiste für den Schnellzugriff:** In der englischen Umgebung heißt sie offiziell »Quick Access Toolbar«, was aber sofort (auch in der Literatur) auf »QAT« verkürzt wurde. Diese Leiste gehört grundsätzlich dem Benutzer. Durch einen Klick mit der rechten Maustaste auf ein beliebiges Steuerelement und Auswahl des Eintrags *Zu Symbolleiste für den Schnellzugriff hinzufügen* kann sein Befehl der QAT zugefügt werden. Sie kann auch über die Anwendungsoptionen im Dialogfeld *Anpassen* mit Befehlen und Makros ergänzt werden. Eine begrenzte Anzahl vordefinierter Symbole stehen für Makros zur Verfügung. Es ist jedoch nicht möglich, eigene Grafiken als Symbole zu verwenden. Falls sich der Benutzer für seine Symbolleiste mehr Platz wünscht, kann diese unterhalb der Multifunktionsleiste positioniert werden, wie die Abbildung 17.1 veranschaulicht.

## Die Rolle von *CommandBars* in Word 2007

Das *CommandBars*-Objektmodell existiert noch, hat aber an Bedeutung verloren. Wichtig bleibt es für die Arbeit mit Kontextmenüs. Die über den früheren Menübefehl *Extras/Anpassen/Symbolleiste* erreichbare Schnittstelle wurde ersatzlos gestrichen, so dass Kontextmenüs nur programmtechnisch erstellt und angepasst werden können. Da es für den VBA-Entwickler sonst keine Möglichkeit gibt, Befehle in der Nähe des bearbeiteten Texts zu positionieren, bleibt noch eine Zeit lang Platz für das Objektmodell im Köcher des visierten Office-Entwicklers.

Bestehender Code, der Menüs und Symbolleisten anpasst, funktioniert weiterhin. Nur befinden sich alle seiner Ergebnisse auf einer einzigen Registerkarte namens *Add-Ins*. Es stehen drei Zeilen für Steuerelemente (Symbolschaltflächen) zur Verfügung. Nehmen die Symbolleisten und Symbolschaltflächen mehr Platz in Anspruch, als in der Fensterbreite zur Verfügung steht, erscheinen rechts und links schmale Balken mit Pfeilen, die das Blättern durch die ganze Breite ermöglichen.

### Neue *CommandBars*-Methoden für das Ribbon

Obwohl das *CommandBars*-Objektmodell für die Erstellung und Verwaltung von Symbolleisten nur noch für Kontextmenüs zuständig ist, wurde es gleichwohl in Bezug auf das Ribbon um einige wich-

tige Methoden erweitert. Diese sind in Tabelle 17.1 aufgelistet und ermöglichen die Ausführung der anwendungseigenen Befehle sowie die Abfrage gewisser Eigenschaften anwendungseigener Steuerelemente.

**HINWEIS** Mehr zum Thema `idMso` erfahren Sie im Abschnitt »Word-eigene Schaltflächen benutzen« in diesem Kapitel.

Tabelle 17.1 Neue Methoden des *CommandBars*-Objekts

Methode	Beschreibung	Datentyp des Rückgabewerts
<code>Execute(idMso)</code>	Führt den mit dem <code>idMso</code> verbundenen Befehl aus	Keiner
<code>GetEnabledMso(idMso)</code>	Ermittelt, ob das Steuerelement gesperrt ist	Boolean
<code>GetImageMso(idMso, Width, Height)</code>	Gibt das Steuerelementbild in der angegebenen Höhe und Breite zurück	<code>IPictureDisp</code>
<code>GetLabelMso(idMso)</code>	Gibt die Beschriftung des Steuerelements zurück	Zeichenkette
<code>GetPressedMso(idMso)</code>	Ermittelt, ob das Steuerelement gedrückt ist	Boolean
<code>GetScreenTipsMso(idMso)</code>	Gibt den Screentip (QuickInfo) zurück	Zeichenkette
<code>GetSuperTipsMso(idMso)</code>	Gibt den Text des Supertips zurück	Zeichenkette
<code>GetVisibleMso(idMso)</code>	Ermittelt, ob das Steuerelement sichtbar ist	Boolean

## Das Ribbon erweitern

Wie eingangs erwähnt, wird das Ribbon durch XML definiert, das entweder als Teil der Dokumentstruktur integriert ist, oder von einem COM Add-In geladen wird. Die zweite Methode wird in Kapitel 10 im Abschnitt über VSTO beschrieben. In diesem Kapitel beschränken wir uns auf die erste Methode. Die Grundlagen des XML-Codes bleiben in beiden Fällen die gleichen.

**HINWEIS** Sind Sie mit XML noch nicht vertraut, finden Sie mehr zum Thema im Kapitel 22.

## Werkzeuge

Die Leser, die vor zehn oder fünfzehn Jahren in WordBasic oder einer anderen Programmiersprache Benutzerschnittstellen entworfen haben, werden glauben, die Uhr hat sich zurückgedreht. Entwickler, die mit den modernen RAD-Entwicklerschnittstellen aufgewachsen sind, werden am Anfang wahrscheinlich noch mehr Mühe haben. Das Problem: Es gibt (noch) keinen grafischen Entwurfsmodus für die Ribbon-Erweiterung.

Jeder Teil der Ribbon-Erweiterung muss mühsam in XML geschrieben werden. Das visuelle Ergebnis wird erst sichtbar, wenn die XML-Datei gespeichert und das Dokument in Word geöffnet wurde. Es ist ein ständiges Hin und Her zwischen dem XML-Editor und der Word-Anwendung, das ein gutes visuelles Vorstellungsvermögen verlangt. Von bedeutender Hilfe ist ein geeignetes Werkzeug,

um die XML-Datei zu erstellen. Wer sich weniger mit dem Niederschreiben befassen muss, hat für das Gestalterische mehr Kapazität übrig.

Da XML-Dateien reine Textdateien sind, geht es auch mit einem beliebigen Texteditor. Sobald die Datei mehr als einige Zeilen enthält, gestaltet sich die Arbeit mit einem XML-Editor wesentlich bequemer. Einige dieser Editoren werden im Kapitel 22 im Abschnitt »XML-Namensräume und Schemas« aufgeführt. Sie übernehmen rein administrative Arbeiten wie das Einfügen von Einzügen, Abschlusselementtags und Anführungszeichen. Manche können das XML mit einem Schema validieren.

Sehr nützlich erweist sich der XML-Editor von Visual Studio 2005, da er »IntelliSense« anbietet. Falls Ihnen Visual Studio 2005 Professional (oder höher) zur Verfügung steht, empfehlen wir, VSTO 2005 SE herunterzuladen und zu installieren (es ist kostenlos). Erstellen Sie damit ein Word-Add-In (mehr dazu steht im Kapitel 10 im Abschnitt zu VSTO), schreiben Sie darin die XML-Datei und testen Sie das Add-In. Neben IntelliSense profitieren Sie auch vom in Visual Studio enthaltenen Debuggen-Modus, was Ihnen viel Zeit erspart. Sobald sicher steht, dass das XML fehlerfrei ist, kann die Datei wie im Abschnitt »Die Ribbon-Erweiterung in das Dokument einbinden« in diesem Kapitel beschrieben, problemlos in ein Word-Dokument eingebunden werden.

Visual Studio Professional wird nicht auf dem Rechner eines jeden Lesers installiert sein. Gute Dienste leistet auch der »Microsoft Office 2007 Custom UI Editor«, der ebenfalls kostenlos unter <http://openxmldeveloper.org/articles/CustomUIEditor.aspx> heruntergeladen werden kann. Obwohl er kein »IntelliSense« anbietet, prüft er die Wohlgeformtheit sowie Gültigkeit des XML-Codes mit dem Ribbon-Schema *customUI.xsd* und verweist auf mögliche Fehler. Zudem bindet er Grafikdateien für die Schaltflächen in das Zieldokument ein. Schließlich erstellt er alle benötigten Einträge für die »Relationships« im *\_rels*-Ordner. Dieses Werkzeug setzt eine Installation von .NET Framework in der Version 2.0 oder 3.0 voraus, das seinerseits ebenfalls kostenlos heruntergeladen werden kann. Dieses Werkzeug wird im Abschnitt »Einblick in die Erweiterung der »einfachen« Vorlage« in diesem Kapitel näher vorgestellt.



Die erwähnten Dateien und andere Hilfsmittel zum Thema finden Sie im Ordner *\Beilagen\Ribbon* auf der CD-ROM zum Buch.

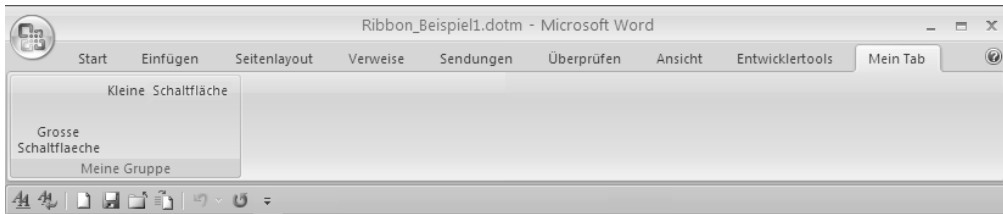
#### HINWEIS

Das .NET Framework ist nicht mit Visual Studio .NET zu verwechseln. Visual Studio .NET ist eine Programmierumgebung für das .NET Framework. Das .NET Framework ist die Umgebung, worin .NET-Anwendungen laufen. Links zu den verschiedenen Versionen von .NET Framework befinden sich auf der Webseite <http://www.microsoft.com/downloads/Browse.aspx?displaylang=de&categoryid=10>. Achten Sie darauf, dass Sie die richtige Version für Ihren Rechner auswählen.

## Die Grundlagen

Wir fangen mit einem sehr einfachen Beispiel an, um die einzelnen Schritte der Ribbon-Erweiterung zu veranschaulichen. Als Ziel streben wir das in Abbildung 17.2 ersichtliche Ergebnis an: Eine Ribbon-Erweiterung mit einer Registerkarte wird einer Dokumentvorlage hinzugefügt. Die zwei Schaltflächen in der einzigen Gruppe der Registerkarte führen VBA-Makros in der Vorlage aus.

Abbildg. 17.2 Das Ergebnis der im vorliegenden Abschnitt beschriebenen Ribbon-Anpassung



## Das Dokument vorbereiten

Eine Ribbon-Erweiterung kann sowohl einem Dokument als auch einer Dokumentvorlage hinzugefügt werden. Sie verhält sich analog zu den Symbolleisten in früheren Word-Versionen:

- Erweiterungen eines Dokuments sind nur sichtbar, wenn im Dokument gearbeitet wird.
- Erweiterungen in einer Dokumentvorlage stehen in allen Dokumenten zur Verfügung, die mit der Vorlage verbunden sind.
- Erweiterungen in einer als globales Add-In geladenen Dokumentvorlage stehen für alle Dokumente bereit.

Für die Erstellung der Ribbon-Erweiterung spielt es keine Rolle, ob mit einem Dokument oder mit einer Dokumentvorlage gearbeitet wird.

Da die Schaltflächen in unserem Beispiel VBA-Makros auslösen sollen, starten Sie zunächst den VBA-Editor von Word durch Drücken von **[Alt] + [F11]**. Die Makros müssen als Public Sub-Prozeduren mit dem passenden Parameter deklariert werden. Die Prozedur für eine Schaltfläche erfordert nur einen Parameter des Datentyps `Office.IRibbonControl`, wie das Listing 17.1 veranschaulicht.

Anschließend wird das Dokument geschlossen und gespeichert.

Listing 17.1 Die von der Ribbon-Erweiterung aufgerufenen VBA-Makros

```
Private Const msgboxTitel As String = "Ribbon-Beispiele"

Public Sub MeinButton1_Click(control As Office.IRibbonControl)
    MsgBox "Meine große Schaltfläche wurde angeklickt.", vbOKOnly, msgboxTitel
End Sub

Public Sub MeinButton2_Click(control As Office.IRibbonControl)
    MsgBox "Meine kleine Schaltfläche wurde angeklickt.", vbOKOnly, msgboxTitel
End Sub
```

## Die XML-Datei vorbereiten

Das XML für die Ribbon-Erweiterung wird in einer getrennten Datei erstellt, die anschließend in das Word 2007-XML-Dokument integriert wird. Wir fangen an mit der Erstellung eines Ordners für die XML-Datei:

1. Öffnen Sie den Windows-Explorer und wählen Sie den Ordner *Desktop*.
2. Klicken Sie mit der rechten Maustaste auf eine leere Stelle in der rechten Seite des Fensters und auf den Eintrag *Neu/Ordner* wählen.

3. Geben Sie dem neu erstellten Ordner den Namen *customUI*. Achten Sie auf die Großschreibung. (Dieser Name ist fest vorgeschrieben und darf nicht geändert werden.)

Starten Sie nun den Texteditor und geben Sie den XML-Code aus Listing 17.2 ein. Auch hier muss genau auf die Klein- und Großschreibung geachtet werden: XML kennt diesbezüglich keine Gnade und bestraft Fehler härter als jeder Schulmeister des 19. Jahrhunderts!

Einzig die von Ihnen festgelegten Werte, die nicht im Schema *customUI.xsd* vorgeschrieben sind, wie für die Attribute *label* (Beschriftung) und *id*, unterliegen nicht dieser Regel. Für diese Werte dürfen Sie die Großschreibung bestimmen. Der Wert für *onAction* entspricht dem Namen der von der Schaltfläche auszuführenden VBA-Prozedur.

Die Reihenfolge der ersten fünf Ebenen bleibt immer gleich: *customUI*, *ribbon*, *tabs*, *tab*, *group*. Alle Steuerelemente werden immer als Bestandteil einer Gruppe definiert. Das Aussehen und das Verhalten der Steuerelemente werden durch Attribute bestimmt. Beispielsweise wird die Größe der Schaltflächen durch das Attribut *size*, die Beschriftung durch das Attribut *label* und das auszuführende Makro durch das Attribut *onAction* festgelegt.

**Listing 17.2** Eine sehr einfache benutzerdefinierte Erweiterung für die Multifunktionsleiste

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui" >
  <ribbon>
    <tabs>
      <tab id="MeinTab" label="Mein Tab" >
        <group id="MeineGruppe" label="Meine Gruppe" >
          <button id="MeinButton1" label="Grosse Schaltflaeche" size="large"
            onAction="MeinButton1_Click" />
          <button id="MeinButton2" label="Kleine Schaltflaeche" size="normal"
            onAction="MeinButton2_Click" />
        </group>
      </tab>
    </tabs>
  </ribbon>
</customUI>
```

#### PROFITIPP

Es ist empfehlenswert, das Kontrollkästchen *Fehler in Benutzeroberflächen in Add-Ins anzeigen* in der Kategorie *Erweitert* der *Word-Optionen* im Abschnitt *Allgemein* zu aktivieren. Ansonsten werden eventuell durch das Ribbon-XML verursachte Fehlermeldungen nicht angezeigt. Diese liefern wichtige Informationen über die Position des Fehlers in der XML-Datei und warum Word die Erweiterung nicht akzeptieren kann.

Speichern Sie die Datei mit dem Namen *customUI.xml* in dem vorbereiteten Ordner auf dem Desktop. (Dieser Name ist ebenfalls fest vorgeschrieben.) Falls Sie Sonderzeichen wie Umlaute gebraucht haben, denken Sie daran, im Dialogfeld *Speichern unter* den Eintrag *Unicode* im Listenfeld *Codierung* auszuwählen.

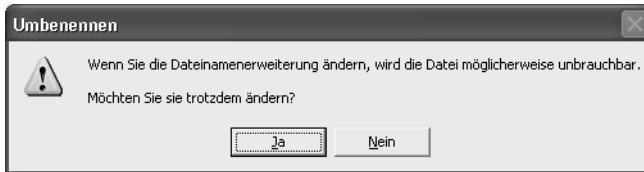
## Die Ribbon-Erweiterung in das Dokument einbinden

Wählen Sie im Windows-Explorer den Ordner mit dem vorbereiteten Word-Dokument und markieren Sie den Dokumenteintrag. Das Fenster soll so positioniert werden, dass der Desktop mit dem Ordner *customUI* sichtbar ist. Nun gehen Sie wie folgt vor:

1. Klicken Sie mit der rechten Maustaste auf den Dokumentnamen und wählen Sie im Kontextmenü den Eintrag *Umbenennen*.
2. Drücken Sie die `[Ende]`-Taste und tippen Sie `».zip«` (ohne Anführungszeichen) ein. Bestätigen Sie die Änderung mit der `[↵]`-Taste. Bestätigen Sie ebenfalls die Anfrage in Abbildung 17.3, ob Sie dies wirklich wollen. (Der Dokumentname wird mit der Endung `».zip«` ergänzt.)

Abbildg. 17.3

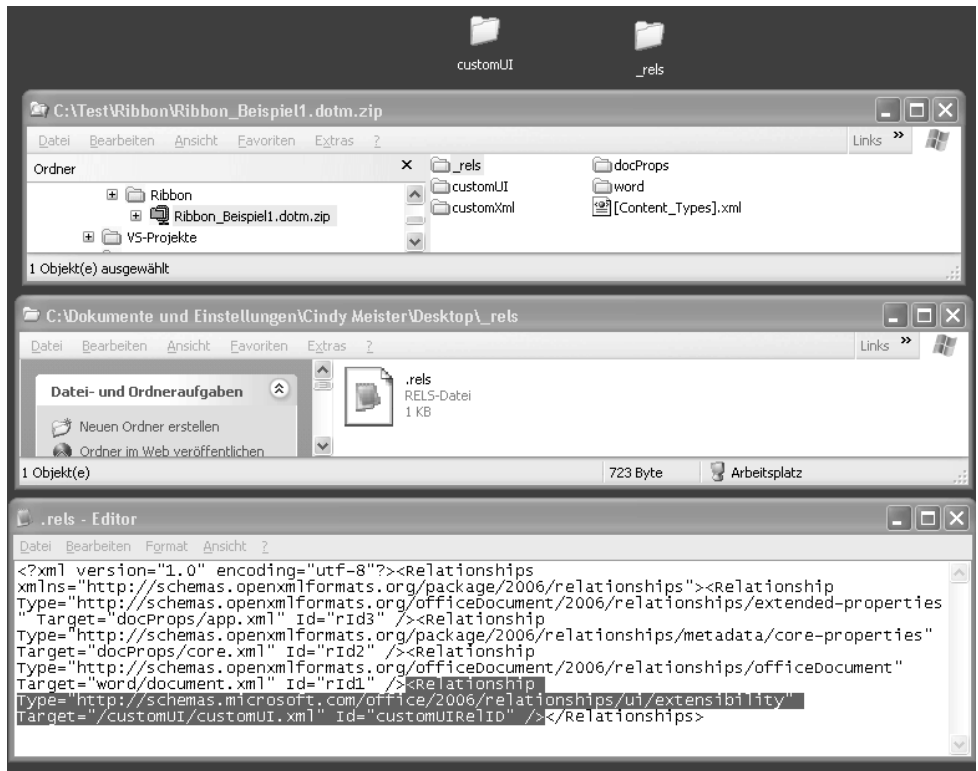
Sicherheitsabfrage, die bei Umbenennung einer Dateiendung erscheint



3. Klicken Sie nochmals mit der rechten Maustaste auf den Dokumentnamen, und wählen Sie im Kontextmenü den Untermenübefehl *Öffnen mit/ZIP-komprimierte Ordner*. (Arbeiten Sie mit Windows Vista, klicken Sie doppelt darauf.)
4. Ziehen Sie mit gedrückter linker Maustaste den Ordner *customUI* vom Desktop in das rechte Windows-Explorer-Fenster. Er wird der ZIP-Datei hinzugefügt.
5. Ziehen Sie mit gedrückter linker Maustaste den Ordner *\_rels* vom Windows-Explorer-Fenster zum Desktop und öffnen Sie den Ordner (siehe Abbildung 17.4).
6. Klicken Sie mit der rechten Maustaste auf die Datei *.rels*, und wählen Sie im Kontextmenü den Untermenübefehl *Öffnen mit/Texteditor* (siehe Abbildung 17.4). (Die *.rels*-Datei legt die Verbindungen zwischen den verschiedenen Teilen des Office-Dokuments fest.)
7. Suchen Sie das abschließende Element `</Relationships>` und positionieren Sie die Einfügemarke unmittelbar davor. Definieren Sie eine Verbindung zum Ordner *customUI* wie folgt. Achten Sie dabei unbedingt auf die Großschreibung (siehe Abbildung 17.4):

```
<Relationship Type="http://schemas.microsoft.com/office/2006/relationships/ui/
extensibility" Target="/customUI/customUI.xml" Id="customUIRelID" />
```

**Abbildg. 17.4**

 Bearbeitung eines Office-Dokuments als ZIP-Datei: der Inhalt des Ordners *\_rels*


8. Speichern und schließen Sie die Datei sowie den Ordner.
9. Im Windows-Explorer-Fenster markieren Sie den Ordner *\_rels* und löschen ihn.
10. Ziehen Sie mit festgehaltener linken Maustaste den Ordner *\_rels* vom Desktop in das Windows-Explorer-Fenster.
11. Auf der linken Seite des Windows-Explorer-Fensters klicken Sie auf den Ordner, worin sich die Dokument-Datei befindet (eine Ebene höher). Klicken Sie mit der rechten Maustaste auf den Dateinamen, wählen im Kontextmenü *Umbenennen*, drücken die **Ende**-Taste und entfernen die Dateiendung *.zip*.

Testen Sie das Ergebnis, indem Sie auf das Dokument doppelklicken, um es in Word zu öffnen. Wenn alles korrekt ausgeführt wurde, erscheint *Mein Tab* in der Multifunktionsleiste, wie in Abbildung 17.2.



Das Coding zum aktuellen Abschnitt finden Sie in den Beispieldateien *Beispiel1\_customUI.xml* sowie *Ribbon\_Beispiel1.dotm*. Diese befinden sich auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap17*.



## Vertieftes Wissen

Im vorangegangenen Abschnitt wurden die Grundlagen zu Anpassungen der Ribbon-Schnittstelle vorgestellt. Ziel dieses Abschnitts ist es nun, Ihnen anhand eines weiteren Beispiels einen vertiefenden Einblick in die Erweiterungsmöglichkeiten zu vermitteln. Der XML-Code für die Ribbon-Erweiterungen der drei Beispieldateien (Dokumentvorlagen) befindet sich in Listing 17.3, Listing 17.4 sowie Listing 17.5.

Anhand des Beispiels wird erläutert, wie mit der Multifunktionsleisten-Schnittstelle eine globale Vorlage anwendungsweit Befehle zur Verfügung stellt. Zudem werden Sie sehen, wie die Erweiterungen anderer Dokumentvorlagen mit der globalen Vorlage zusammenarbeiten. Der Hauptteil des Abschnitts befasst sich dann mit den verschiedenen Steuerelementen der Ribbon-Erweiterung und deren Attributen.

### Ribbon-Tabs teilen

Eingangs wurde erwähnt, die Ribbon-Erweiterungen mehrerer Dokumente und Add-Ins seien kumulativ, und die Erweiterung eines Dokuments oder eines Add-Ins habe keinen Zugriff auf diejenigen eines anderen. Das stimmt zwar, jedoch ist es möglich, eine Erweiterung so einzurichten, dass sie vom `customUI-XML` mehrerer Dokumente geteilt werden darf.

Nehmen wir an, eine Firma möchte eine Schnittstelle mit einigen wichtigen Befehlen für alle Dokumente zur Verfügung stellen. Jede Dokumentvorlage bringt zudem aufgabenspezifische Befehle mit sich, die auf der gleichen Registerkarte erscheinen.

In früheren Word-Versionen wurde eine Symbolleiste mit Makros in einer Vorlage erstellt. Die Vorlage wird als globales Vorlagen-Add-In geladen, so dass diese Symbolleiste immer zur Verfügung steht. Die anderen Vorlagen fügen ihre Symbolschaltflächen mit `CustomizationContext = ThisDocument` dieser Symbolleiste hinzu (damit sind die Symbolschaltflächen nur in Dokumenten von dieser Vorlage sichtbar).

Ähnliches geht auch mit Ribbon-Erweiterungen, aber das »Wie« ist nicht sofort erkennbar. Der Schlüssel liegt im Gebrauch eines Namensraums (Namespace). Wie in Kapitel 22 diskutiert, ermöglicht ein Namensraum die Zuteilung der XML-Elemente zu bestimmten Kategorien. Standardmäßig gehören alle Elemente von `customUI-XML` dem Namensraum `http://schemas.microsoft.com/office/2006/01/customui` – das steht im eröffnenden XML-Element. Die Angabe weiterer Namensräume ist aber erlaubt.

Vergleichen Sie die in Listing 17.3 wiedergegebene `customUI-XML` mit denjenigen in Listing 17.4 sowie in Listing 17.5. Sie weisen alle zusätzlich auf den Namensraum `xmlns:n="http://ISBN3-86063-989-7.com/RibbonXML"`. Das Präfix »n« für den Namensraum wird im Tab-Element als Teil des Attributwerts `idQ` (steht für qualifizierte ID) eingesetzt: `idQ="n:MeinTab"`. Dies erlaubt allen drei Erweiterungen die gleiche Registerkarte zu benutzen, wie in den folgenden Abbildungen ersichtlich. Die Gruppe links auf der Registerkarte *Mein Tab* ist in der globalen Vorlage (Listing 17.4) definiert. Die Gruppen rechts davon befinden sich in der jeweiligen mit dem Dokument verbundenen Dokumentvorlage.

#### HINWEIS

`idQ` kann auch mit Group- und weiteren Steuerelementen benutzt werden. Lesen Sie dazu den Teil 2 des Artikels »Customizing the 2007 Office Fluent Ribbon For Developers« von Frank Rice und Ken Getz, auf der Webseite <http://msdn2.microsoft.com/en-us/library/aa338199.aspx>. Alternativ können Sie im Schema `customUI.xsd` nachschauen.

**HINWEIS**

Das Schema *customUI.xsd* kann von der Microsoft-Webseite <http://www.microsoft.com/downloads/details.aspx?FamilyId=15805380-F2C0-4B80-9AD1-2CB0C300AEF9&displaylang=en> heruntergeladen werden.

**Listing 17.3** Eine Ribbon-Erweiterung mit Namensraum ermöglicht das gemeinsame Teilen der Ribbon-Elemente

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui" xmlns:n="http://
ISBN3-86063-989-7.com/RibbonXML" >
  <ribbon>
    <tabs>
      <tab idQ="n:MeinTab" label="Mein Tab" insertBeforeMso="TabHome" >
        <group id="MeineGruppe" label="Meine Gruppe" >

          <button id="MeinButton1" label="Große Schaltfläche" size="large"
            imageMso="AppointmentColorDialog" screentip="Meine große Schaltfläche"
            supertip="Der Supertip für meine große Schaltfläche"
            onAction="MeinButton1_Click" />

          <button id="MeinButton2" label="Kleine Schaltfläche" size="normal"
            image="Eightball" screentip="Meine kleine Schaltfläche"
            supertip="Der Supertip für meine kleine Schaltfläche"
            onAction="MeinButton2_Click" />

        </group>
      </tab>
    </tabs>
  </ribbon>
</customUI>
```

**Listing 17.4** Das XML einer zweiten Dokumentvorlage mit dem gleichen Namensraum wie in Listing 17.3

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui"
  xmlns:n="http://ISBN3-86063-989-7.com/RibbonXML" >
  <commands>
    <command idMso="FileSave" onAction="MeinFileSave" />
  </commands>
  <ribbon startFromScratch="false" >
    <tabs>
      <tab idQ="n:MeinTab" label="Mein Tab" insertBeforeMso="TabHome" keytip="M" >
        <group id="StartGruppe" label="Dokumente verwalten" >
          <splitButton id="VorlageWaehlen" size="large" showLabel="true" >
            <button idMso="FileNewDefault" />
            <menu>
              <button id="StartDok1" tag="Einfaches Dokument" onAction="VorlageAuswahl"
                label="Einfaches Dokument" />
              <button id="StartDok2" tag="Offerte oder Rechnung"
                onAction="VorlageAuswahl" label="Offerte oder Rechnung" />
            </menu>
          </splitButton>

          <button idMso="FileSave" />
          <button idMso="FileOpen" />
          <button idMso="FileClose" />
        </group>
      </tab>
    </tabs>
  </ribbon>
</customUI>
```

**Listing 17.4** Das XML einer zweiten Dokumentvorlage mit dem gleichen Namensraum wie in Listing 17.3 (Fortsetzung)

```

        <dialogBoxLauncher>
            <button idMso="FileNew" />
        </dialogBoxLauncher>
    </group>
</tab>
</tabs>
</ribbon>
</customUI>

```

**Listing 17.5** Der XML-Code der dritten der Beispieldokumentvorlagen, die alle die gleiche Registerkarte teilen

```

<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui" xmlns:n="http://
ISBN3-86063-989-7.com/RibbonXML" onLoad="ribbonGeladen" >

    <commands>
        <command idMso="MenuPublish" enabled="false" />
    </commands>

    <ribbon>
        <tabs>
            <tab idQ="n:MeinTab" label="Mein Tab" insertBeforeMso="TabHome" >
                <group id="GroupOfferte_Rechnung" label="Offerte &amp;&amp; Rechnung" >
                    <box id="cbBox" boxStyle="vertical" >
                        <buttonGroup id="UntergruppeFormatieren">
                            <toggleButton id="WordSchriftGruppe" imageMso="GroupFont"
                                screentip="Schriftartgruppe einblenden"
                                supertip="Blendet die Word-Schriftartgruppe ein und aus."
                                onAction="WordSchriftGruppe_Click" keytip="H" />

                            <toggleButton id="WordAbsatzGrupp" imageMso="GroupParagraph"
                                onAction="WordAbsatzGruppe_Click" keytip="A" />

                            <toggleButton idMso="ApplyStylesPane" showLabel="false" keytip="F" />

                            <button id="sep_GroupOfferteRechnung_1" keytip="1" />

                            <button id="FelderAktualisieren1" showLabel="false"
                                screentip="Berechnungen aktualisieren" imageMso="HappyFace"
                                visible="true" supertip="Aktualisiert die Felder in der Produktentabelle."
                                onAction="FelderAktualisieren_Click" keytip="B" />
                        </buttonGroup>

                        <checkBox id="cbAnschrift" label="Empfängeradresse"
                            screentip="Empfängeradresse eingeben"
                            supertip="Blendet die Gruppe mit für das Eingabe der Anschrift ein und aus."
                            onAction="cbWerkzeuge_Pressed" keytip="E" />

                        <checkBox id="cbProdukte" label="Produkte auswählen"
                            screentip="Produkte auswählen"
                            supertip="Blendet die Gruppe mit für das Einfügen der Produkte ein und aus."
                            onAction="cbWerkzeuge_Pressed" keytip="P" />
                    </box>
                </group>
            </tab>
        </tabs>
    </ribbon>
</customUI>

```

**Listing 17.5** Der XML-Code der dritten der Beispieldokumentvorlagen, die alle die gleiche Registerkarte teilen *(Fortsetzung)*

```

<group id="GroupAnschrift" label="Empfängeradresse"
  getVisible="Group_GetVisible" >
  <box id="boxEmpfängerName" boxStyle="horizontal" >
    <editBox id="editEmpfängerVorname" label="Vorname"
      sizeString="wwwwwwwwwwwwwww" onChange="Adresse_OnChange"
      getText="editBox_GetText" keytip="R" />

    <editBox id="editEmpfängerNachname" label="Nachname"
      sizeString="wwwwwwwwwwwwwww" onChange="Adresse_OnChange"
      getText="editBox_GetText" keytip="M" />
  </box>
  <box id="boxEmpfängerStraße" boxStyle="horizontal" >
    <editBox id="editEmpfängerStraße" label="Straße"
      sizeString="wwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwww"
      onChange="Adresse_OnChange" getText="editBox_GetText" keytip="B" />
  </box>
  <box id="boxEmpfängerPLZOrt" boxStyle="horizontal" >
    <editBox id="editEmpfängerPLZ" label="PLZ"
      sizeString="00000" onChange="Adresse_OnChange" getText="editBox_GetText"
      keytip="L" />

    <editBox id="editEmpfängerOrt" label="Ort" sizeString="wwwwwwwwwwwwwwwwwwwww"
      onChange="Adresse_OnChange" getText="editBox_GetText" keytip="T" />

    <checkBox id="cbAnrede" label="Anrede" onAction="cbAnrede_Pressed"
      getPressed="cbAnrede_GetPressed" keytip="D" />
  </box>
  <box id="boxAnrede" boxStyle="vertical" >
    <labelControl id="labelComboAnrede" label="Anrede auswählen oder eingeben"
      getVisible="comboAnrede_GetVisible" />

    <comboBox id="comboAnrede" sizeString="Die Anrede auswählen / eingeben"
      screentip="Die Anrede auswählen"
      supertip="Sie können die Anrede aus der Liste wählen, oder
        selber eine eingeben. Der Nachname des Empfängers
        wird automatisch hinzugefügt."
      onChange="comboAnrede_Select" getText="comboAnrede_GetText"
      getVisible="comboAnrede_GetVisible" keytip="W" >
      <item id="Anrede1" label="Sehr geehrter Herr" />
      <item id="Anrede2" label="Sehr geehrte Frau" />
      <item id="Anrede3" label="Sehr geehrter Herr Dr." />
      <item id="Anrede4" label="Sehr geehrte Frau Dr." />
    </comboBox>
  </box>
</group>

<group id="GroupProdukte" label="Produkte auswählen"
  getVisible="Group_GetVisible" >
  <gallery id="galleryProdukte" label="Produktliste" onAction="Gallery_OnSelect"
    getItemImage="Gallery_GetItemImage" getItemLabel="Gallery_GetItemLabel"
    getItemScreentip="Gallery_GetItemScreentip"
    getItemSupertip="Gallery_GetItemSupertip" getItemCount="Gallery_GetItemCount"
    getItemID="Gallery_GetItemID" columns="3" itemHeight="50" itemWidth="50"
    keytip="K" />

```

**Listing 17.5** Der XML-Code der dritten der Beispieldokumentvorlagen, die alle die gleiche Registerkarte teilen (Fortsetzung)

```
<button id="FelderAktualisieren2" label="Berechnungen aktualisieren"
  screentip="Berechnungen aktualisieren" imageMso="HappyFace" visible="true"
  supertip="Aktualisiert die Felder in der Produktentabelle."
  onAction="FelderAktualisieren_Click" />
</group>

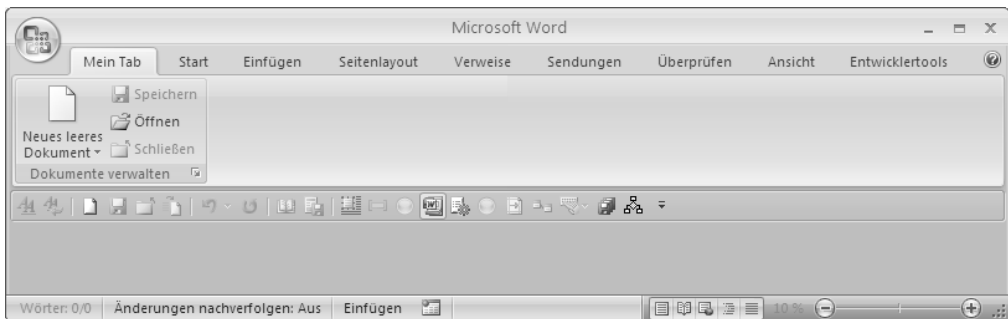
<group idMso="GroupFont" getVisible="GroupFont_GetVisible" />

<group idMso="GroupParagraph" getVisible="GroupParagraph_GetVisible" />
</tab>
</tabs>
</ribbon>
</customUI>
```

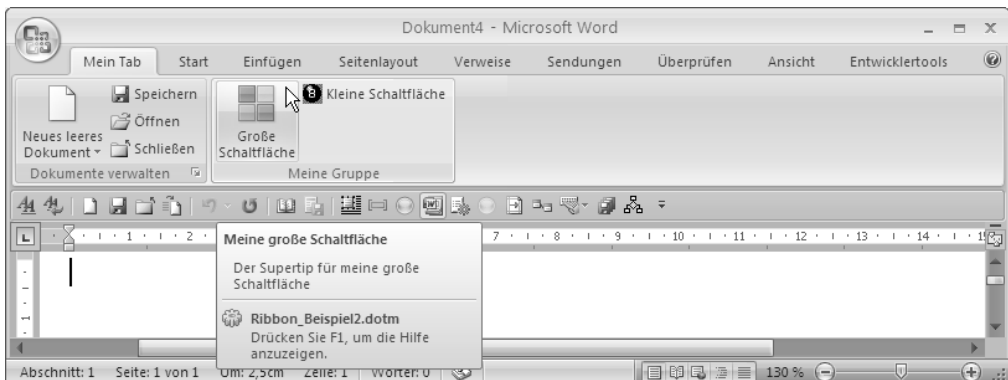


Den XML-Code im obigen Abschnitt finden Sie in den Beispieldateien *Beispiel2\_customUI.xml*, *Beispiel3\_customUI.xml* und *Beispiel4\_customUI.xml*. Diese befinden sich auf der CD-ROM zum Buch im Ordner `\Beispiele\Kap17`.

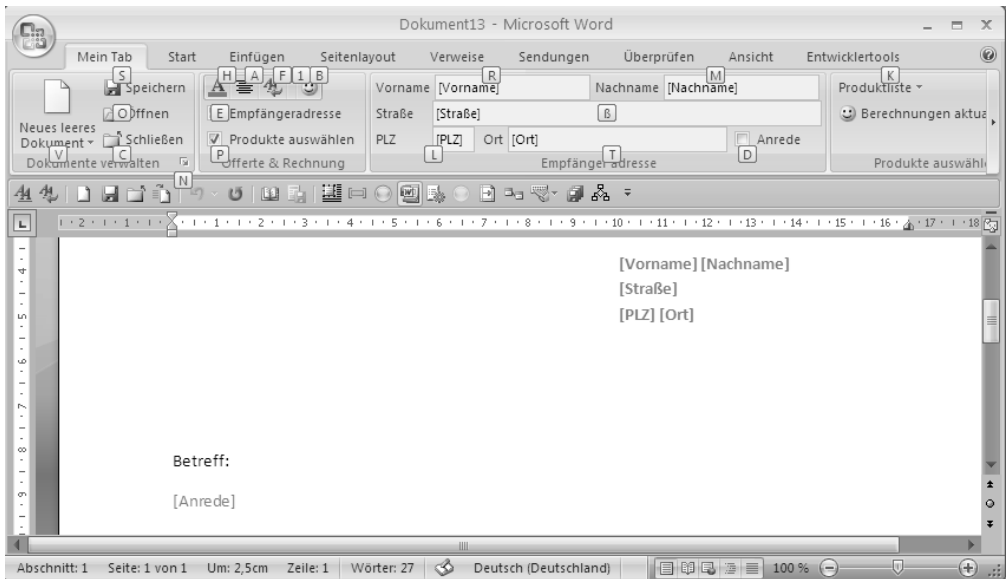
**Abbildg. 17.5** Die Ribbon-Erweiterung der globalen Vorlage



**Abbildg. 17.6** Die Registerkarte in Abbildung 17.5 wird ergänzt durch die Ribbon-Erweiterung der einfachen Vorlage



**Abbildg. 17.7** Die Ribbon-Erweiterung der Vorlage für Offerten mit allen Gruppen eingblendet, plus Keytips



## Word-eigene Schaltflächen benutzen

Schaltflächen, die in der Multifunktionsleiste der Anwendung vorhanden sind, können auch in benutzerdefinierten Ribbon-Erweiterungen eingesetzt werden. Mit dem Befehl werden auch die Art des Steuerelements sowie seine Grafik und seine Beschriftung übernommen. Anstelle des Attributs `id` beziehungsweise `idQ` wird das Attribut `idMso` gebraucht. Beispiele hierfür finden Sie in Listing 17.4, wo die Word-eigenen Befehle *DateiNeu*, *DateiÖffnen*, *DateiSchließen* sowie *DateiSpeichern* Wiederverwendung finden.

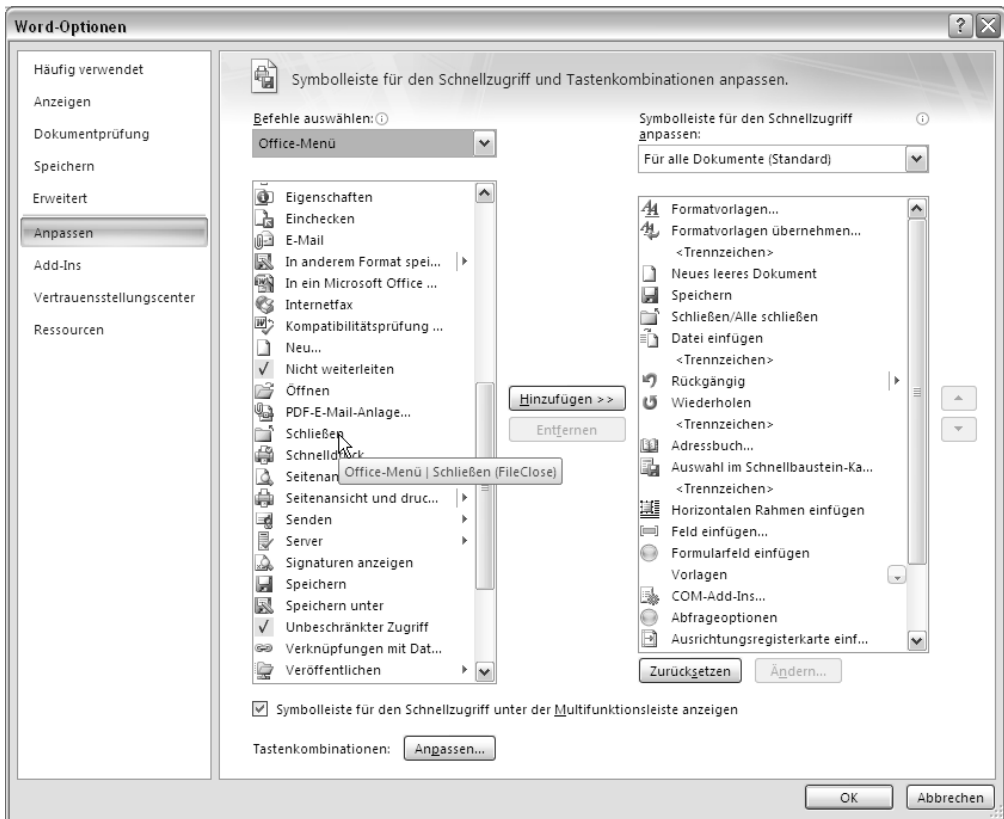
Bitte beachten Sie, dass die Art des Steuerelements für einen Word-eigenen Befehl fest vorgeschrieben ist. Ist der `idMso`-Befehl beispielsweise mit einem `button`-Steuerelement verbunden, kann es nur in einem `button`-Steuerelement eingesetzt werden.

Die folgende Codezeile bindet die Schaltfläche sowie den Befehl für *DateiSchließen* in die Ribbon-Erweiterung ein.

```
<button idMso="FileClose" />
```

Eine echte Herausforderung ist das Herausfinden des passenden Werts: Wie lautet wohl die »gute« Zeichenkette für das Attribut `idMso`? Da gibt es einen kleinen Trick:

1. Klicken Sie auf den kleinen Pfeil am rechten Ende der Symbolleiste für den Schnellzugriff und wählen Sie im zugehörigen Dropdownmenü den Eintrag *Weitere Befehle* aus.
2. Suchen Sie in der Liste *Befehle auswählen* den gewünschten Befehl aus.
3. Positionieren Sie den Mauszeiger darauf, um die QuickInfo zu lesen. Am Ende steht in Klammern der Wert für `idMso` (siehe Abbildung 17.8).

Abbildg. 17.8 Den Wert für das Attribut *idMso* herausfinden

**HINWEIS** Microsoft stellt eine Liste der *idMso*-Werte in Form einer Excel-Arbeitsmappe zur Verfügung unter <http://www.microsoft.com/downloads/details.aspx?familyid=4329d9e9-4d11-46a5-898d-23e4f331e9ae&displaylang=en#filelist>. Patrick Schmid bietet ebenfalls unter <http://pschmid.net/office2007/ribbonx/reference/index.php> eine Liste der Office-eigene Symbole als Excel-Arbeitsmappe an. Die Befehlsnamen sind jedoch in beiden Fällen in englischer Sprache.

## Word-Befehle übersteuern

In allen Word-Versionen ist es möglich, einer Prozedur den Namen eines Word-eigenen Befehls zu geben, um die Ausführung dieses Befehls abzufangen und abzuändern (Einzelheiten lesen Sie in Kapitel 20). Die Ribbon-Erweiterung bietet diese Funktionalität ebenfalls an (was sie dann auch auf COM Add-Ins ausdehnt).

Die abzufangenden Befehle werden unter dem Element `<commands>` unmittelbar vor dem Element `<ribbon>` aufgelistet. Die folgenden Codezeilen stammen aus dem Listing 17.4:

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui"
  xmlns:n="http://ISBN3-86063-989-7.com/RibbonXML" >
  <commands>
    <command idMso="FileSave" onAction="MeinFileSave" />
  </commands>
  <ribbon startFromScratch="false" >
```

Das `<command>`-Element akzeptiert nur vier Attribute: `enabled`, `getEnabled`, `idMso` (erforderlich) sowie `onAction`.

- **enabled:** Legt fest, ob der Befehl ausführbar ist (»false« bedeutet, der Befehl kann nicht ausgeführt werden)
- **getEnabled:** Die Erweiterung ruft eine Prozedur ab, die den Wert dynamisch zurückgibt
- **idMso:** Der Name des Befehls
- **onAction:** Die Prozedur, die anstelle des Befehls auszuführen ist

Die `onAction`-Prozedur für das `Command`-Element hat zwei Parameter. Der erste enthält das rufende Steuerelement. Mit dem zweiten legt die Prozedur fest, ob die Ausführung des anwendungseigenen Befehls zu annullieren ist (bei einem Wert von »false« wird der Befehl wie üblich ausgeführt). Für dieses Beispiel darf das Dokument erst gespeichert werden, wenn die Dokumenteigenschaft *Titel* einen Wert hat (Listing 17.6).

**Listing 17.6** Diese Prozedur wird anstelle des Word-eigenen Befehls *Dokumentspeichern* ausgeführt.

```
Sub MeinFileSave(control As Office.IRibbonControl, ByRef cancelDefault)
  Do While Len(ActiveDocument.BuiltInDocumentProperties(wdPropertyTitle)) = 0
    'Blendet das Dialogfeld Dokumenteigenschaften ein
    CommandBars.FindControl(ID:=750).Execute
  Loop
  cancelDefault = False
End Sub
```

#### PROFITIPP

Auf diesem Weg können Befehle in der *Office*-Schaltfläche außer Kraft gesetzt werden. Die *Office*-Schaltfläche selbst kann nicht aus der Multifunktionsleiste entfernt werden. Zudem ist es auch nicht möglich, die beiden Schaltflächen rechts unten – *Word-Optionen* sowie *Word beenden* – außer Kraft zu setzen.

## Einen Tab anwählen

Die Ribbon-Erweiterung bietet keine Möglichkeit, einer bestimmten Registerkarte den Fokus zu geben. Beim Öffnen eines Dokuments wird immer die erste Registerkarte der Multifunktionsleiste angewählt. Falls dies eine Registerkarte in der Ribbon-Erweiterung sein soll, muss es an erster Stelle positioniert werden, was mit dem Attribut `insertBeforeMso` erreicht wird (`TabHome` ist die `idMso` der Registerkarte *Start*):

```
<tab idQ="n:MeinTab" label="Mein Tab" insertBeforeMso="TabHome" >
```

Der einzige Umweg, um eine bestimmte Registerkarte anzuwählen, ist, mit `SendKeys` die Tastenschläge für den Keytip durchzugeben. Wenn der Benutzer die `[Alt]`-Taste drückt, erscheinen neben jeder Registerkarte ein oder mehrere Buchstaben. Durch Drücken dieser Buchstabenkombination



wird die Registerkarte angewählt und die Keytips seiner Steuerelemente eingeblendet, wie in Abbildung 17.7 ersichtlich. Um einen Keytip für die eigenen Registerkarten, Gruppen und Steuerelemente festzulegen, fügen Sie dem XML-Element das Attribut `keytip` zu:

```
<tab idQ="n:MeinTab" label="Mein Tab" insertBeforeMso="TabHome" keytip="M" >
```

Jedoch ist bei `SendKeys` Vorsicht geboten, da bei mehrfacher Belegung einer Taste die Multifunktionsleiste dem Buchstaben eine Zahl hinzufügt – und schon schlägt `SendKeys` fehl. Nachfolgend ist die Kommandozeile angegeben, um mit `SendKeys` die Registerkarte *Start* anzuwählen:

```
SendKeys "%R"
```

Die Ribbon-Erweiterung bei Null anfangen

Falls Sie alle vordefinierten Gruppen ausschalten möchten, ist es nicht notwendig, diese einzeln aufzulisten und deren `visible`-Attribut auf »falsch« festzulegen. Stattdessen kann das `ribbon`-Element um das Attribut `startFromScratch="true"` ergänzt werden.

PROFITIPP

Eigentlich gehört die Symbolleiste für den Schnellzugriff dem Benutzer und der Entwickler hat unter normalen Umständen keinen Zugriff darauf. Erfordert jedoch eine Lösung die Anpassung dieser Symbolleiste, geht das nur mit `startFromScratch="true"`. Die von der Symbolleiste für den Schnellzugriff unterstützten Elemente sind im weiter vorne erwähnten Artikel »Customizing the 2007 Office Fluent Ribbon For Developers« (Teil 2) aufgelistet.

Die Steuerelemente

Die Multifunktionsleiste unterstützt natürlich mehr Steuerelemente als die bisher vorgestellte, herkömmliche Form einer Schaltfläche. Eine kurze Übersicht bietet die Tabelle 17.2.

Tabelle 17.2 Liste der dem Ribbon zur Verfügung stehenden Steuerelemente

XML-Element	Bezeichnung	Beschreibung
<button>		Schaltfläche
<checkBox>		Kontrollkästchen
<comboBox>		Kombinationsfeld
<dialogBoxLauncher>		Startet ein Dialogfeld (blendet es ein)
<dropDown>		Dropdownfeld
<dynamicMenu>		Ein Menü, das bei Laufzeit erstellt wird
<editBox>		Bearbeitungsfeld
<gallery>		Katalog oder Sammlung; bietet eine grafische Liste zur Auswahl an
<labelControl>		Bezeichnungsfeld
<menu>		Menü
<menuSeparator>		Menütrennlinie

**Tabelle 17.2** Liste der dem Ribbon zur Verfügung stehenden Steuerelemente (Fortsetzung)

XML-Element Bezeichnung	Beschreibung
<separator>	Trennlinie
<splitButton>	Trennschaltfläche; vereint eine Schaltfläche mit einem Menü
<toggleButton>	Umschaltfläche

Jedes Steuerelement hat einen eigenen Satz an Attributen. Manche haben viele Attribute gemeinsam, andere Attribute gelten nur für ein oder zwei Elemente. Eine vollständige Liste finden Sie im zweiten Teil des Artikels »Customizing the Office (2007) Fluent Ribbon For Developers«, auf der Webseite <http://msdn2.microsoft.com/en-us/library/aa338199.aspx>. Alternativ können sie dem *CustomUI.xsd*-Schema entnommen werden.

Die folgenden Beispiele veranschaulichen den Gebrauch vieler dieser Steuerelemente und ihrer Attribute.

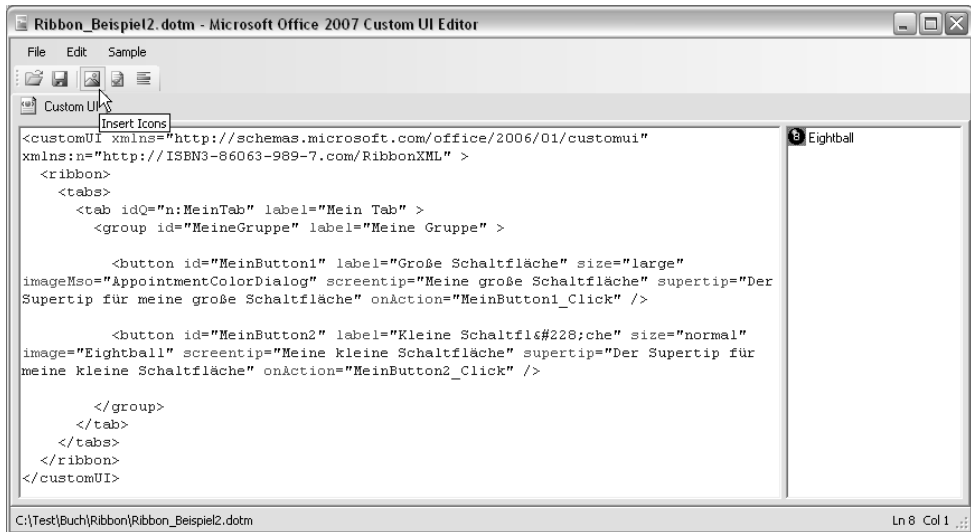
### Einblick in die Erweiterung der »einfachen« Vorlage

Die Ribbon-Erweiterung der ersten der drei Beispielvorlagen entspricht grundlegend der des ersten Beispiels mit einigen zusätzlichen Attributen. Sie zeigt lediglich die Möglichkeiten der Schaltfläche auf. Auf dieser Grundlage stellen wir das Werkzeug »Microsoft Office 2007 Custom UI Editor« vor. Um die Dokumentvorlage zu erstellen, gehen Sie wie folgt vor:

1. Legen Sie ein neues Dokument in Word 2007 an und speichern Sie es als Dokumentvorlage mit Makros (Dateiendung *.dotm*). Schließen Sie danach die Datei wieder.
2. Falls Sie die Anwendung »Microsoft 2007 Custom UI Editor« heruntergeladen haben, starten Sie diese.  
(Andernfalls geben Sie den XML-Code aus dem Listing 17.3 in einen beliebigen Texteditor oder XML-Editor ein und gehen weiter vor, wie weiter vorne in diesem Kapitel im Abschnitt »Die Ribbon-Erweiterung in das Dokument einbinden« beschrieben.)
3. Über die Befehlsfolge *File/Open* des Custom UI Editors öffnen Sie die gerade erstellte Dokumentvorlage.
4. Geben Sie den customUI-XML-Code aus Listing 17.3 in das Fenster ein.
5. Mit dem Custom UI Editor gestaltet sich das Einbinden von Grafiken in das Dokument problemlos. Klicken Sie einfach auf die Schaltfläche *Insert Icons*, navigieren Sie zum Ordner, in dem sich die Grafik befindet, und wählen Sie diese aus. Die Grafik erscheint in einer Spalte rechts neben dem XML mit seiner Bezeichnung, wie die Abbildung 17.9. veranschaulicht. Diese Bezeichnung wird als Wert für das Attribut *image* eingesetzt.

Abbildg. 17.9

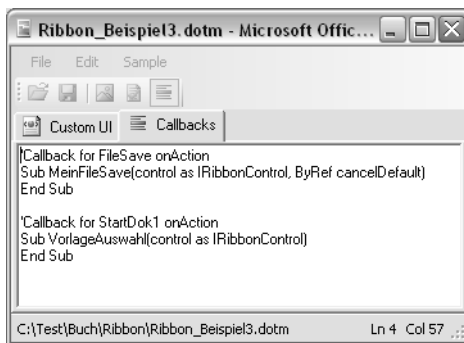
Eine Grafikdatei für die Multifunktionsleiste in ein Word-Dokument speichern



6. Der Custom UI Editor kann die Strukturen für die von `onAction`-Attributen benötigten Makros erstellen. Dies ist sehr hilfreich, da alle Argumente mitgeliefert werden. Um die Erstellung zu veranlassen, klicken Sie auf die nebenstehend gezeigte Schaltfläche *Generate Callbacks*. Ein zweites Fenster (Abbildung 17.10) mit dem Code erscheint, worin Sie die Zeilen markieren und kopieren können (Klick mit der rechten Maustaste oder `[Strg] + [C]`); eine Bearbeitung des Codes ist in dieser Umgebung nicht möglich.

Abbildg. 17.10

Die Makrostrukturen für `onAction`-Prozeduren vom Custom UI Editor erstellen lassen



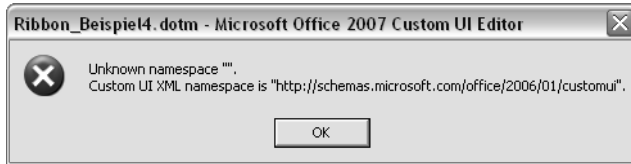
#### HINWEIS

Die von der Ribbon-Erweiterung aufgerufenen Prozeduren werden allgemein als »Callbacks« bezeichnet. Eine Liste der Callbacks (für die Programmiersprachen C#, VBA, C++ sowie VB.NET) mit ihren Parametern befindet sich im Teil 3 des Artikels von Frank Rice und Ken Getz (<http://msdn2.microsoft.com/en-us/library/ms406046.aspx>).



7. Klicken Sie auf die nebenstehend gezeigte Schaltfläche *Validate*, um das XML einer Kontrolle zu unterziehen. Falls der XML-Code wohlgeformt ist und dem *customUI.xsd*-Schema entspricht, erscheint die Meldung »CustomUI XML is well-formed!« Enthält das XML einen oder mehrere Fehler, wird darauf hingewiesen. Das Beispiel in Abbildung 17.11 informiert beispielsweise, dass der erforderliche Namensraum im Element *customUI* fehlt.

**Abbildg. 17.11** Der Custom UI Editor gibt hilfreiche Information zu Fehlern im *customUI*-XML-Code



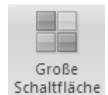
8. Haben Sie die Bearbeitung beendet, speichern Sie Ihre Arbeit. Die Vorlage wird automatisch geschlossen.

#### **ACHTUNG**

Ein Dokument kann in den Custom UI Editor nicht geöffnet werden, wenn es bereits in Word geöffnet ist. Es ist hingegen möglich, ein im Editor geöffnetes Dokument in Word zu öffnen. Denken Sie aber daran, dass in diesem Fall Änderungen im Dokument nicht gespeichert werden, obwohl Word dazu keine Warnung einblendet. Zusammenfassend: Sie können XML-Erweiterungen in Word testen, während das XML im Custom UI Editor geöffnet und bearbeitbar ist. Fehlerkorrekturen im Dokument (beispielsweise Anpassungen des VBA-Codes) gehen jedoch beim Schließen des Dokuments verloren, auch wenn Sie das Dokument ausdrücklich speichern.


### Die Attribute des Steuerelements *Schaltfläche*

Das einzig verwendete Steuerelement in dieser Ribbon-Erweiterung ist die Schaltfläche (button). Dieses Beispiel erläutert einige der meist gebrauchten Attribute; deren Wirkung ist in Abbildung 17.6 ersichtlich. Diese werden von vielen Steuerelementen unterstützt und immer gleich eingesetzt.



- **tag:** Ermöglicht die Hinterlegung von Informationen, die dem Programmablauf dienen (siehe auch das Listing 17.7)
- **label:** Die Beschriftung des Steuerelements
- **size:** Legt die Größe der Schaltfläche fest. Der Wert *large* veranlasst die Multifunktionsleiste, eine große Schaltfläche zu zeichnen. Standardmäßig werden kleinere Schaltflächen mit dem Wert *normal* dargestellt.
- **imageMso:** Zeichnet ein Office-eigenes Symbol (im Beispiel das vierfarbige Viereck auf der großen Schaltfläche) auf das Steuerelement
- **image:** Zeichnet eine im Dokument gespeicherte Grafik (im Beispiel der Billardball mit der Nummer »8«) auf der Schaltfläche
- **screentip:** Wenn der Mauszeiger über ein Steuerelement positioniert wird, erscheint eine Quick-Info. Der *screentip* bildet den oberen Teil (die Überschrift) der QuickInfo.

**HINWEIS**

Es stellt sich immer wieder die Frage, ob es möglich ist, den untersten Teil der QuickInfo zu entfernen oder anzupassen, um beispielsweise eigene Hilfetexte anzubieten (siehe Abbildung 17.6). Die Antwort darauf ist »Nein«. Immer mehr Add-Ins kommen auf den Markt und greifen in die Office-Umgebung ein. Viele Benutzer wissen nicht, dass störendes Verhalten und Probleme nicht durch Microsoft-Produkte verursacht werden, sondern durch Add-Ins von Drittanbietern, und geben Microsoft die Schuld. Nach diesen Erfahrungen will das Office-Team klar machen, wer in der Benutzerschnittstelle wofür verantwortlich ist. Zudem muss der Benutzer jederzeit ein solches Add-In ausschalten können. Die in der QuickInfo über die Taste  angebotene Hilfe enthält entsprechende Angaben.

- **supertip:** Der supertip erscheint im mittleren Teil der QuickInfo und dient als Hilfetext für den Befehl
- **onAction:** Führt die angegebene Prozedur aus

## Die globale Dokumentvorlage

Die globale Dokumentvorlage dieses Beispiels steuert hauptsächlich die Dokumentverwaltung. Sie enthält eine Gruppe mit Befehlen für das Erstellen, das Öffnen und das Schließen von Dokumenten, die in Abbildung 17.5 ersichtlicht ist. Der XML-Code dafür befindet sich in Listing 17.4. Diese Dokumentvorlage setzt neben gewöhnlichen Schaltflächen die folgenden Ribbon-Erweiterungselemente ein.

### Das Steuerelement *dialogBoxLauncher*

In der unteren rechten Ecke vieler Gruppen der Multifunktionsleiste befindet sich ein winziges Rechteck. Dies ist eine Schaltfläche, die ein Dialogfeld mit erweiterter Funktionalität einblendet. Diese Schaltfläche ist das Steuerelement *dialogBoxLauncher*.

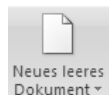
```
<dialogBoxLauncher>
  <button idMso="FileNew" />
</dialogBoxLauncher>
```

Dieses Steuerelement muss ein Unterelement des Typs *button* enthalten. Es befindet sich zwingend an letzter Stelle der Elemente einer Gruppe.

In diesem Beispiel wird das Word-eigene Dialogfeld *Neues Dokument* eingeblendet. Das Unterelement *button* darf aber auch eine Prozedur aufrufen, die ein benutzerdefiniertes Formular einblendet.

### Das Steuerelement *splitButton*

Eine Trennschaltfläche besteht aus drei Teilen: der Trennschaltfläche als »Behälter« mit seinen Attributen, einer gewöhnlichen Schaltfläche (*button*) oder einer Umschaltfläche (*toggleButton*), sowie eines Menüs (*menu*).



In der Beispiel-Dokumentvorlage ist die Schaltfläche mit dem Word-eigenen Befehl verbunden, der ein neues, leeres Dokument erstellt. Das Menü enthält seinerseits zwei Schaltflächen, die das gleiche in Listing 17.7 ersichtliche Makro aufrufen.

```
<splitButton id="VorlageWaehlen" size="large" >
  <button idMso="FileNewDefault" />
  <menu>
    <button id="StartDok1" tag="Einfaches Dokument" onAction="VorlageAuswahl"
      label="Einfaches Dokument" />
    <button id="StartDok2" tag="Offerte oder Rechnung" onAction="VorlageAuswahl"
      label="Offerte oder Rechnung" />
  </menu>
</splitButton>
```

Diese Prozedur unterscheidet anhand der Werte des tag-Attributes des rufenden Steuerelements, was zu tun ist. Je nach Steuerelement wird die passende Dokumentvorlage für das zu erstellende Dokument gewählt.

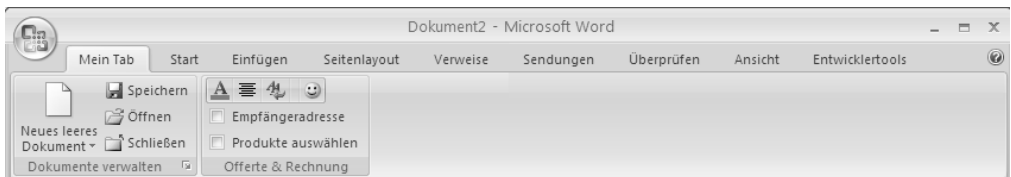
**Listing 17.7** Ein neues Dokument von einer bestimmten Dokumentvorlage erstellen

```
Sub VorlageAuswahl(control As Office.IRibbonControl)
  Dim pfad As String
  pfad = ThisDocument.path & "\"
  Select Case control.Tag
    Case "Einfaches Dokument"
      Documents.Add Template:=pfad & "Ribbon_Beispiel2.dotm"
    Case "Offerte oder Rechnung"
      MsgBox pfad & "Ribbon_Beispiel4.dotm"
    Case Else
  End Select
End Sub
```

## Die dritte Beispielsvorlage

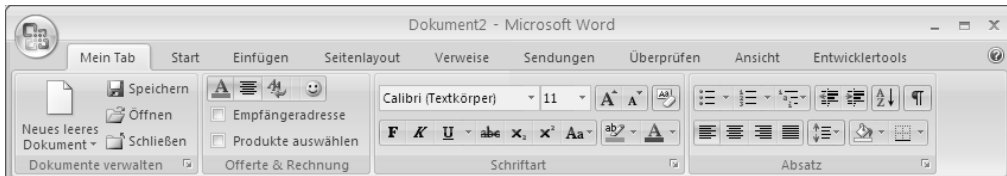
Die dritte Beispielsvorlage ist für die Erstellung von Offerten und Rechnungen gedacht. Nur die erste Gruppe ist beim Öffnen eines Dokuments sichtbar (vergleichen Sie die Abbildung 17.12 mit der Abbildung 17.7). Ihre Steuerelemente blenden nach Bedarf Gruppen mit weiteren Werkzeugen ein, die für das Schreiben einer Offerte oder einer Rechnung benötigt werden.

**Abbildg. 17.12** Die Ribbon-Erweiterung *Mein Tab* beim Öffnen einer Offerte; nur die erste Gruppe ist sichtbar



Die Umschaltflächen (toggleButton) in der obersten Zeile blenden die Word-eigenen Gruppen *Schriftart* sowie *Absatz* (Abbildung 17.13) und den Aufgabenbereich *Formatvorlage übernehmen* ein. Die vierte Schaltfläche aktualisiert die Feldfunktionen im Dokument.

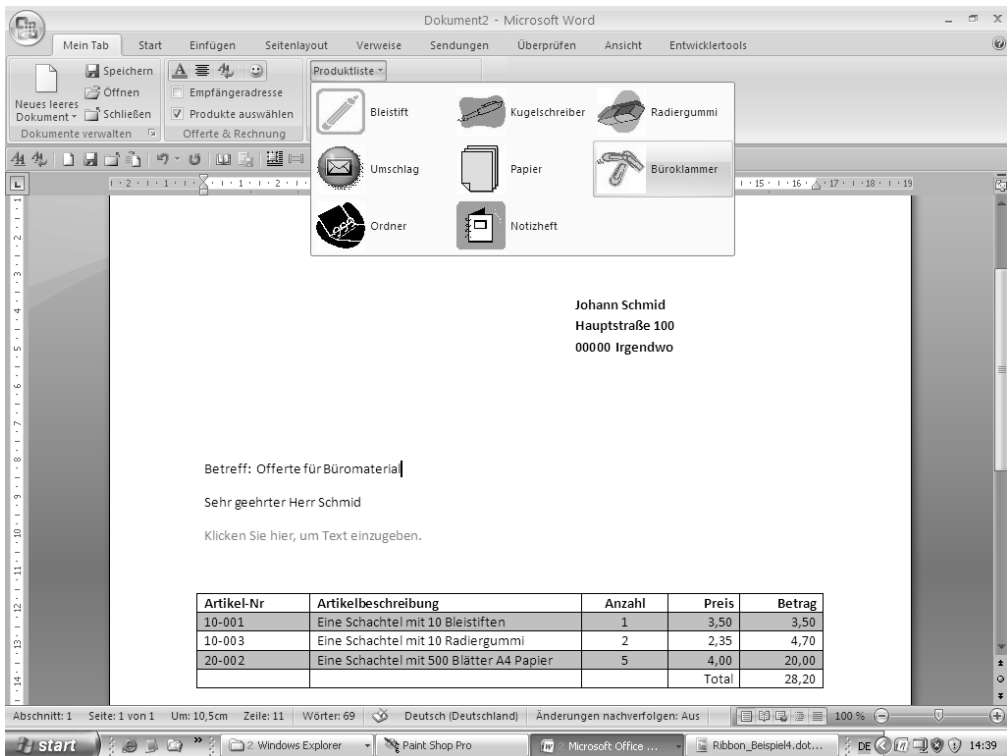
**Abbildg. 17.13** Die meistbenutzten Formatierungsbefehle stehen auf der gleichen Registerkarte mit den Dokumentwerkzeugen zur Verfügung



In der zweiten und dritten Zeile befinden sich zwei Kontrollkästchen (checkBox), die Gruppen für die Eingabe der Empfängeradresse (diese sind in Abbildung 17.7 ersichtlich) sowie für die Auswahl von Produkten ein- und ausblenden.

Die Steuerelemente der Gruppe *Empfängeradresse* sind vom Typ editBox. Ihr Inhalt wird aus den und in die vordefinierten Inhaltssteuerelemente im Dokument gelesen bzw. geschrieben. In dieser Gruppe befindet sich noch ein Kontrollkästchen, das ein Kombinationsfeld (comboBox) ein- und ausblendet. Damit kann der Benutzer eine vordefinierte Anrede auswählen oder eine eigene eingeben. Die Anrede wird beim Schreiben in das Inhaltssteuerelement automatisch ergänzt mit dem Nachnamen der Empfängeradresse.

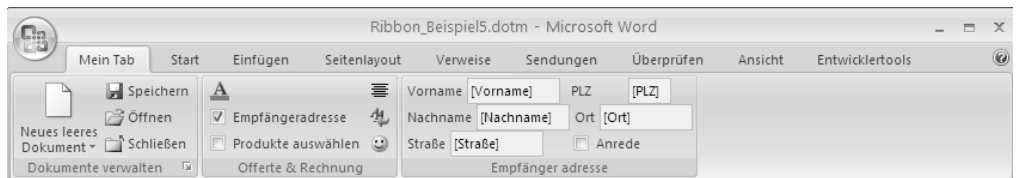
**Abbildg. 17.14** Produkte in der Offerte über einen Katalog aufnehmen



Die Gruppe *Produkte auswählen* enthält einen Katalog (gallery) mit Produkten (Abbildung 17.14). Im Dokument werden diese in einer Tabelle aufgelistet mit Feldfunktionen für die Berechnungen. Dem Benutzer bleibt lediglich die Aufgabe, die Quantität einzugeben und die Berechnungen zu aktualisieren, wofür es in der Multifunktionsleiste mehrere Schaltflächen gibt. Beim Anwählen eines Produkts wird die Tabelle um eine Zeile mit den nötigen Angaben erweitert. Falls die Tabelle noch nicht im Dokument vorhanden ist, wird sie zunächst erstellt.

Dieses Beispiel veranschaulicht zudem die Anordnung und das Ausrichten von Steuerelementen in den Gruppen. Die von der Ribbon-Erweiterung hierfür zur Verfügung gestellten Möglichkeiten sind zwar auf die Elemente *box* und *buttonGroup* sowie das Attribut *sizeString* begrenzt, aber mit etwas Fantasie und Ausdauer ist ein ansprechendes Ergebnis möglich. Wie die Ribbon-Erweiterung ohne den Einsatz dieser Elemente aussehen würde, ist in Abbildung 17.15 ersichtlich. Die Elemente werden von oben nach unten und von links nach rechts angeordnet; die Ausrichtung ist eher chaotisch.

Abbildg. 17.15 Von der Ribbon-Erweiterung frei angeordnete Steuerelemente



Nachfolgend werden die Steuerelemente im Detail zusammen mit dem mit diesen verbundenen VBA-Code vorgestellt.

### Das Steuerelement *box*

Das Steuerelement *box* gruppiert die Steuerelemente innerhalb einer Gruppe. Es darf alle befehlsausführenden Steuerelemente sowie weitere *box*- und *buttonGroup*-Elemente enthalten.

Nebst den üblichen Attributen wie *id* und *visible* ist das Attribut *boxStyle* erwähnenswert. Mit diesem wird festgelegt, ob die darin enthaltenen Steuerelemente von links nach rechts oder von oben nach unten angelegt werden. Die zwei gültigen Werte sind *horizontal* (waagrecht) sowie *vertical* (senkrecht).

Das Beispiel veranschaulicht, wie das Steuerelement das Layout beeinflusst. Es legt fest, welche neben- und welche untereinander liegen, sowie ihre Reihenfolge. Im folgenden Codefragment veranlasst die *Box*, dass die darin enthaltenen *editBox*-Steuerelemente nebeneinander liegen.

```
<box id="boxEmpfängerName" boxStyle="horizontal" >
  <editBox id="editEmpfängerVorname" label="Vorname" sizeString="www"
    onChange="Adresse_OnChange" getText="editBox_GetText" keytip="R" />
  <editBox id="editEmpfängerNachname" label="Nachname" sizeString="www"
    onChange="Adresse_OnChange" getText="editBox_GetText" keytip="M" />
</box>
```

### Das Steuerelement *buttonGroup*

Die Wirkung des Steuerelements *buttonGroup* ist ähnlich der einer *Box*, aber es unterstützt nicht alle Steuerelemente. Nur die Steuerelemente *button*, *dynamicMenu*, *gallery*, *splitButton* sowie *toggleButton* dürfen Teil einer *buttonGroup* sein.





Dieses Steuerelement hat keine erwähnenswerten Attribute. Die darin enthaltenen Steuerelemente werden waagerecht aneinander gereiht.

Hier sorgt die `buttonGroup` dafür, dass die Schaltflächen ohne störende Abstände in der oberen Zeile schön zusammenbleiben.

```
<buttonGroup id="UntergruppeFormatieren" >
  <toggleButton id="WordSchriftGruppe" imageMso="GroupFont" screentip="Schriftartgruppe
    einblenden" supertip="Blendet die Word-Schriftartgruppe ein und aus."
    onAction="WordSchriftGruppe_Click" keytip="H" />
  <toggleButton id="WordAbsatzGrupp" imageMso="GroupParagraph"
    onAction="WordAbsatzGruppe_Click" keytip="A" />
  <toggleButton idMso="ApplyStylesPane" showLabel="false" keytip="F" />
  <button id="sep_GroupOfferteRechnung_1" keytip="1" />
  <button id="FelderAktualisieren1" showLabel="false"
    screentip="Berechnungen aktualisieren" imageMso="HappyFace" visible="true"
    supertip="Aktualisiert die Felder in der Produktentabelle."
    onAction="FelderAktualisieren_Click" keytip="B" />
</buttonGroup>
```

### Das Steuerelement *toggleButton*



Eine gewöhnliche Schaltfläche führt immer den gleichen Befehl aus; von einer Umschaltfläche wird erwartet, dass sie zwischen zwei Eigenschaften oder Zuständen hin und her wechselt und diesen Zustand gleichzeitig widerspiegelt. Oft wird sie benutzt, um etwas ein- und auszublenden, so wie im vorliegenden Beispiel die Schriftart- und Absatz-Gruppen.

Ein besonderes Attribut des Steuerelements ist `getPressed`. Damit kann der Zustand der Umschaltfläche bei der Initialisierung der Ribbon-Erweiterung festgelegt werden. Standardmäßig wird sie im Zustand »aus« (entspricht dem Wert »falsch«) geladen. Mit der dem Attribut zugewiesenen Prozedur kann dynamisch entschieden werden, welchen Zustand die Umschaltfläche haben soll.

Das Attribut `onAction` der beiden Umschaltflächen im obigen Codefragment zeigt auf Prozeduren in der Vorlage. Stellvertretend wird in Listing 17.8 der Code für die Umschaltfläche mit der `id="WordSchriftGruppe"` angezeigt, da beide Prozeduren nach dem gleichen Muster aufgebaut sind.

Bei Betätigung dieser Umschaltfläche wird die Prozedur `WordSchriftGruppe_Click` ausgeführt. Die `onAction`-Prozedur einer Umschaltfläche verfügt im Gegensatz zu einer gewöhnlichen Schaltfläche über zwei Parameter. Der erste ist identisch: Er übergibt das rufende Steuerelement. Der andere ist vom Datentyp *Boolean* und übermittelt, ob der Zustand der Umschaltfläche »ein« oder »aus« ist. Anhand dieser Parameter wird entschieden, welche Handlungen auszuführen sind.

In diesem Beispiel wird der Wert einer globalen Variablen (`groupFontVisible`) festgelegt, die den Wert der Parameter zwischenspeichert. Da die Arbeitsweise der Ribbon-Erweiterung es verbietet, von außerhalb auf die Steuerelemente Einfluss zu nehmen, darf der Code die Gruppe nicht direkt ein- oder auszublenden.

### Das Attribut *getVisible*

Stattdessen muss die Ribbon-Erweiterung erneut initialisiert werden, was die letzte Befehlszeile der Prozedur veranlasst: `thisRibbon.Invalidade`. Dadurch wird das `getVisible`-Callback der Gruppe ausgelöst:

```
<group idMso="GroupFont" getVisible="GroupFont_GetVisible" />.
```

Die Prozedur `Group_GetVisible` übergibt den Wert der globalen Variablen an die Parameter `returnedVal`. Ist er »wahr«, wird die Gruppe eingeblendet, sonst wird sie ausgeblendet.

**Listing 17.8** Die Prozeduren, um eine Gruppe mittels einer Umschaltfläche ein- und auszublenden

```
'Werden über mehrere Prozeduren eingesetzt
Public thisRibbon As Office.IRibbonUI
Private groupFontVisible As Boolean

'Wird beim Laden der Ribbon-Erweiterung ausgeführt
Sub ribbonGeladen(ribbon As IRibbonUI)
    Set thisRibbon = ribbon
End Sub

'Callback for WordSchriftGruppe onAction
Sub WordSchriftGruppe_Click(control As IRibbonControl, pressed As Boolean)
    If pressed = True Then
        groupFontVisible = True
    Else
        groupFontVisible = False
    End If
    thisRibbon.Invalidate
End Sub

Sub GroupFont_GetVisible(control As IRibbonControl, ByRef returnedVal)
    returnedVal = groupFontVisible
End Sub
```

### Das Attribut *onLoad*

Nun stellt sich die Frage, woher das Objekt `thisRibbon` stammt. Auch dieses ist als globale Variable vom Typ `Office.RibbonUI` deklariert. Es steht für die Ribbon-Erweiterung des Dokuments. Das Objekt wurde der Variablen in der Prozedur `ribbonGeladen` zugewiesen. Diese Prozedur wurde ihrerseits, anlässlich des Ladens des Dokuments, bei der ersten Initialisierung der Ribbon-Erweiterung über `onLoad="ribbonGeladen"` aufgerufen. `onLoad` ist ein Attribut des obersten Elements der XML-Datei `customUI`. Dieser Callback wird im Gegensatz zu allen anderen nur dieses eine Mal ausgeführt.

Ein `RibbonUI`-Objekt stellt nur zwei Methoden und keine Eigenschaften zur Verfügung. Die eine Methode `Invalidate` wurde oben vorgestellt. Die andere Methode `InvalidateControl` veranlasst die Initialisierung eines einzelnen Steuerelements über seine `id`. Normalerweise wird diese zweite Methode bevorzugt, da sie effizienter ist. Weil jedoch ein Word-eigenes Steuerelement (was die Gruppe *Schriftart* mit der `idMso="GroupFont"` darstellt) in der Ribbon-Erweiterung mehrmals vorkommen kann, muss die gesamte Ribbon-Erweiterung neu initialisiert werden: Die Ribbon-Erweiterung kann nicht wissen, welches Steuerelement zu aktualisieren ist – sie haben ja alle den gleichen `idMso`-Wert. Ein Beispiel für `InvalideControl` befindet sich in Listing 17.9 für das Steuerelement `checkBox`.

### Das Steuerelement *checkBox*

Ein Kontrollkästchen ist einer Umschaltfläche sehr ähnlich. Beide werden für die Verwaltung eines »Ein-oder-aus«-Zustands eingesetzt und haben viele Attribute gemein-



sam. Ein `checkBox`-Steuerelement hat jedoch weniger Gestaltungsmöglichkeiten und verfügt daher über weniger Attribute. Es unterstützt beispielsweise keine Grafik und seine Größe ist festgeschrieben, weshalb Attribute wie `getImage`, `getImageMso`, `getShowImage`, `showImage` sowie `size` fehlen.

Im vorliegenden Beispiel rufen beide Kontrollkästchen der Gruppe *Offerte & Rechnung* die gleichen `onAction`- sowie `getPressed`-Prozeduren auf, die in Listing 17.9 ersichtlich sind.

```
<checkBox id="cbAnschrift" label="Empfängeradresse"
  screentip="Empfängeradresse eingeben"
  supertip="Blendet die Gruppe mit für die Eingabe der Anschrift ein und aus."
  onAction="cbWerkzeuge_Pressed" keytip="E" />
```

Der Ablauf ist fast gleich wie bei den Umschaltflächen: Ein Klick auf das Kontrollkästchen löst die Prozedur `cbWerkzeuge_Pressed` aus, die den Zustand des Steuerelements in einer globalen Variablen zwischenspeichert. In diesem Fall kann die Methode `InvalidateControl` eingesetzt werden, da es sich nicht um Word-eigene Gruppen handelt. Diese neue Initialisierung veranlasst die Gruppe `GroupAnschrift`, den Callback für `getVisible` aufzurufen, der die Prozedur `Group_GetVisible` ausführt.

```
<group id="GroupAnschrift" label="Empfängeradresse" getVisible="Group_GetVisible" >
```

**Listing 17.9** Die mit den *checkBox*-Steuerelementen verbundenen Prozeduren

```
Private groupAnschriftVisible As Boolean
Private groupProdukteVisible As Boolean



Sub cbWerkzeuge_Pressed(control As IRibbonControl, pressed As Boolean)
  Select Case control.id
    Case "cbAnschrift"
      If pressed = True Then
        groupAnschriftVisible = True
      Else
        groupAnschriftVisible = False
      End If
      thisRibbon.InvalidateControl "GroupAnschrift"
    Case "cbProdukte"
      If pressed = True Then
        groupProdukteVisible = True
      Else
        groupProdukteVisible = False
      End If
      thisRibbon.InvalidateControl "GroupProdukte"
    Case Else
      MsgBox "Unbekanntes Kontrollkästchenelement"
  End Select
End Sub

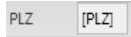
'Callback für alle Nicht-idMso-Gruppen
Sub Group_GetVisible(control As IRibbonControl, ByRef returnedVal)
  Select Case control.id
    Case "GroupAnschrift"
      returnedVal = groupAnschriftVisible
    Case "GroupProdukte"
      returnedVal = groupProdukteVisible
    Case Else
```

**Listing 17.9** Die mit den *checkBox*-Steuerelementen verbundenen Prozeduren (Fortsetzung)


```
MsgBox "Unbekannte Gruppe: " & control.Tag
End Select
End Sub
```

### Das Steuerelement *editBox*

Ein Bearbeitungsfeld ist für die Texteingabe vorgesehen. Zu Zeiten der Symbolleiste machte es meist Sinn, ein Dialogfeld einzublenden. Die Anordnungsmöglichkeiten der Ribbon-Erweiterung sind aber so flexibel, dass sie sich für beschränkte Benutzereingaben nutzen lässt. Der größte damit verbundene Nachteil ist, dass es weder mit der - noch der -Taste möglich ist, zum nächsten Feld zu wechseln. Der Benutzer muss dafür entweder die Maus oder die entsprechenden Keytips benutzen.



Das Steuerelement hat einige interessanten Attribute:

- **getText**: Holt den im Bearbeitungsfeld anzuzeigenden Text
- **maxLength**: Legt die maximale Anzahl der einzugebenden Zeichen fest
- **onChange**: Wird ausgeführt, wenn der Benutzer die Texteingabe bestätigt. Dies erfolgt beispielsweise durch Klicken außerhalb des Felds oder durch Drücken der -Taste (ein *editBox*-Steuerelement besitzt kein *onAction*-Attribut).
- **sizeString**: Legt die Breite des Bearbeitungsfeldes fest. Es wird breit genug gezeichnet, dass die angegebene Zeichenkette darin Platz hat.

Ein Beispiel dafür:

```
<editBox id="editEmpfängerPLZ" label="PLZ" maxLength="5" sizeString="00000"
onChange="Adresse_OnChange" getText="editBox_GetText" keytip="L" />
```

Die von *getText* und *onChange* aufgerufenen Prozeduren sind in Listing 17.10 ersichtlich. Alle *editBox*-Steuerelemente des Beispiels rufen die gleichen Prozeduren auf. Welches Steuerelement der Auslöser war, wird anhand des *id*-Werts ermittelt.

Bei *onChange* wird der Inhalt des Bearbeitungsfeldes in das entsprechende Inhaltssteuerelement geschrieben. (Beim Öffnen des Dokuments bzw. beim Erstellen eines neuen Dokuments wird durch alle Inhaltssteuerelemente im Dokument geschleift. Die Steuerelemente werden der Auflistung *Offerte\_ContentControls* zugefügt, ihre *Tag*-Eigenschaft wird als *Index*-Wert festgelegt.)

Das *getText*-Callback macht das Gegenteil: Es schreibt den Inhalt des passenden Inhaltssteuerelements in das rufende Bearbeitungsfeld.

**Listing 17.10** Die vom Steuerelement *editBox* aufgerufenen Prozeduren

```
Sub Adresse_OnChange(control As IRibbonControl, text As String)
    Select Case control.id
        Case "editEmpfängerVorname"
            Offerte_ContentControls.Item("EmpfängerVorname").Range.text = text
        Case "editEmpfängerNachname"
            Offerte_ContentControls.Item("EmpfängerNachname").Range.text = text
        Case "editEmpfängerStraße"
            Offerte_ContentControls.Item("EmpfängerStraße").Range.text = text
        Case "editEmpfängerPLZ"
            Offerte_ContentControls.Item("EmpfängerPLZ").Range.text = text
    End Select
End Sub
```

Listing 17.10 Die vom Steuerelement *editBox* aufgerufenen Prozeduren (Fortsetzung)

```

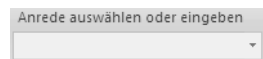
    Offerte_ContentControls.Item("EmpfängerPLZ").Range.text = text
    Case "editEmpfängerOrt"
        Offerte_ContentControls.Item("EmpfängerOrt").Range.text = text
    Case Else
    End Select
End Sub

Sub editBox_GetText(control As IRibbonControl, ByRef returnedVal)
    Select Case control.id
        Case "editEmpfängerVorname"
            returnedVal = Offerte_ContentControls.Item("EmpfängerVorname").Range.text
        Case "editEmpfängerNachname"
            returnedVal = Offerte_ContentControls.Item("EmpfängerNachname").Range.text
        Case "editEmpfängerStraße"
            returnedVal = Offerte_ContentControls.Item("EmpfängerStraße").Range.text
        Case "editEmpfängerPLZ"
            returnedVal = Offerte_ContentControls.Item("EmpfängerPLZ").Range.text
        Case "editEmpfängerOrt"
            returnedVal = Offerte_ContentControls.Item("EmpfängerOrt").Range.text
        Case Else
        End Select
    End Sub

```

### Das Steuerelement *comboBox*

Ein Kombinationsfeld ist seinem Namen gerecht: Es kombiniert ein Bearbeitungsfeld mit einem Dropdownfeld. Im Gegensatz zu einem Dropdownfeld darf der Benutzer auch beliebigen Text eingeben, anstatt lediglich einen Eintrag aus der Liste zu wählen.



Das comboBox-Steuerelement vereint aus diesem Grund viele Attribute der Steuerelemente dropdown und gallery sowie editBox. In diesem Beispiel wird keines der Listen-Attribute eingesetzt, nur solche, die unter editBox bereits vorgestellt wurden. (Das Steuerelement dropdown ist nicht Teil dieses Beispiels; die Attribute werden unter gallery näher erläutert.)

```

<comboBox id="comboAnrede" sizeString="Die Anrede auswählen / eingeben"
    screentip="Die Anrede auswählen" supertip="Sie können die Anrede aus der Liste wählen,
    oder selber eine eingeben. Der Nachname des Empfängers wird automatisch hinzugefügt."
    onChange="comboAnrede_Select" getText="comboAnrede_GetText"
    getVisible="comboAnrede_GetVisible" keytip="W" >
    <item id="Anrede1" label="Sehr geehrter Herr" />
    <item id="Anrede2" label="Sehr geehrte Frau" />
    <item id="Anrede3" label="Sehr geehrter Herr Dr." />
    <item id="Anrede4" label="Sehr geehrte Frau Dr." />
</comboBox>

```

Die Einträge der Liste sind in diesem Beispiel fester Bestandteil des Ribbon-XML und werden durch das Element item festgelegt. item hat die üblichen Attribute id, image, imageMso, label, screentip sowie supertip.

Der mit diesem Steuerelement verbundene VBA-Code ist in Listing 17.11 ersichtlich. Das Kombinationsfeld ist bei der Einblendung der Gruppe *Empfängeradresse* nicht eingeblendet. Die Sichtbarkeit wird nach dem bekannten Muster über die Attribute getPressed sowie pressed (die Prozeduren

*cbAnrede\_GetPressed* bzw. *cbAnrede\_Pressed*) des Kontrollkästchens *Anrede* sowie über das Attribut *getVisible* des Kombinationsfelds geregelt.

Wie beim Bearbeitungsfeld wird der im Feld angezeigte Text über das Attribut *getText* gesteuert und die Benutzerauswahl über das Attribut *onChange*. Diese rufen die Prozeduren *comboAnrede\_GetText* bzw. *comboAnrede\_Select* auf. Auch hier dient ein Inhaltssteuerelement als Zielbereich.

**Listing 17.11** Die vom Steuerelement *comboBox* aufgerufenen Prozeduren

```
Private dropdownAnredeVisible As Boolean
Sub cbAnrede_GetPressed(control As IRibbonControl, ByRef returnedVal)
    returnedVal = dropdownAnredeVisible
End Sub

Sub cbAnrede_Pressed(control As IRibbonControl, pressed As Boolean)
    If pressed = True Then
        dropdownAnredeVisible = True
    Else
        dropdownAnredeVisible = False
    End If
    thisRibbon.InvalidateControl control.id
    thisRibbon.InvalidateControl "comboAnrede"
    thisRibbon.InvalidateControl "labelComboAnrede"
End Sub

Sub comboAnrede_GetVisible(control As IRibbonControl, ByRef returnedVal)
    returnedVal = dropdownAnredeVisible
End Sub

Sub comboAnrede_Select(control As IRibbonControl, text As String)
    Offerte_ContentControls.Item("BriefAnrede").Range.text = _
        text & " " & Offerte_ContentControls.Item("EmpfängerNachname").Range.text
End Sub

Sub comboAnrede_GetText(control As IRibbonControl, ByRef returnedVal)
    returnedVal = ""
End Sub
```

### Das Steuerelement *labelControl*

Über dem Kombinationsfeld befindet sich eine Beschriftung. Beschriftungen sind kein besonders spannendes Thema, aber dennoch nützlich, wenn sie nicht am vorgesehenen Ort (in diesem Fall links neben dem Steuerelement) erscheinen sollen. Auffallend im Beispiel sind die Leerzeichen im Wert des *label*-Attributs: Sie sorgen dafür, dass die Beschriftung mehr oder weniger mit dem darunter befindlichen Kombinationsfeld links ausgerichtet ist (sonst erscheint sie zu weit nach links).

```
<labelControl id="labelComboAnrede" label="  Anrede auswählen oder eingeben"
    getVisible="comboAnrede_GetVisible" />
```

In diesem Beispiel teilt das Steuerelement das *getVisible*-Callback des Kombinationsfelds und wird folglich mit ihm ein- und ausgeblendet.



### Das Steuerelement *gallery*

Was der Beschriftung an Spannung mangelt, weist dafür der Katalog auf. Eigentlich ist er in vielen Belangen ein bebildertes Dropdownfeld, was man ihm aber nicht sofort ansieht. Da das Dropdownfeld bislang nicht behandelt wurde, werden an dieser Stelle einige noch nicht vorgestellte Attribute erläutert. Alle außer den Attributen `columns`, `rows` sowie diejenigen, die die Höhe und Breite eines Eintrags festlegen, hat das Steuerelement `dropDown` mit dem `gallery` gemeinsam.

- **columns:** Eine statische Zahl, die die Anzahl der gewünschten Spalten festlegt
- **rows:** Eine statische Zahl, die die Anzahl der gewünschten Zeilen festlegt
- **itemHeight:** Legt die Höhe eines Eintrags in Pixel fest
- **itemWidth:** Legt die Breite eines Eintrags in Pixel fest
- **getItemCount:** Holt die Anzahl der im Katalog anzuzeigenden Einträge
- **getItemHeight:** Holt die gewünschte Höhe für einen Eintrag
- **getItemWidth:** Holt die gewünschte Breite für einen Eintrag
- **getItemId:** Holt den ID-Wert eines einzelnen Eintrags
- **getItemImage:** Holt die Grafik eines einzelnen Eintrags
- **getItemLabel:** Holt die Beschriftung eines einzelnen Eintrags
- **getItemScreenTip:** Holt den obersten Teil der QuickInfo eines einzelnen Eintrags
- **getItemSupertip:** Holt den unteren Teil der QuickInfo eines einzelnen Eintrags
- **getSelectedItemId:** Holt den Wert des `id`-Attributs des vom Benutzer gewählten Eintrags
- **getSelectedItemIndex:** Holt den Index-Wert des vom Benutzer gewählten Eintrags

Das Beispiel setzt einige dieser Attribute ein:

```
<gallery id="galleryProdukte" label="Produktliste" onAction="Gallery_OnSelect"
  getItemImage="Gallery_GetItemImage" getItemLabel="Gallery_GetItemLabel"
  getItemScreenTip="Gallery_GetItemScreenTip" getItemSupertip="Gallery_GetItemSupertip"
  getItemCount="Gallery_GetItemCount" getItemID="Gallery_GetItemID" columns="3"
  itemHeight="50" itemWidth="50" keytip="K" />
```

Die Callback-Prozeduren sind in Listing 17.12 ersichtlich. Die Daten für die Produktliste befinden sich in einer XML-Datei. Von besonderem Interesse ist die Prozedur *Gallery\_GetItemImage*, da sie statt einer Zeichenkette eine Grafik an die Ribbon-Erweiterung zurückgibt. Die Grafik muss als *stdOLE-Picture* vorliegen – nicht etwas, das die meisten Office-Entwickler tagtäglich einsetzen.

**Listing 17.12** Vom Steuerelement *gallery* aufgerufene Prozeduren

```
Sub Gallery_GetItemCount(control As IRibbonControl, ByRef count)
  Dim xmlDoc As MSXML2.DOMDocument
  Dim xmlProduktNodes As MSXML2.IXMLDOMNodeList

  Set xmlDoc = New MSXML2.DOMDocument
  xmlDoc.async = False
  xmlDoc.Load appPfad & "Produkte.xml"
  Set xmlProduktNodes = xmlDoc.SelectNodes("/Produkte/Produkt")
  count = xmlProduktNodes.Length
End Sub
```

**Listing 17.12** Vom Steuerelement *gallery* aufgerufene Prozeduren (Fortsetzung)

```

    Set xmlDoc = Nothing
End Sub

Sub Gallery_GetItemID(control As IRibbonControl, index As Integer, ByRef id)
    Dim lIndex As Long
    lIndex = index
    id = ProduktInfoAusXMLHolen(lIndex, "artikelNr")
End Sub

Sub Gallery_GetItemLabel(control As IRibbonControl, index As Integer, ByRef label)
    Dim lIndex As Long
    lIndex = index
    label = ProduktInfoAusXMLHolen(lIndex, "bezeichnung")
End Sub

Sub Gallery_GetItemScreentip(control As IRibbonControl, index As Integer, ByRef screen)
    Dim lIndex As Long
    lIndex = index
    screen = ProduktInfoAusXMLHolen(lIndex, "bezeichnung")
End Sub

Sub Gallery_GetItemSupertip(control As IRibbonControl, index As Integer, ByRef screen)
    Dim lIndex As Long
    lIndex = index
    screen = ProduktInfoAusXMLHolen(lIndex, "beschreibung")
End Sub

Sub Gallery_GetItemImage(control As IRibbonControl, index As Integer, ByRef image)
    Dim lIndex As Long
    Dim pic As stdole.StdPicture

    lIndex = index
    image = ProduktInfoAusXMLHolen(lIndex, "bild")
    Set pic = stdole.StdFunctions.LoadPicture(appPfad & image)
    Set image = pic
End Sub

Private Function ProduktInfoAusXMLHolen(ProduktIndex As Long, _
    ProduktName As String) As String

    Dim info As String
    Dim xmlDoc As MSXML2.DOMDocument
    Dim xmlProduktNodes As MSXML2.IXMLDOMNodeList
    Dim xmlProdukt As MSXML2.IXMLDOMNode

    Set xmlDoc = New MSXML2.DOMDocument
    xmlDoc.async = False
    xmlDoc.Load appPfad & "Produkte.xml"
    Set xmlProduktNodes = xmlDoc.SelectNodes("/Produkte/Produkt")
    Set xmlProduktNodes = xmlDoc.SelectNodes("/Produkte/Produkt")
    Set xmlProdukt = xmlProduktNodes(ProduktIndex)
    info = xmlProdukt.Attributes.getNamedItem(ProduktName).text

    Set xmlDoc = Nothing
    ProduktInfoAusXMLHolen = info
End Function

```



Listing 17.12 Vom Steuerelement *gallery* aufgerufene Prozeduren (Fortsetzung)

```

End Function

Sub Gallery_OnSelect(control As IRibbonControl, selectedID As String, selectedIndex As Integer)
    Dim rng As Word.Range
    Dim tbl As Word.Table
    Dim rw As Word.Row
    Dim beschreibung As String
    Dim preis As String
    Dim rngLetzteZelle As Word.Range

    Set rng = Offerte_ContentControls("Produktliste").Range
    If rng.Tables.count < 1 Then
        Set tbl = ProdukttabelleErstellen(rng)
        If tbl Is Nothing Then
            MsgBox "Die Tabelle konnte nicht erstellt werden."
            Exit Sub
        End If
        rng.MoveEnd Unit:=wdCharacter, count:=1
        rng.Delete
    Else
        Set tbl = rng.Tables(1)
    End If
    Set rw = tbl.Rows.Add(BeforeRow:=tbl.Rows(tbl.Rows.count))

    If ProduktAngabenHolen(selectedID, beschreibung, preis) Then
        rw.Cells(1).Range.text = selectedID
        rw.Cells(2).Range.text = beschreibung
        rw.Cells(3).Range.text = "0"
        rw.Cells(4).Range.text = preis
        Set rngLetzteZelle = rw.Cells(5).Range
        rngLetzteZelle.Collapse wdCollapseStart
        rngLetzteZelle.Fields.Add Range:=rngLetzteZelle, text:="=Product(Left) \# " _
            & Chr$(34) & "0,00" & Chr$(34)
        tbl.Range.Fields.Update
    Else
        MsgBox "Produkt konnte nicht eingefügt werden."
    End If
End Sub

Private Function ProduktAngabenHolen(ByVal selectedID As String, _
    ByRef beschreibung As String, ByRef preis As String) As Boolean

    Dim erfolg As Boolean
    Dim xmlDoc As MSXML2.DOMDocument
    Dim xmlProduktNodes As MSXML2.IXMLDOMNodeList
    Dim xmlProdukt As MSXML2.IXMLDOMNode
    Dim id As String

    erfolg = False
    Set xmlDoc = New MSXML2.DOMDocument
    xmlDoc.async = False
    xmlDoc.Load appPfad & "Produkte.xml"
    Set xmlProduktNodes = xmlDoc.SelectNodes("/Produkte/Produkt")
    Set xmlProduktNodes = xmlDoc.SelectNodes("/Produkte/Produkt")

```

**Listing 17.12** Vom Steuerelement *gallery* aufgerufene Prozeduren (Fortsetzung)

```

For Each xmlProdukt In xmlProduktNodes
    id = xmlProdukt.Attributes.getNamedItem("artikelNr").text
    If id = selectedID Then
        beschreibung = xmlProdukt.Attributes.getNamedItem("beschreibung").text
        preis = xmlProdukt.Attributes.getNamedItem("preis").text
        erfolg = True
        Exit For
    End If
Next

Set xmlDoc = Nothing
ProduktAngabenHolen = erfolg
End Function

Private Function ProdukttabelleErstellen(rng As Word.Range) As Word.Table
    'Der Code wurde aus Platzgründen weggelassen.
End Function

```



Die Beispieldateien *Ribbon\_Beispiel2.dotm*, *Ribbon\_Beispiel3.dotm*, *Ribbon\_Beispiel4.dotm* und *Produkte.xml* sowie die zugehörigen neun Grafikdateien finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap17*. Vertrauen Sie diesen Ordnern unter *Word Optionen/Vertrauensstellencenter/Einstellungen für das Vertrauensstellencenter/Vertrauenswürdige Speicherorte/Neuen Speicherort hinzufügen*, wie in Kapitel 14 beschrieben.

Die folgenden Steuerelemente und Attribute wurden im Add-In-Beispiel des Abschnitts zum Thema VSTO-Add-Ins in Kapitel 10 gebraucht. Sie werden in einer dokumentenzentrischen Lösung auf gleiche Art eingesetzt.

### loadImage

Bislang haben wir zwei Möglichkeiten vorgestellt, um Symbole für eine Schaltfläche einzubinden: Die Grafik wird (mit Hilfe des Custom UI Editors) in das Dokument importiert oder sie wird durch die Callback-Prozedur eines getImage-Attributs dynamisch geladen. Es gibt eine dritte Vorgehensweise: das loadImage-Attribut des ribbon-Elements.

```

<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui" onLoad="OnLoad"
loadImage="ribbon_getImages">

```

Die mit dem Attribut verbundene Callback-Prozedur wird nur einmal, beim Laden der Ribbon-Erweiterung, ausgeführt. Die Funktion erhält über die Parameter *imageID* von jedem Steuerelement der Ribbon-Erweiterung, das ein *image*-Attribut enthält, dessen Wert (der Name der Grafik). Die Grafikdatei wird in den Speicher geladen und in der Form einer *stdole IPictureDisp* an die Ribbon-Erweiterung zurückgegeben. Diese Methode ist effizienter als die mehrmalige Ausführung von getImage-Prozeduren, vorausgesetzt, die Symbole müssen während der Word-Sitzung nicht dynamisch geändert werden.

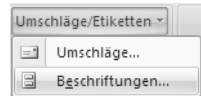
Die VBA-Version des Codes befindet sich in Listing 17.13.

**Listing 17.13** Eine von Visual Studio geladene Grafik in einen *stdole IpictureDisp*-Datentyp umwandeln

```
Public Sub ribbon_getImages(ByVal imageID As String, ByRef image)
    Dim appPfad As String
    appPfad = ActiveDocument.Path & "\"
    Set image = stdole.StdFunctions.LoadPicture(appPfad & imageID)
End Sub
```

### dynamicMenu

Wie bereits erwähnt, muss eine Multifunktionsleiste vollständig definiert sein, bevor sie von einer Anwendung geladen wird. Mittels der mit dem XML verbundenen Callback-Prozeduren können Steuerelemente ein- und ausgeblendet werden. Es ist jedoch nicht möglich, zusätzliche zu erstellen oder vorhandene zu löschen. Die einzige Ausnahme bildet das Steuerelement *dynamicMenu*. Über die mit dem Attribut *getContent* verbundene Prozedur, wie der VBA-Beispielcode in Listing 17.14, ist es möglich, nach Bedarf ein Menü dynamisch zu definieren. Das XML wird genau gleich aufgebaut wie in der XML-Datei für das Ribbon. Achten Sie darauf, dass der Namensraum des Ribbon-XML im Element *menu* enthalten sein muss.

**Listing 17.14** Die mit dem Attribut *getContent* verbundene Prozedur, um ein Menü dynamisch zu definieren.

```
Public Sub menu_GetContent(ByVal control As Office.IribbonControl, ByRef menuString)
    Dim menuXML As String = "", q As String = Chr(34)
    menuXML = "<menu xmlns=" & q & "http://schemas.microsoft.com/office/2006/01/customui" _
        & q & "><button idMso=" & q & "EnvelopesAndLabelsDialog" & q & " /><button idMso=" _
        & q & "LabelsDialog" & q & " /></menu>"
    menuString = menuXML
End Sub
```

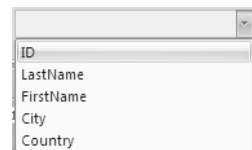


Der Beispielcode für *loadImage* und *dynamicMenu* befindet sich in der Datei *Ribbon\_Beiispiel5.docm* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap17*.

### dropDown

Die Steuerelemente *comboBox*, *editBox* und *dropDown* haben viele Attribute gemeinsam. Zusätzlich dazu stellt *dropDown* die folgenden Attribute zur Verfügung:

- **getSelectedItemID:** Die aufgerufene Prozedur legt bei der Initialisierung anhand des ID-Wertes (Zeichenkette) fest, welcher Eintrag in der Liste vorab zu markieren ist. Benutzen Sie entweder dieses Attribut oder *getSelectedItemIndex*, aber nicht beide zusammen.
- **getSelectedItemIndex:** Die aufgerufene Prozedur legt bei der Initialisierung anhand des Index-Wertes (Ganzzahl) fest, welcher Eintrag in der Liste vorab zu markieren ist. Benutzen Sie entweder dieses Attribut oder *getSelectedItemID*, aber nicht beide zusammen.
- **showItemLabel:** Pendant zum Attribut *showLabel*. Erlaubt die dynamische Festlegung der Sichtbarkeit der Beschriftungen in der Liste.



```
<dropDown id="dropdownMergeFeldListe" onAction="MergeFieldName_Select"
getItemCount="dropDownFeldListe_getItemCount"
getItemID="dropDownFeldListe_getItemLabel"
getItemLabel="dropDownFeldListe_getItemLabel"
getEnabled="sdAktiviert_getEnabled"
sizeString="Das Seriendruckfeld auswählen:" />
```

## Zusammenfassung

In diesem Kapitel wurde die neue Office 2007 Ribbon-Erweiterung vorgestellt. Da den Autoren eine begrenzte Anzahl Seiten zur Verfügung steht, war es nicht möglich, auf alle Einzelheiten der Technologie einzugehen. Diese Angaben werden es Ihnen ermöglichen, die im Internet zusätzlich zur Verfügung stehenden Informationen einzusetzen, um eigene Ribbon-Lösungen erfolgreich zu realisieren.

In diesem Kapitel wurden die folgenden Themen behandelt:

- In einer kurzen Übersicht (Seite 724 ff.) wurden die Unterschiede zu Symbolleisten sowie Beweggründe für die Änderung diskutiert. Auch wurde die Rolle von Symbolleisten in Word 2007 erwähnt.
- Die einzelnen Schritte, um eine Ribbon-Erweiterung zu erstellen und sie in einem Word-Dokument einzubinden, wurden anhand eines einfachen Beispiels ab Seite 728 erläutert. Einige Werkzeuge für die Arbeit mit XML wurden vorgestellt.
- Der letzte Teil (Seite 733 ff.) ging mehr ins Detail. Neben Steuerelementen und ihren Attributen wurden erweiterte Möglichkeiten erläutert.

### HINWEIS

Nachfolgend sind noch einige Ressourcen im Internet zum Thema »Ribbon« aufgeführt:

- Jens Häupels Blog: <http://blogs.msdn.com/jensha/search.aspx?q=Ribbon&p=1>
- Die Einstiegseite für das Thema auf MSDN (englisch): <http://msdn2.microsoft.com/en-us/office/aa905530.aspx>
- Drei MSDN-Artikel von Frank Rice und Ken Getz (englisch):
  - <http://msdn2.microsoft.com/en-us/library/ms406046.aspx>
  - <http://msdn2.microsoft.com/en-us/library/aa338199.aspx>
  - <http://msdn2.microsoft.com/en-us/library/aa722523.aspx>
- Webseite von Patrick Schmid: <http://www.pschrnid.net>

## Kapitel 18

# Tastaturbelegungen

### In diesem Kapitel:

Tastenbelegungen im Objektmodell	763
Tastenkombinationen verwalten	771
Tastenbelegung mit COM-Add-In verbinden	773
Zusammenfassung	777

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 in Word bereits eine große Anzahl von vordefinierten Tastenkombinationen enthalten sind. Sie finden ausführliche Listen in der Hilfe unter dem Begriff »Tastenkombination«. Diese sind jedoch nicht in 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 18.1 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 bereits zugewiesen ist. Durch Betätigung der Schaltfläche *Zuordnen* wird die Belegung bestätigt oder Sie können eine andere Kombination eingeben.

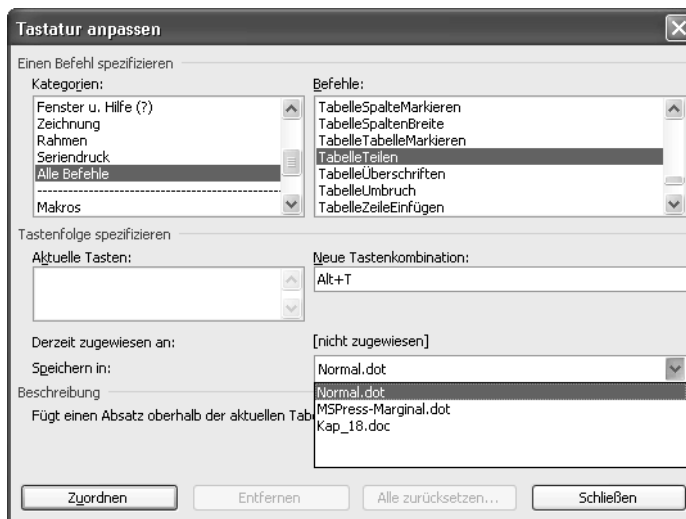
Das Dialogfeld ist auch in Word 2007 enthalten. Sie erreichen es über die Kategorie *Anpassen* im Dialogfeld *Word-Optionen*. Klicken Sie dort auf die Schaltfläche *Anpassen*.

Auch hier ist es wichtig, den *Kontext* festzulegen. Wurden in mehreren Kontexten die gleiche Tastenkombination verschiedenen Befehlen zugeordnet, gelten die in Kapitel 14 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. 18.1 Fast jedem Befehl, Makro, Formatvorlage, Schriftart oder AutoText-Eintrag 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> 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

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.

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 18.1.

**Command** spezifiziert, was zu machen ist, beispielsweise der Befehl-, Makro- oder Formatvorlagenname (siehe auch Tabelle 18.1). 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-ROM zum Buch befindet sich im Ordner *\Beilagen\Interne Word-Befehle* die Datei *Interne WordBefehle.doc* mit einer Tabelle, die alle englischen und deutschen Befehlsnamen gegenüberstellt.

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 18.1.) Benutzerdefinierte Prozeduren, die Parameter verlangen, stehen Tastenbelegungen nicht zur Verfügung.

Zudem befindet sich eine Spalte »WordBasic.[KeyMacro\$]()« in Tabelle 18.1. Diese Funktion liefert für einige KeyCategorys den deutschen Befehlsnamen eines Word-internen Befehls, der einer benutzerdefinierten Tastenkombination zugewiesen wurde. Diese Funktion verwendet zwei Parameter:

- Index: Der numerische Index-Wert des *benutzerdefinierten* KeyBinding
- Context: 0 für die *Normal.dot*; 1 für die angehängte Dokumentvorlage des aktuellen Dokuments

WordBasic.KeyMacro\$(1, 0) 'Beispiel-Ergebnis: Fett statt Bold

**Tabelle 18.1** Beispiele für die Eigenschaften *KeyCategory*, *Command* und *CommandParameter*

KeyCategory	Command	Command-Parameter	WordBasic. [KeyMacro\$]()
wdKeyCategoryDisable (gesperrt) 0	[Leer]	[Leer]	""
wdKeyCategoryCommand (Befehl) 1	"ParaKeepWithNext"	[Leer]	"AbsätzeNichtTrennen"
wdKeyCategoryCommand (erweiterter Befehl) 1	"Farbe"	"12"	"Violet"
wdKeyCategoryCommand (erweiterter Befehl) 1	"DateiDateiÖffnen"	"C:\My Documents\test.doc"	"Open test.doc"
wdKeyCategoryCommand (Zeichen in der angegebenen Symbol-Schriftart) 1	"Symbol"	"?ZapfDingbats BT"	"ZapfDingbats BT: 61512"
wdKeyCategoryMacro (Makro) 2	"Normal.NewMacros. Makro1"	[Leer]	"Normal.NewMacros. Macro1"
wdKeyCategoryFont (Schriftart) 3	"Arial Narrow"	[Leer]	"Arial Narrow"
wdKeyCategoryAutotext (AutoText) 4	"Seite X von Y"	[Leer]	"Seite X von Y"
wdKeyCategoryStyle (Formatvorlage) 5	"Umschlagadresse"	[Leer]	"Umschlagadresse Formatvorlage"
wdKeyCategorySymbol (Symbol in normal Text) 6	"1/4"	[Leer]	""
wdKeyCateogryPrefix 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«.</p> <p>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 18.1 Beispiele für die Eigenschaften *KeyCategory*, *Command* und *CommandParameter* (Fortsetzung)

KeyCategory	Command	Command-Parameter	WordBasic. [KeyMacro\$]()
wdKeyCategoryNil -1	<p>Diese <b>KeyCategory</b>-Eigenschaft gibt <b>wdKeyCategoryNil</b> (-1) zurück, wenn die Tastenkombination im fraglichen Kontext nicht zugewiesen wurde. Beispiel:</p> <p>Die Kombination <span>Strg</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.</p>		
<p>Zum »?« in der Spalte »CommandParameter«: Wenn das erste Zeichen der <b>CommandParameter</b>-Eigenschaft eines »Symbol«-<b>Command</b> in der angegebenen Schriftart im VBA-Editor nicht wiedergegeben werden kann, weil es sich um ein 2-Byte-Unicode-Zeichen handelt, das mit der Funktion <b>AscW</b> ermittelt wurde, gibt VBA ein Fragezeichen zurück. Das Gleiche gilt für das Ergebnis der <b>Command</b>-Eigenschaft des <b>KeyCategory</b> <b>wdKeyCategorySymbol</b>.</p>			

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

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
```

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
```

Optimierung der  
Benutzerschnittstelle

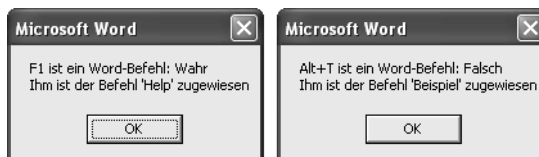
Tastaturkombinationen können auch mit zwei aufeinander folgenden Tastenfolgen definiert werden. In Word sind für internationale Zeichen einige Tastenkombinationen vordefiniert, wie `[Strg]+[']`, `[U]`, mit der das Zeichen »ú« eingefügt wird. Dann haben die Eigenschaften wie `FindKey` folgende Syntax:

```
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 18.1 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.

Das Resultat der Prozedur *TestWordBefehl* in Listing 18.1 ist in Abbildung 18.2 ersichtlich. Die Zeichenkette mit der Tastenkombination wurde durch die Eigenschaft **KeyString** des `KeyBinding`-Objekts erzeugt.

**Abbildg. 18.2** Zwischen einer benutzerdefinierten Tastenbelegung und einem Word-Befehl unterscheiden



**Listing 18.1** 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
```

**Listing 18.1** Feststellen, ob eine Tastenbelegung mit einem benutzerdefinierten oder Word-internen Befehl verbunden ist (Fortsetzung)

```
If kb Is Nothing Then
    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 entsprechen der nebenstehend gezeigten Abbildung. Klicken Sie damit auf den Befehl, der Sie interessiert. Das Dialogfeld *Tastatur anpassen* erscheint mit allen Belegungen.



Die Beispieldatei *Bsp18\_01.dot* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap18*.


Das Listing 18.2 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 18.3 ersichtlich. Vergleichen Sie die Spalte »Kat.« mit den Angaben in Spalte »KeyCategory« der Tabelle 18.1.

## HINWEIS

Bitte beachten Sie, dass 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. 18.3** 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	prjBsp18_01.Bsp18_01.TestMakro		Falsch
Alt+N	1	RemoveBullets Numbers		Falsch
Alt+O	1	FileOpenFile	C:\WordBuch\Kap18\TestOeffnen.doc	Falsch
Alt+P	6	¶		Falsch
Alt+S, P	1	ToolsCustomize		Falsch
Alt+Strg+!	1	OpenUpPara		Falsch
Alt+Strg+Umschalt+F1	1	OpenUpPara		Falsch
Alt+T	4	Beispiel		Falsch
Strg+Umschalt+A, P	2	prjBsp18_01.Bsp18_01.AlleTastaturBelegungenAuflisten		Falsch

Die Prozedur *AlleTastenBelegungenAuflisten* schleift zuerst durch alle wdKey-Konstantwerte, 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 18.2 enthalten).

Am Schluss wird die Zeichenkette sBefehlsliste in das aktuelle Dokument eingefügt und in eine Tabelle umgewandelt. Diese wird nach den Spalten »Word-Befehl«, dann »Tastaturk.« sortiert (die benutzerdefinierten Tastenbelegungen stehen damit am Tabellenanfang).

**Listing 18.2** 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
```

Listing 18.2 Alle Tastenbelegungen eines Kontextes auflisten (Fortsetzung)

```

        index = index + 1
        sBefehlsListe = sBefehlsListe & vbCr &
            BefehlslisteZusammenstellen2(1KeyArg1, 1KeyArg2, 1KeyArg)
    End If
Next 1KeyArg
'Code, um Tastenbelegungen mit zwei und mehr Befehlstasten zu prüfen,
'folgen an dieser Stelle in der vollständiger Prozedur auf der CD
    Next 1Counter
Next 1KeyArg1

'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 1Arg1 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(1Arg1))
    On Error GoTo 0
    If kb Is Nothing Then
        bWordBefehl = True
        Set kb = Application.FindKey(1Arg1)
    Else
        bWordBefehl = False
    End If

    s = Beschreibung(kb, bWordBefehl)
    BefehlslisteZusammenstellen1 = s
End Function

Function BefehlslisteZusammenstellen2(ByVal 1Arg1 As Long, ByVal 1Arg2 As Long, _
    ByVal 1Arg 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(1Arg1, 1Arg2), 1Arg)
    On Error GoTo 0
    If kb Is Nothing Then
        bWordBefehl = True
        Set kb = Application.FindKey(BuildKeyCode(1Arg1, 1Arg2), 1Arg)
    Else

```

**Listing 18.2** Alle Tastenbelegungen eines Kontextes auflisten (*Fortsetzung*)

```

    bWordBefehl = False
End If

s = Beschreibung(kb, bWordBefehl)
BefehlslisteZusammenstellen2 = s
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 *Bsp18\_02.dot* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap18*.

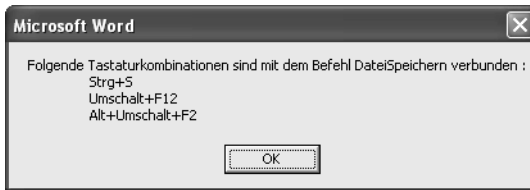
## Tastenbelegungen eines Befehls ermitteln

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-Eigenschaften zu prüfen. Dies wäre jedoch kompliziert und zudem relativ langsam.

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 18.3 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. 18.4 Die Tastaturbelegungen für den Befehl *DateiSpeichern*Listing 18.3 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 & vbCr & vbTab & kb.KeyString
    Next
    MsgBox sList
End Sub

```



Die Beispieldatei *Bsp18\_03.dot* finden Sie auf der CD-ROM zum Buch im Ordner `\Beispiele\Kap18`.

## Tastenkombinationen verwalten

Nachdem nun die Grundlagen bekannt sind, schauen wir uns die Befehle an, um Tastenbelegungen zu verwalten.

### Tastenbelegungen entfernen

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

```





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
```

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 Word-Befehl, wird diese im Menü erscheinen anstelle einer gesperrten. Im obigen Beispiel erscheint beispielsweise  +  anstelle von  + .



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

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 gültige `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.

Das folgende Codefragment zeigt, wie ein Satz von Tastenkombinationen erstellt wird, um eine Gruppe ähnlicher `AutoText`-Einträge einzufügen. Jede Kombination beginnt mit  + , gefolgt von einer weiteren 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 der 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>.

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 Formattvorlage »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 öffentlichen 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 18.4 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 angegeben. 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 COM-Add-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 18.4** 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 18.5 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 erfahren Sie in Kapitel 10.

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 18.5** Code im COM-Add-In

```
'Code in COM-Add-In Modul
Dim wdApp As Word.Application

Private Sub AddinInstance_OnConnection(ByVal Application As Object, _
    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
```

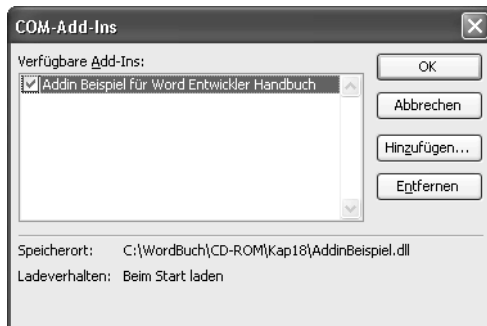
## Das Beispiel installieren

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* (auf der Buch-CD im Ordner *\Beispiele\Kap18*) darf sich in einem beliebigen Ordner befinden.
4. Das COM-Add-In muss auf dem Rechner registriert werden.
  - Rufen Sie den Befehl *Ausführen* im *Start*-Menü von Windows XP auf (unter Windows Vista drücken Sie dazu die Tastenkombination  + **R**).
  - Geben Sie die Befehlszeile ein:  
`regsvr32 "[Laufwerk]\[Pfadangabe]\AddinBeispiel.dll"`  
und führen Sie sie aus.
5. 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.
6. Führen Sie den Menübefehl *COM-Add-Ins* aus; das Dialogfeld in Abbildung 18.5 erscheint.

Abbildg. 18.5

COM-Add-Ins verwalten



7. Klicken Sie auf *Hinzufügen* und navigieren Sie zum Ordner, in dem sich das COM-Add-In befindet.
8. Wählen Sie dieses aus und bestätigen Sie das Dialogfeld mit *OK*. Es wird registriert und aktiviert.
9. Schließen Sie Word und starten Sie die Anwendung erneut.
10. Drücken Sie **Alt** + **L**. Wenn alles korrekt läuft, sollte das folgende Meldungsfeld erscheinen.

Abbildg. 18.6 Die Tastenkombination blendet dieses Meldungsfeld ein



#### HINWEIS

Falls Sie die DLL-Datei in einen anderen Ordner verschieben, müssen Sie die Schritte 4, und 6 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\Kap18*. 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. Damit werden die Referenzen zur Word-Version nicht angepasst werden müssen.

## Zusammenfassung

Dieses Kapitel befasst sich mit der programmtechnischen Verwaltung und Erstellung von Tastenkombinationen. Die folgenden Schwerpunkte wurden veranschaulicht:

- Wie werden Tastenbelegungen im Objektmodell verwaltet (Seite 763)?
- Wie werden die Tastenkombinationen eines Kontextes ermittelt (Seite 765)?
- Wie lässt sich herausfinden, welche Tastenkombinationen einem bestimmten Befehl zugewiesen sind (Seite 770)?
- Wie werden Tastenbelegungen entfernt (Seite 771)?
- Wie werden neue Tastenkombinationen hinzugefügt (Seite 772)?
- Wie kann eine Prozedur in einem COM-Add-In einer Tastenkombination in Word zugewiesen werden (Seite 773)?



## Kapitel 19

# Aufgabenbereiche

### In diesem Kapitel:

Allgemeines zum Aufgabenbereich oder, was steckt dahinter	780
Der programmtechnische Umgang mit Aufgabenbereichen (Task Panes)	784
Zusammenfassung	806

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 Word 2002 und früher lautet die Antwort, »Nein«. Für Word 2003 gibt es Dokument-spezifische Lösungen über die Smart Document-Funktionalität (Kapitel 24), sowie über VSTO 2005 (Kapitel 10). Die CustomTaskPane-Schnittstelle wurde in Office 2007 eingeführt, steht VBA-Entwickler jedoch nicht zur Verfügung. Das Add-In-Werkzeug VSTO 2005 SE bindet diese Schnittstelle ein und wird in Kapitel 10 kurz vorgestellt.




Was den ersten Teil dieser Frage angeht – Word-eigene Arbeitsbereiche den eigenen Bedürfnissen anzupassen – stehen uns nur beschränkte Möglichkeiten zur Verfügung. Ein australischer Kollege, Steve Hudson, hat sie in Word 2003 eingehend erforscht und uns erfreulicherweise die Erlaubnis gegeben, seine Erkenntnisse für dieses Buch zu übersetzen. Die hier aufgeführten Angaben gelten primär für Word 2003. Die Aufgabenbereiche, die in Word 2007 noch vorhanden sind, können wie hier beschrieben weiterhin angesprochen werden.

#### 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 im 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, 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 20).

#### HINWEIS

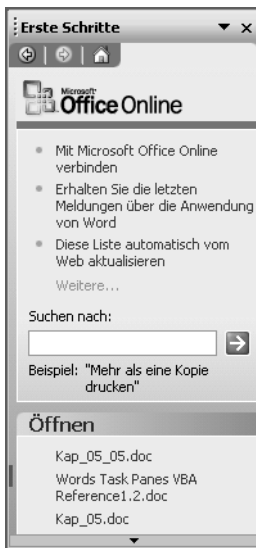
*CommandBars* (Symbolleisten) wurden in Kapitel 16 dargestellt.

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* (12.0 ist für Word 2007, 11.0 für Word 2003; 10.0 für Word 2002; 9.0 für Word 2000) verwaltet.



Die Ähnlichkeit zu einer üblichen Symbolleiste ist in Abbildung 19.1 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 in Word 2003 allerdings seiner Größe wegen kaum wünschenswert wäre).

Abbildg. 19.1 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 19.1).

## Objektmodell-Schnittstellen für Work Panes

Die Objekte der Tabelle 19.1 und einige ihrer Eigenschaften sind, wenn Sie das Kapitel 16 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 19.1 Eigenschaften des Work Pane-Objekts (eine Symbolleiste)

Objekt	Eigenschaft/Methode	Datentyp
Application	ShowStartupDialog	Boolean
CommandBars("Task Pane")	Height	Long

**Tabelle 19.1** 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 Aufgabenbereichs – 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 19.2

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 19.2** 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 2 diskutiert, ist es oft besser, vorausschbaren Fehler auszuweichen, anstatt sie abzufangen. Die Tests in Tabelle 19.3 sind nützlich, um festzustellen, ob die Umgebung gewisse Aufgabenbereiche unterstützt.

**Tabelle 19.3** 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 direkt angesprochen werden können.

Die Tabelle 19.4 listet die Aufgabenbereiche auf mit ihren `WdTaskPanes`-Typen, deren Werten, der ID-Nummer des `CommandBar.Controls`, in welchen Word-Versionen der Aufgabenbereich vorhanden ist, 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 19.4** Die von Word zur Verfügung gestellten Aufgabenbereiche mit Informationen für den programmtechnischen Umgang

Beschriftung	WdTaskPanes-Enumerator	Wert	Command-Bar.Control ID	Gültig für	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, 2003, 2007	ClipArt einfügen
<i>Neues Dokument</i>	[kein]	[kein]	18	Word 2002, 2003 (In 2007 blendet dieser Befehl das Dialogfeld <i>Neues Dokument</i> ein.)	Bietet Listen von Dokumentarten, Speicherorte für Vorlagen, und der zuletzt benutzten Vorlagen an
<i>Zwischenablage</i>	[kein]	[kein]	809	Word 2002, 2003, 2007	Enthält die Einträge in der Office-Zwischenablage. Ein programmmäßiger Zugriff auf den Inhalt dieser Zwischenablage steht im Word-Objektmodell nicht zur Verfügung.
<i>Dokumentaktionen</i>	wdTaskPane-Document-Actions	7	[kein]	Word 2003, 2007	Eine Smart Document-Lösung ist aktiv
<i>Dokument schützen</i>	wdTaskPane-Document-Protection	6	7116	Word 2003, 2007	Schnittstelle für die Festlegung der Schutzart für ein Dokument und aktiviert diesen Schutz
<i>Dokumentaktualisierungen</i>	wdTaskPane-Document-Updates	13	7423	Word 2003, 2007	Das aktive Dokument ist eine Kopie eines im Dokumentarbeitsbereich vorhandenen Dokuments
<i>Dokumentverwaltung</i>	wdTaskPane-Document-Management	16		Word 2007	Stellt Features einer Dokumentarbeitsbereich-Website eines SharePoint-Services zur Verfügung
<i>Faxdienste</i>	wdTaskPane-FaxService	11	[kein]	Word 2003, 2007	Eine Faxdienstleistung ist verfügbar
<i>Formatvorlagen und Formatierungen</i>	wdTaskPane-Formatting	0		Word 2002, 2003, 2007 (unter dem Namen <i>Formatvorlage</i> )	Stellt eine Liste von Formatvorlagen und im Dokument vorhandenen Formatierungen zur Verfügung
<i>Formatinspektor</i>	wdTaskPane-Style-Inspector	15		Word 2007	Bietet Einsicht in die Formatierungseigenschaften der gegenwärtigen Markierung und ermöglicht deren Anpassung

**Tabelle 19.4** Die von Word zur Verfügung gestellten Aufgabenbereiche mit Informationen für den programmtechnischen Umgang (Fortsetzung)

Beschriftung	WdTaskPanes-Enumerator	Wert	Command-Bar.Control ID	Gültig für	Kontext
<i>Formatvorlage übernehmen</i>	wdTaskPane-ApplyStyle	17		Word 2007	Dient als Ersatz für die Combobox in der früheren Symbolleiste <i>Formatierung</i>
<i>Hilfe</i>	wdTaskPane-Help	9	984	Word 2003 (In Word 2007 blendet dieser Befehl das <i>Hilfe</i> -Fenster ein.)	Schnittstelle für die Suche nach einem Thema in der Hilfe und zeigt eine List der Suchergebnisse an
<i>Seriendruck</i>	wdTaskPane-MailMerge	2	6070	Word 2002, 2003, 2007	Enthält sechs Schritte, die den Benutzer durch die Erstellung eines Seriendrucks führt. Kann teilweise mit VBA beeinflusst werden (siehe Kapitel 7).
<i>Recherchieren</i>	wdTaskPane-Research	10	7343	Word 2003, 2007	Schnittstelle für das Recherchieren eines Themas. Bietet standardmäßig u.a. den Thesaurus sowie Übersetzungsdienste an, falls die Werkzeuge installiert sind.  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://msdn2.microsoft.com/en-us/library/aa159647(office.11).aspx">http://msdn2.microsoft.com/en-us/library/aa159647(office.11).aspx</a> .
<i>Formatierungen anzeigen</i>	wdTaskPane-Reveal-Formatting	1	6094	Word 2002, 2003, 2007	Zeigt an, mit welchen Formatierungen die Markierung formatiert wurde, und ob diese direkt oder von einer bestimmten Formatvorlage stammen.
<i>Einfache Suchoptionen</i>	wdTaskPane-Search	4	5905	Word 2002, 2003 (Die Konstante ist in Word 2007 noch vorhanden, verursacht jedoch eine Fehlermeldung »Dieser Befehl ist nicht verfügbar«.)	Eine Schnittstelle, um nach Dateien zu suchen

**Tabelle 19.4** Die von Word zur Verfügung gestellten Aufgabenbereiche mit Informationen für den programmtechnischen Umgang (Fortsetzung)

Beschriftung	WdTaskPanes-Enumerator	Wert	Command-Bar.Control ID	Gültig für	Kontext
Freigegebene Arbeitsbereiche	wdTaskPane-Shared-Workspace	8	7710	Word 2003, 2007	Verwaltung von Dokumenten, die auf einer SharePoint-Seite gespeichert sind
Recherchieren	wdTaskPane-Translate	3	7021	Word 2002, 2003, 2007	Der Arbeitsbereich <i>Recherchieren</i> im Übersetzungsmodus
Signaturen	wdTaskPane-Signature	14		Word 2007	Listet die digitalen Signaturen, mit denen ein Dokument signiert wurde, auf
XML-Dokument	wdTaskPane-XMLDocument	12		Word 2003, 2007	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 24).
XML-Struktur	wdTaskPane-XMLStructure	5		Word 2003, 2007	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 24).

## 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 Codefragment zeigt, wie sich feststellen lässt, 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
```



2007

Im Gegensatz zu früheren Versionen können in Word 2007 mehrere Aufgabenbereiche gleichzeitig eingeblendet werden. Es ist also notwendig, die `Visible`-Eigenschaft eines jeden Aufgabenbereichs zu prüfen. Nur der Aufgabenbereich *Dokumentaktionen* gibt für die Eigenschaft `Caption` einen Wert zurück.

## 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 19.4 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 wiedergegeben. 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 19.5 vorgestellt.

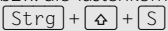
**Tabelle 19.5** Die Menübefehlsfolgen (alphabetisch nach Menüpunkt), die einen Aufgabenbereich einblenden, und der VBA-Code, um die Handlung abzufangen

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 Smart Document-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> gesteuert.
<i>Extras/Briefe und Sendungen/Serienbriefferstellung</i> sowie der Eintrag <i>Serien-Druck</i> aus der Liste im Aufgabenbereich	'Den alten Seriendruckassistent statt 'des Aufgabenbereichs einblenden Sub MailMergeWizard() Dialogs(wdDialogMailMergeHelper).Show End Sub

**Tabelle 19.5** Die Menübefehlsfolgen (alphabetisch nach Menüpunkt), die einen Aufgabenbereich einblenden, und der VBA-Code, um die Handlung abzufangen (*Fortsetzung*)

Menüpunkt	Codeschnipsel
<i>Extras/Dokument schützen</i> sowie der gleichnamige Eintrag aus der Liste im Aufgabenbereich	<pre>'Das Dialogfeld für Formatierungseinschränkungen direkt einblenden Sub ToolsProtect()     Dialogs( _         wdDialogFormattingRestrictions). _         Show End Sub  'Das alte Dialogfeld für den Dokumentschutz Sub ToolsProtect()     Dialogs( _         wdDialogToolsProtectDocument). _         Show End Sub</pre>
<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	<pre>'Den Befehl abfangen Sub Research() End Sub  Sub ResearchLookup End Sub</pre>
<i>Extras/Sprache/Übersetzen</i>	<pre>'Fängt den Befehl ab Sub Translate() End Sub  Sub TranslatePane() End Sub</pre>
<i>Extras/Thesaurus</i> (fängt die Suche im Aufgabenbereich <i>Recherchieren</i> <b>nicht</b> ab)	<pre>'Das alte Dialogfeld einblenden Sub ToolsThesaurusRR()     Dialogs(wdDialogToolsThesaurus).Show End Sub</pre>
<i>Faxdienst</i> (aus der Liste im Aufgabenbereich)	<pre>'Fängt den Befehl ab Sub FaxService() End Sub</pre>
<i>Format/Formatierung anzeigen</i> sowie der gleichnamige Eintrag aus der Liste im Aufgabenbereich	<pre>'Statt den Aufgabenbereich das Dialogfeld Format/Zeichen einblenden Sub FormattingProperties()     Dialogs(wdDialogFormatFont).Show End Sub</pre>
<i>Format/Formatvorlagen und Formatierungen</i> sowie der gleichnamige Eintrag aus der Liste im Aufgabenbereich	<pre>'Das Dialogfeld Formatvorlage anstelle des Aufgabenbereichs einblenden Sub FormattingPane()     Dialogs(wdDialogFormatStyle).Show End Sub</pre>

**Tabelle 19.5** Die Menübefehlsfolgen (alphabetisch nach Menüpunkt), die einen Aufgabenbereich einblenden, und der VBA-Code, um die Handlung abzufangen (*Fortsetzung*)

Menüpunkt	Codeschnipsel
Der Eintrag <i>XML Struktur</i> in der Liste im Aufgabenbereich	'Fängt den Befehl ab Sub ViewXMLStructure() End Sub
Der Befehl <i>Formatvorlage übernehmen</i> , der der QAT zugewiesen werden kann bzw. die Tastenkombination 	'Word 2007. Fängt den Befehl ab. Sub StyleApplyPane() End Sub
Die Befehlsfolge <i>Vorbereiten/Signaturen anzeigen</i> im Menü der Office-Schaltfläche, sofern das Dokument digital signiert ist.	'Word 2007. Fängt den Befehl ab. Sub ViewSignatures() End Sub
* In Word 2007 nicht vorhanden	



Die Beispieldatei *Bsp19\_01.doc* enthält einige der Prozedurskelette der Tabelle 19.5. Sie befindet sich auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap19*.

## Zur Verfügung stehende Aufgabenbereiche auflisten

Die Prozedur in Listing 19.1 wurde für die ursprünglichen Nachforschungen für dieses Kapitel verwendet. Sie schleift durch die TaskPanes-Auflistung und listet alle TaskPanes in einem neuen Dokument auf. Sie veranschaulicht, wie Aufgabenbereiche eingeblendet und angesprochen werden.



Word 2007 verwaltet Aufgabenbereiche anders, als Word 2003. In Word 2007 können mehrere Aufgabenbereiche gleichzeitig eingeblendet werden. Der einzige Aufgabenbereich, der über CommandBars ("Task pane") angesprochen wird, ist *Dokumentaktionen*, der Teil der Smart Document-Technologie ist. Der Code in Listing 19.1 liefert deshalb in Word 2007 kein brauchbares Ergebnis.

**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 19.1** 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
```

**Listing 19.1** Alle Aufgabenbereiche auflisten (Fortsetzung)

```
'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
        Resume Next
    End If
End With

Application.DisplayAlerts = wdAlertsAll
Set rngInsertion = Nothing
Set docReport = Nothing
End Sub
```



Die Beispieldatei *Bsp19\_02.doc* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap19*.

## 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. In Word 2007 ist nur der Aufgabenbereich *Formatvorlagen und Formatierungen* (unter dem Namen *Formatvorlagen*) vorhanden.

## 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 19.2 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.

- 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 das angegebene Dokument oder die angegebene 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 der Wert `msoEditFile` zugewiesen.

**Abbildg. 19.2**

Der Aufgabenbereich *Neues Dokument* mit den vier Zielabschnitten, die den *Add-* sowie *Remove-*Methoden zur Verfügung stehen



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 19.2 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 19.2** 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, _
    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
End Sub

'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
```

**Listing 19.2** Die Prozeduren *NewDocumentAdd* und *NewDocumentRemove* sind Wrapper für das Hinzufügen und Entfernen von Einträgen in Aufgabenbereichen (*Fortsetzung*)

```
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 19.3), 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
```

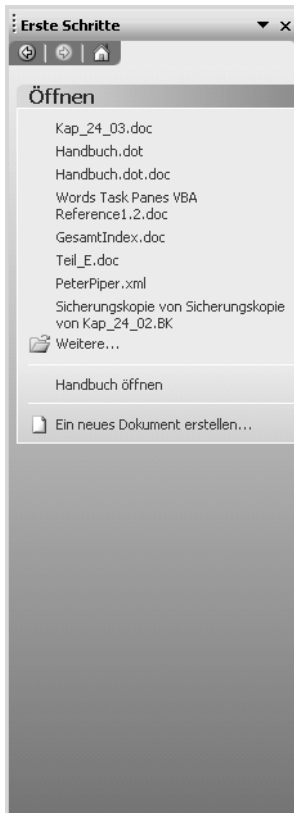
Auch der Aufgabenbereich *Erste Schritte* ist mit dem *NewDocument*-Objekt und seinen *Add*- und *Remove*-Methoden ansprechbar. Die Syntax ist die gleiche, nur ist der einzige zur Verfügung stehende Abschnitt *msoOpenDocument*. Diese Einträge erscheinen im mittleren Abschnitt des Aufgabenbereichs.

Das folgende Codefragment zeigt, wie ein Eintrag in den Aufgabenbereich mit Hilfe des Wrappers in Listing 19.2 eingefügt wird.

```
Sub EintraginErsteSchritteEinfügen()
    NewDocumentAdd "C:\test\Dok4.doc", msoOpenDocument, "Dok 4", msoEditFile
End Sub
```



**Abbildg. 19.3** Der Eintrag »Handbuch öffnen« im Aufgabenbereich *Erste Schritte* wurde mit der *NewDocument.Add*-Methode hinzugefügt



Die Beispieldatei *Bsp19\_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\Kap19*.

## 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?

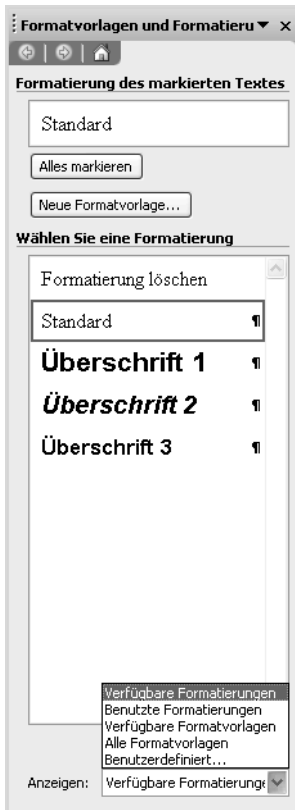
### Word 2003

Zuunster im Aufgabenbereich befindet sich eine Dropdownliste *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 19.4 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. 19.4 Die Liste des Aufgabenbereichs *Formatvorlagen und Formatierungen* lässt sich anpassen

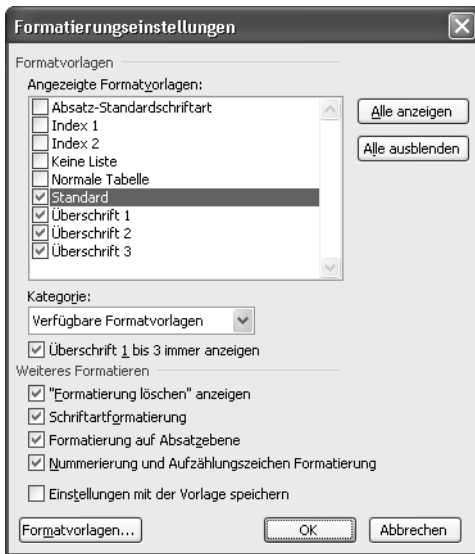


**HINWEIS**

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 19.5 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 anzeigen soll.

**Abbildg. 19.5** Die Elemente, Formatierungen und Formatvorlagen nach Kategorie festlegen, die im Aufgabenbereich aufzulisten sind



## Word 2007



Diese Schnittstelle für den Einsatz von Formatierungen und Formatvorlagen wurde für Word 2007 verfeinert sowie erweitert. Dadurch wird es einerseits dem Benutzer einfacher gemacht, die korrekten Formatierungen anzuwenden. Andererseits stehen dem Entwickler von Dokumentvorlagen sowie dem professionellen Dokumentbearbeiter bessere Verwaltungswerkzeuge zur Verfügung.

Den Aufgabenbereich in Word 2007 entnehmen Sie bitte der Abbildung 19.6. Über die zwei Kontrollkästchen unten wird

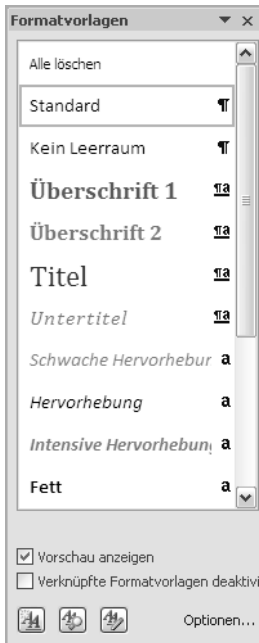
- Die Formatierung in der Anzeige unterdrückt bzw. zugelassen
- Die Zuweisung von verknüpften Formatvorlagen für Textmarkierungen ausgesetzt (die Formatvorlage wird dem ganzen Absatz zugewiesen)



### HINWEIS

Verknüpfte Formatvorlagen wurden in Word 2002 eingeführt und sind für die berechtigten »Zchn«-Einträge in den Listen verantwortlich. Bis Word 2007 blieben sie dem Benutzer mehr oder weniger verborgen. Mehr dazu finden Sie in Kapitel 6.

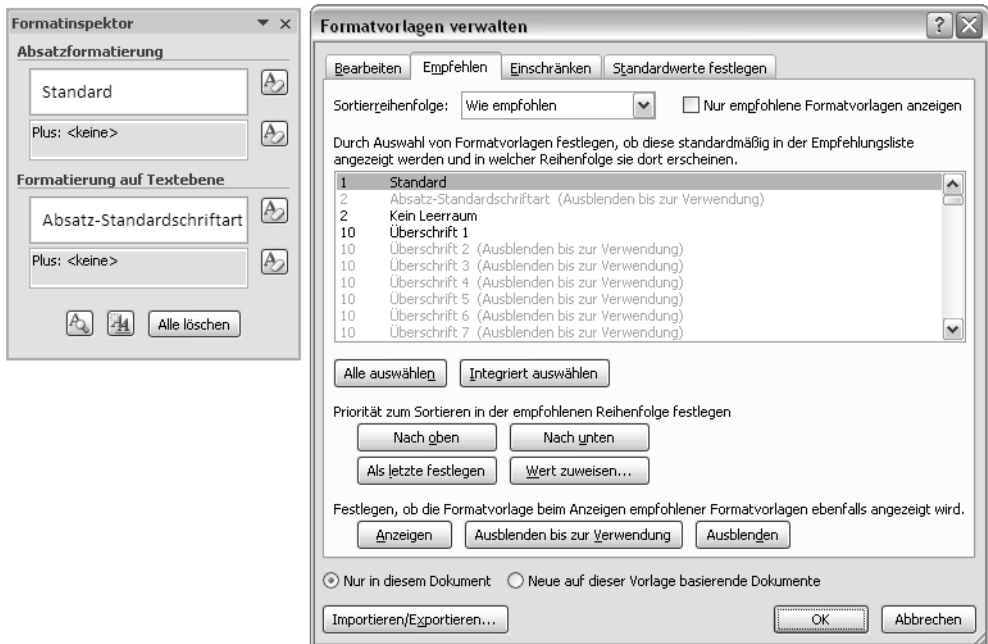
Abbildg. 19.6 Der Word 2007-Aufgabenbereich *Formatvorlagen* unterscheidet sich von dem in Word 2003



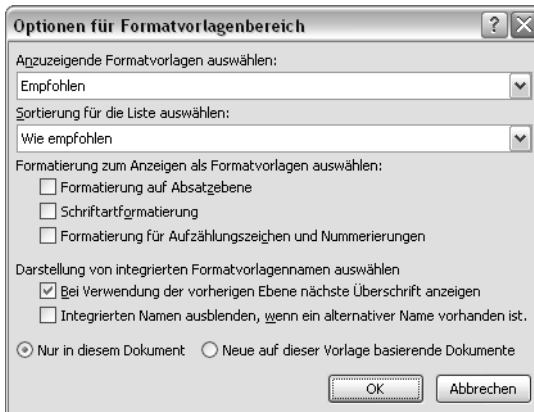
Die drei Schaltflächen sowie der Link blenden weitere Aufgabenbereiche und Dialogfelder ein (von links nach rechts):

- *Neue Formatvorlage*
- *Formatinspektor* (Abbildung 19.7, links), womit die direkte Formatierung eines Absatzes oder einer Textstelle gezielt entfernt werden kann
- *Formatvorlagen verwalten* (Abbildung 19.7, rechts), der Zugang für die Bearbeitung aller Formatvorlagen bietet, sowie die Sichtbarkeit und Verfügbarkeit der Formatvorlagen im Dokument verwaltet
- *Optionen* für den Aufgabenbereich *Formatvorlagen* (Abbildung 19.8), der eine ähnliche Funktionalität wie das Dialogfeld *Formatierungseinstellungen* in Word 2003 hat

Abbildg. 19.7 Neue Werkzeuge für die Arbeit mit Formatvorlagen in Word 2007



Abbildg. 19.8 Festlegen, welche Einträge im Word 2007-Aufgabenbereich *Formatvorlagen* sichtbar sind



**HINWEIS** Mehr Informationen zu den neuen Möglichkeiten finden Sie im Microsoft Press-Buch »Office 2007 – Das Profibuch«.

Im Gegensatz zu Word 2003 und früher hat der Anwender in Word 2007 die Möglichkeit, Formatvorlagen eine Priorität zuzuweisen. Sortiert werden können die Listen in den Aufgabenbereichen nach Bedarf alphabetisch, nach dem Typ, nach der Schriftart, nach der basierenden Formatvorlage

oder nach Priorität. Zudem stehen drei Stufen der Sichtbarkeit zur Verfügung: *Anzeigen*, *Ausblenden bis zur Verwendung* sowie *Ausblenden*.

### Die Liste im Aufgabenbereich programmtechnisch anpassen

Egal, um welche Version von Word es sich handelt, hört sich das Anpassen der Liste zwar leicht an, ist aber in der Handhabung etwas komplex, vor allem wenn sie programmtechnisch erfolgen soll.

Die Tabelle 19.6 listet die benötigten Eigenschaften auf, um den Inhalt der Liste zu bestimmen. Wenn Sie demzufolge den Eintrag *Formatierung löschen* (in Word 2007 *Alle löschen*) aus der Liste verbannen möchten, müsste es mit dieser Codezeile klappen:

```
ActiveDocument.FormattingShowClear = False
```

VBA führt die Anweisung auch ohne Probleme aus, und in Word 2007 verschwindet der Eintrag *Alle löschen* aus der Liste. Leider ist in Word 2003 keine Wirkung ersichtlich. Würden Sie das Dialogfeld *Formatierungseinstellungen* öffnen, wäre das Kontrollkästchen tatsächlich deaktiviert. Kopfschüttelnd 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 19.6** 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 19.7)
Schriftartformatierung	Document.FormattingShowFont	Boolean
Nummerierung und Aufzählungszeichen Formatierung	Document.FormattingShowNumbering	Boolean
Formatierung auf Absatzebene	Document.FormattingShowParagraph	Boolean
* Ob bei Verwendung der vorherigen Überschriftenebene die nächste Überschriftenebene angezeigt wird	Document.FormattingShowNextLevel	Boolean
* Ob benutzerdefinierte Formatvorlagen angezeigt werden	Document.FormattingShowUserStyleName	Boolean
Extras/Optionen/Bearbeiten/Formatierung verfolgen	Options.FormatScanning	Boolean
* Neu in Word 2007		

Tabelle 19.7 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
* wdShowFilterFormattingRecommended	5	Empfohlene Formatvorlagen
* Neu in Word 2007		

Soviel zu den allgemeinen Befehlen für den Aufgabenbereich. Wie steht's mit der Auflistung der Formatvorlagen?

Das Style-Objekt hat eine verborgene Eigenschaft: *Visibility* (mehr über verborgene Elemente finden Sie in Kapitel 1). Vermutlich ist sie verborgen, weil sie sich nur im beschränkten Rahmen anwenden lässt. Damit lässt sich die Sichtbarkeit eines Formatvorlageneintrags im Aufgabenbereich festlegen. Kurioserweise wird der Eintrag aufgeführt, wenn *Visibility* = *False* ist, und wird unterdrückt, wenn *Visibility* = *True*.

In Word 2003 wirkt sich diese Eigenschaft nur bei einer neuen *Vorlage* aus. 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 19.3 veranschaulicht die programmtechnische Voreinstellung des Aufgabenbereichs *Formatvorlagen und Formatierungen*.

Listing 19.3 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 *Bsp19\_04.dot* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap19*.



2007

Auch in Word 2007 bleibt die Eigenschaft *Visibility* verborgen. Sie verliert jedoch ihre Wirkung nicht wie in Word 2003 und kann jederzeit angewendet werden. In der Benutzeroberfläche entspricht sie den Schaltflächen *Anzeigen* (*Visibility* = *False*) sowie *Ausblenden* (*Visibility* = *True*) im Dialogfeld *Formatvorlagen verwalten*.

Wie steht's jedoch mit dem Befehl *Ausblenden bis zur Verwendung*? Die entsprechende Eigenschaft – *UnhideWhenUsed* – ist ebenfalls verborgen. Die beiden, *Visibility* und *UnhideWhenUsed*, arbeiten zusammen. Entspricht *Visibility* dem Wert »Falsch«, zeigt *UnhideWhenUsed* keine Wirkung – die Formatvorlage ist auf jeden Fall im Aufgabenbereich sichtbar. Bei *Visibility* = *True* bestimmt *UnhideWhenUsed*, ob die Formatvorlage bei der Verwendung im Dokument in der Liste sichtbar wird. Ist der Wert »Falsch«, bleibt sie verborgen. Einmal sichtbar hat *UnhideWhenUsed* keine Wirkung mehr; wird sie auf »Falsch« festgelegt, bleibt der Eintrag im Aufgabenbereich sichtbar, außer *Visibility* wird erneut auf »Wahr« festgelegt.

Noch eine verborgene Eigenschaft kontrolliert die Sichtbarkeit einer Formatvorlage: *Hidden*. Sie kommt zwar auch im Objektmodell von Word 2003 vor, zeigt dort jedoch keine Wirkung. In Word 2007 hingegen wird bei der Festlegung *Hidden* = *True* die Formatvorlage aus jeglicher Liste, auch denjenigen im Dialogfeld *Formatvorlagen verwalten*, verbannt.

Im Gegensatz zu den vorherigen Eigenschaften ist *Priority*, um die Priorität des Eintrags in einer Liste festzulegen, nicht verborgen. Sie entspricht der Schaltfläche *Wert zuweisen*. *Priority* erwartet eine Ganzzahl, die festlegt, wo in der Liste sich die Formatvorlage einreihen soll. Je niedriger die Zahl, desto höher in der Liste befindet sich der Eintrag, wenn die Sortierreihenfolge »Wie empfohlen« beträgt.

Auch diese Sortierreihenfolge kann über das Objektmodell festgelegt werden, und zwar durch die Verwendung der Eigenschaft *Document.StyleSort*. Die verfügbaren Werte entnehmen Sie bitte der Tabelle 19.8.

**Tabelle 19.8** *WdStyleSort*-Werte für die Word 2007-Eigenschaft *Document.StyleSort*

WdStyleSort-Enumeration	Wert	Entspricht der Option
<i>wdStyleSortByName</i>	0	Alphabetisch
<i>wdStyleSortRecommended</i>	1	Wie empfohlen
<i>wdStyleSortByFont</i>	2	Schriftart
<i>wdStyleSortByBasedOn</i>	3	Basierend auf
<i>wdStyleSortByType</i>	4	Nach Typ

Das Listing 19.4 veranschaulicht einige der hier vorgestellten Word 2007-Eigenschaften für die Bereitstellung von Formatvorlagen. Im aktiven Dokument werden die Word-eigenen Formatvorlagen ausgeblendet, dann neue erstellt. Nur diese werden in den Listen angezeigt.



Listing 19.4 Die Formatvorlagen für das aktuelle Dokument einrichten

```

Sub Firma_Dok_Erstellen()
    Dim fvText As Word.Style
    Dim styl As Word.Style
    Dim doc As Word.Document

    Set doc = ActiveDocument
    'Alle vorhandenen Formatvorlagen verbergen.
    For Each styl In doc.Styles
        styl.Visibility = True
    Next
    'Die ersetzten Formatvorlagen gänzlich aus den Listen entfernen.
    doc.Styles(wdStyleHeading1).Hidden = True
    doc.Styles(wdStyleHeading2).Hidden = True
    doc.Styles(wdStyleHeading3).Hidden = True
    doc.Styles(wdStyleBodyText).Hidden = True

    Set fvText = FV_Text_erstellen(doc)
    FV_U1_Erstellen doc
    FV_U2_Erstellen doc
    FV_U3_Erstellen doc

    doc.FormattingShowFilter = wdShowFilterFormattingRecommended

    doc.FormattingShowUserStyleName = True
    doc.StyleSortMethod = wdStyleSortRecommended
    doc.Content.Style = fvText
    'Den Aufgabenbereich Formatvorlagen einblenden.
    CommandBars.ExecuteMso idMso:="StylesPane"
End Sub

Function FV_U1_Erstellen(doc As Word.Document) As Word.Style
    Dim styl As Word.Style
    Set styl = doc.Styles.Add("U1", wdStyleTypeParagraphOnly)
    styl.Priority = 2
    styl.Visibility = False
    styl.Hidden = False
    With styl.Font
        'Schriftartformatierungen werden in den folgenden Zeilen festgelegt
    End With
    With styl.ParagraphFormat
        'Absatzformatierungen werden in den folgenden Zeilen festgelegt.
    End With
    styl.NoSpaceBetweenParagraphsOfSameStyle = False
    styl.LanguageID = wdSwissGerman
    styl.NoProofing = False
    styl.Frame.Delete
    styl.QuickStyle = True
    styl.NextParagraphStyle = doc.Styles("Fliesstext")
    styl.Locked = True
    Set FV_U1_Erstellen = styl
End Function

Function FV_Text_erstellen(doc As Word.Document) As Word.Style
    'Ähnlicher Code wie für die Funktion FV_U1_Erstellen folgt.
End Function

```

**Listing 19.4** Die Formatvorlagen für das aktuelle Dokument einrichten (*Fortsetzung*)

```
Function FV_U2_Erstellen(doc As Word.Document) As Word.Style
    'Ähnlicher Code wie für die Funktion FV_U1_Erstellen folgt.
End Function

Function FV_U3_Erstellen(doc As Word.Document) As Word.Style
    'Ähnlicher Code wie für die Funktion FV_U1_Erstellen folgt.
End Function
```



Die Dokumentvorlagen *Bsp19\_05.dotm* befinden sich auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap19*. Kopieren Sie die Datei in den *Startup*-Ordner, um die Formatvorlageneinstellungen in einem beliebigen Dokument vorzunehmen.

## 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 hinter dem Aufgabenbereich stehende Technologie (Seite 780).
- Welche Schnittstellen das Word-Objektmodell für »Work Panes« zur Verfügung stellt (Seite 781).
- Registry-Einträge, mit denen das Verhalten von Aufgabenbereichen beeinflusst werden kann (Seite 782).
- Von Aufgabenbereichen verursachte Fehlermeldungen (783).
- Die programmtechnische Verwaltung von Aufgabenbereichen (Seite 784).
- Wie der Inhalt bestimmter Aufgabenbereiche (*Neues Dokument*, *Erste Schritte*, sowie *Formatvorlagen und Formatierungen*) beeinflusst werden kann (Seite 792).

## Kapitel 20

# Interne Word-Befehle übersteuern

### In diesem Kapitel:

Word-Befehl außer Kraft setzen

808

Zusammenfassung

812

Bereits in Kapitel 1 wurde dargestellt, 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.

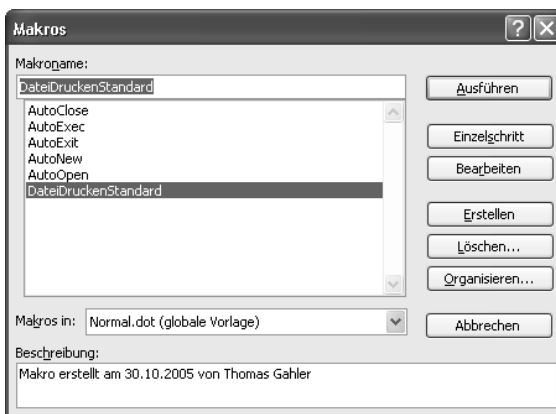
Soll ein einzelner Word-Befehl übersteuert werden, muss ein Makro (Public Sub, ohne Parameter) mit identischer Bezeichnung in einem VBA-Projekt angelegt werden. Dabei spielt es keine Rolle, ob dieses Makro in einem Dokument, der zugehörigen Dokumentvorlage, einem Add-In oder in der *Normal.dot* angelegt 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 20.1). 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 *Bsp20\_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. 20.1 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 20.1, weiterhin benötigt, muss innerhalb des entsprechenden Makros wieder eine Möglichkeit eingebaut werden.

**Listing 20.1** 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 20.1 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 20.2 und Listing 20.3).

**Listing 20.2** 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 Standarddrucker ausgegeben. Ansonsten wird der Ausdruck verweigert und der Anwender entsprechend informiert.

**Listing 20.3** 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 15 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.

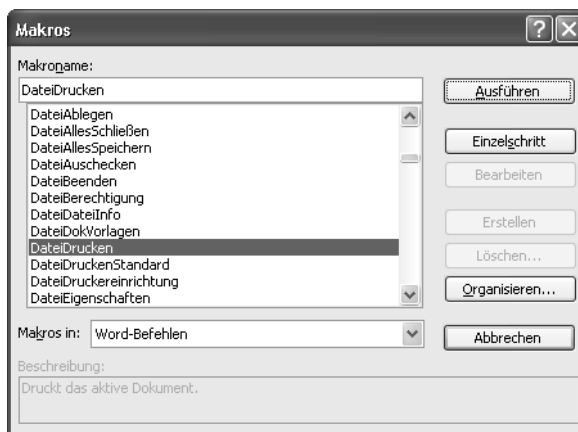
Um diese Liste einsehen zu können, wählen Sie den Menübefehl *Extras/Makro/Makros*. Aktivieren Sie im Listenfeld *Makros in* den Eintrag *Word-Befehlen*. Jetzt werden im Listenfeld *Makroname* alle internen Word-Befehle aufgelistet.



2007

In Word 2007 wählen Sie in der Multifunktionsleiste auf der Registerkarte *Entwicklertools* die Schaltfläche *Makros*, um das Dialogfeld in Abbildung 20.2 einzublenden.

**Abbildg. 20.2** Eine Liste der integrierten Word-Befehle anzeigen lassen



**HINWEIS**

Die Namen der einzelnen Befehle können in vielen Fällen vom Aufbau des Originalmenüs in Word 2003 und früher 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 20.2 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 20.1 zusammengefasst.

**Tabelle 20.1** 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 *Bsp20\_01.dot* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap20*.

# Zusammenfassung

In diesem Kapitel wurde gezeigt, 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 808).
- Es wurde darauf hingewiesen, dass die ursprüngliche Funktionalität nicht mehr zur Verfügung steht, sofern diese nicht im eigentlichen Makro zusätzlich eingebaut wird (Seite 808).
- Schließlich wurde vermittelt, wie die Bezeichnung eines Befehls ermittelt und unabhängig von der eingesetzten Programmsprache verwendet werden kann (Seite 810).



## Kapitel 21

# Zugriff auf den Visual Basic-Editor (VBE)

### In diesem Kapitel:

Notwendige Verweise und Sicherheitseinstellungen	814
Der Visual Basic-Editor	815
Auslesen des VBA-Codes von Komponenten	823
Ersetzen und Entfernen von VBA-Code-Zeilen	831
Hinzufügen von Komponenten zu einem Projekt	834
Entfernen von Komponenten aus einem Projekt	845
Anzeigen von dynamisch erzeugten UserForms	846
Verweise auf Bibliotheken und Dateien ermitteln und zur Laufzeit setzen	849
Zusammenfassung	857

In Kapitel 1 »Word-Makros« wurde der Umgang mit dem Visual Basic-Editor und die Erstellung und Bearbeitung von Makros im Visual Basic-Editor kurz vorgestellt.

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.

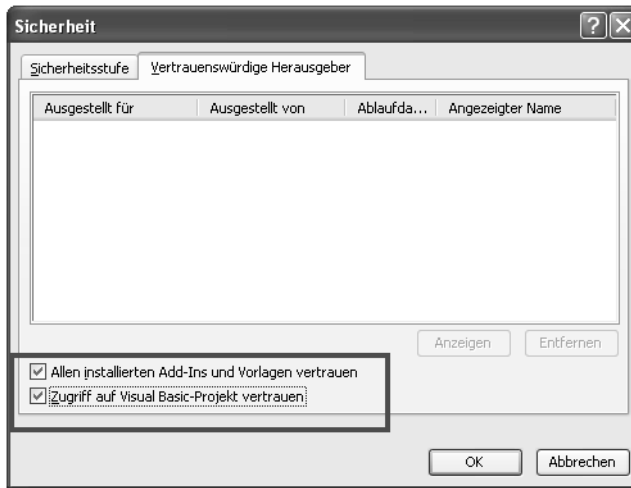
## Notwendige Verweise und Sicherheitseinstellungen

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 21.1 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.



In Word 2007 wird die Sicherheit im *Vertrauensstellungscenter* der *Word-Optionen* verwaltet, das in Kapitel 1 vorgestellt wurde. Stellen Sie sicher, dass sich die Dokumentvorlage in einem vertrauten Speicherort befindet, und aktivieren Sie das Kontrollkästchen *Zugriff auf das VBA-Projektobjektmodell vertrauen* in der Kategorie *Einstellungen für Makros* im Vertrauensstellungscenter.

Abbildg. 21.1 Sicherheitsstufe anpassen: Zugriff auf Visual Basic-Projekt vertrauen



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 9).

**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 (Abbildungung 21.2) 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:

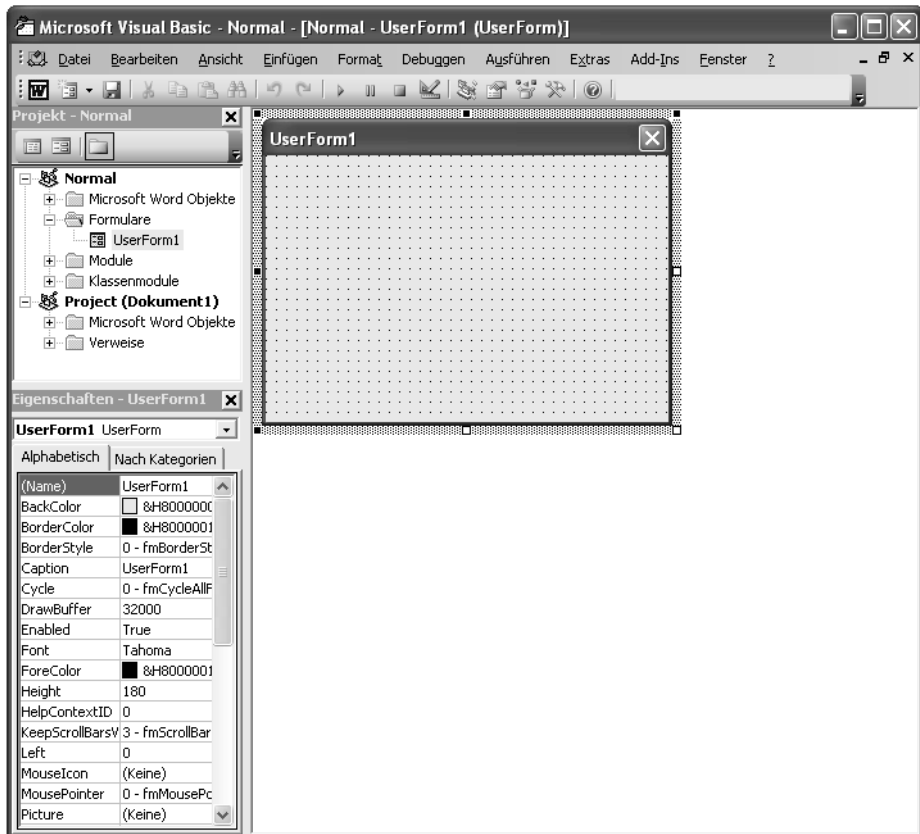
- **Code- bzw. UserForm-Fenster**  
Eingabebereich zum Erstellen, Anzeigen und Bearbeiten von VBA-Code bzw. zum grafischen Erstellen und Bearbeiten von Fenstern oder Dialogfeldern.
- **Projekt-Explorer**  
Zeigt eine hierarchische Liste der Projekte und aller Elemente an, die in den jeweiligen Projekten enthalten sind bzw. von den jeweiligen Projekten referenziert werden.

## ■ Eigenschaftenfenster

Zeigt die Entwurfszeiteigenschaften für ein ausgewähltes Objekt und deren aktuelle Einstellungen an.

Abbildg. 21.2

Übersicht über den Visual Basic-Editor mit UserForm-Fenster, Projekt-Explorer und Eigenschaftenfenster



### 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 *Eigenschaftenfenster* (F4) einblenden.

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
```

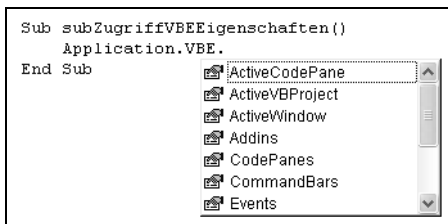
**TIPP**

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 21.3).

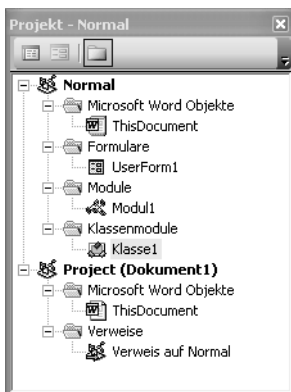
Abbildg. 21.3 Eigenschaft des VBE-Objektes



## Die VBA-Projekte

Im Projekt-Explorer des Visual Basic-Editors (Abbildung 21.4) 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. 21.4 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 und 2007 kennen zwei *StartUp*-Ordner:

- Einen im Programmordner von Microsoft Office 2003  
C:\Programme\Microsoft Office\OFFICE11\STARTUP
- Bzw. einen im Programmordner von Microsoft Office 2007  
C:\Programme\Microsoft Office\OFFICE12\STARTUP
- Einen benutzerspezifischen für jeden Anwender unter Windows XP  
C:\Dokumente und Einstellungen\<Benutzername>\Anwendungsdaten\Microsoft\Word\StartUp
- Bzw. einen benutzerspezifischen für jeden Anwender unter Windows Vista  
C:\Users\<Benutzername>\AppData\Roaming\Microsoft\Word\STARTUP

sofern Microsoft Office 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\Kap21\Bsp21\_01.doc im Modul *modProjektNamen*.

Das Beispiel in Listing 21.1 zeigt für jedes Projekt neben dem Namen auch die Zugriffsmöglichkeit an:

**Listing 21.1**

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 21.2 veranschaulicht.

**Listing 21.2** Ausgabe des vollständigen Projekt-Namens aller geladenen Projekte

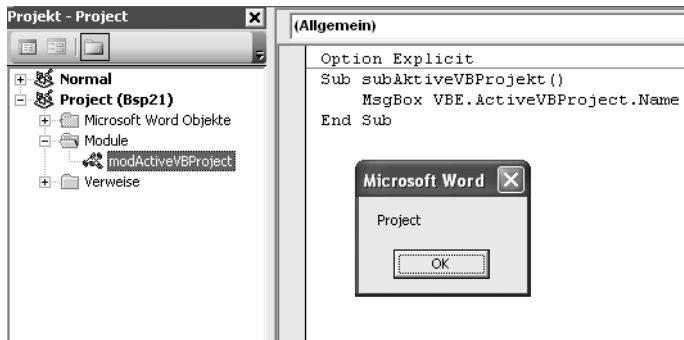
```
Sub subAlleVBProjektNamen()
    Dim strVBP As String
    Dim vbp As VBProject
    For Each vbp In VBE.VBProjects
        On Error Resume Next
        If vbp.FileName = "" Then
            strVBP = strVBP & vbp.Name & ": " & vbTab & "nicht gespeicherte Datei" & vbCrLf
        Else
            strVBP = 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.

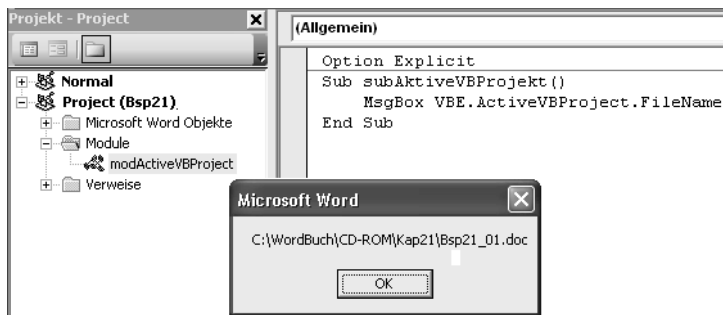
Den Namen des aktuellen Projektes erhalten Sie, wie in Abbildung 21.5 ersichtlich, wieder über die Name-Eigenschaft des ActiveVBProject-Objektes:

```
Sub subAktiveVBProjektName()
    MsgBox VBE.ActiveVBProject.Name
End Sub
```

**Abbildg. 21.5** Anzeige des aktuellen Projekt-Namens


Den Pfad des aktuellen Projektes erhalten Sie wieder über die FileName-Eigenschaften, sofern das Projekt gespeichert wurde (Abbildung 21.6):

```
Sub subAktiveVBProjekt()  
    MsgBox VBE.ActiveVBProject.FileName  
End Sub
```

**Abbildg. 21.6** 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



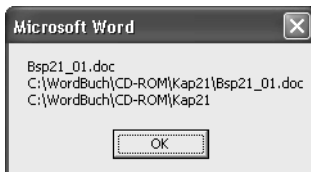
- Path

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 21.7):

```
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. 21.7 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 zum Buch in der Datei `\Beispiele\Kap21\Bsp21_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 21.4 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 die 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 21.1.

**Tabelle 21.1** 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 21.3** 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: "
            Case 1
                strTyp = "Modul:      "
            Case 2
                strTyp = "Klassenmodul: "
            End Select
        End With
    Next vbc
End Sub
```

**Listing 21.3** Auflistung aller in einem Projekt enthaltenen *VBComponents* mit Typangabe (Fortsetzung)

```

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 21.3 werden alle im aktiven VBProject enthaltenen Komponenten mit Typ-Aufschlüsselung und ihrem Namen angezeigt, wie in Abbildung 21.8 ersichtlich.

**Abbildg. 21.8** 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 21.2 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 21.2 erhalten Sie Zugriff auf die Zeilen eines Code-Moduls.

**Tabelle 21.2** Übersicht über die Methoden zum Zugriff auf ein Code-Modul

Zugriffsmethode	Beschreibung
<code>CountOfDeclarationLines</code>	Gibt die Anzahl der Code-Zeilen im Deklarationsabschnitt eines Code-Moduls zurück (siehe den Abschnitt »Zugriff auf den Deklarationsbereich«)
<code>CountOfLines</code>	Gibt die Gesamtzeilenzahl des Code-Moduls an (siehe den Abschnitt »Zugriff auf die Code-Zeilen einzelner Prozeduren«)

**Tabelle 21.2** Übersicht über die Methoden zum Zugriff auf ein Code-Modul (Fortsetzung)

Zugriffsmethode	Beschreibung
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 zum Buch in der Datei `\Beispiele\Kap21\Bsp21_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.

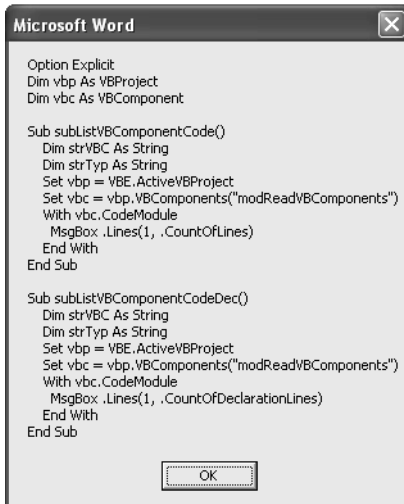
**Listing 21.4** Ausgabe des gesamten VBA-Code eines Code-Moduls

```
Sub subListVBComponentCode()
    Dim strVBC As String
    Dim strTyp As String
    Set vbp = VBE.ActiveVBProject
    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 21.9.

**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. 21.9 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 21.5 veranschaulicht.

Listing 21.5 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 21.9 in Abbildung 21.10 angezeigt.

**Abbildg. 21.10** 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 21.9 die beiden Prozeduren `subListVBComponentCode` und `subListVBComponentCodeDec`.

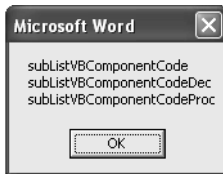
Die Prozeduren selbst lassen sich nur dann ansprechen, wenn die Prozedurnamen bekannt sind. Leider steht keine Methode oder Eigenschaft zur Verfügung, die diese Informationen direkt liefert. Sie können jedoch die Eigenschaft `ProcOfLine` verwenden, die den Prozedurnamen der aktuellen Zeile zurückliefert. 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 21.5 durchläuft zeilenweise den Code der Komponente *modReadVBComponents* und liefert nur die Prozedurnamen zurück. Das Ergebnis ist in Abbildung 21.11 ersichtlich.

**Listing 21.6** 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. 21.11** 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.

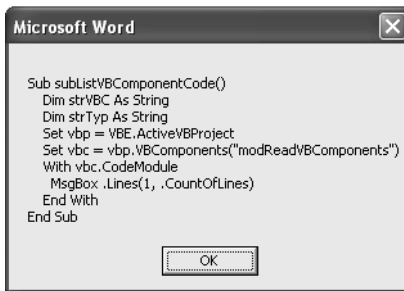
**ACHTUNG** Die `ProcBodyLine`-Eigenschaft liefert die erste Zeile einer Prozedur zurück, in der die Sub-, Function- oder Property-Anweisung enthalten ist. Allerdings können vor der Anweisung auch noch Leerzeilen und Kommentarzeilen stehen, die ebenfalls zu dieser Prozedur gehören.

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 21.7; das Ergebnis befindet sich in Abbildung 21.12.

**Listing 21.7** 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. 21.12** 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 21.8 überprüft zeilenweise auf die Prozedurtypen und liest dann über die passende Konstante die Prozedur aus. ►



**Listing 21.8** Zugriffsermittlung für Eigenschaftenprozeduren und Ausgabe der Prozeduren

```

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 Zugriffs auf den VBA-Code möchten wir Ihnen in Listing 21.9 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 zum Buch in der Datei `\Beispiele\Kap21\Bsp21_01.doc` im Modul `modSearchAndReplaceLines`.

**Listing 21.9** 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
          Exit For
        End If
      End With
    End For
  Next vbp
End Sub

```

**Listing 21.9** Durchsuchen aller Projekte nach einer bestimmten Prozedur (Fortsetzung)

```

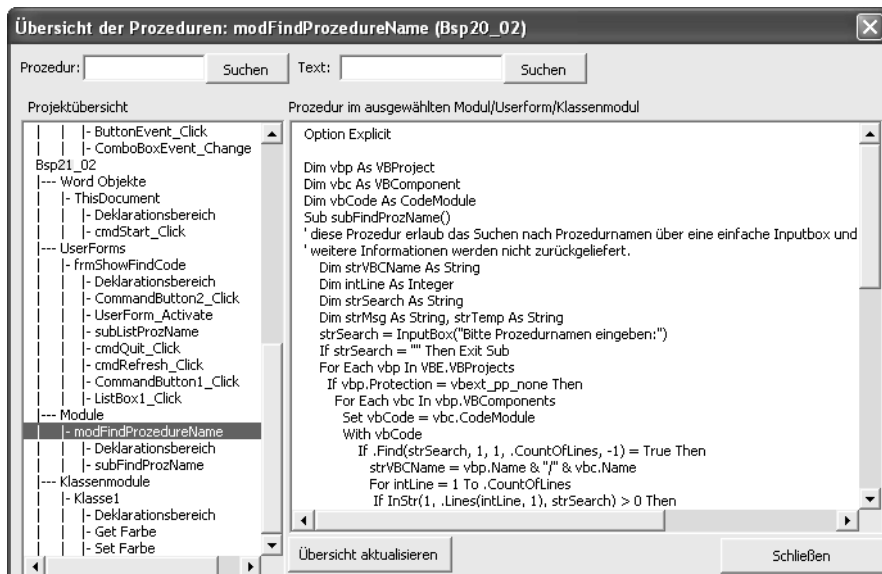
        Next intLine
    End With
    Next vbc
p_next:
    Next vbp
    Set vbCode = Nothing
End Sub

```



In der Beispieldatei *Bsp21\_02.doc* (auf der CD-ROM zum Buch im Ordner *(Beispiele)\Kap21*) finden Sie ein zusätzliches Beispiele, 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 Textsuche über die angezeigte Schaltfläche *VBA-Code/-Prozedur suchen* auf. Es öffnet sich daraufhin das in Abbildung 21.13 gezeigte Benutzerformular.

**Abbildg. 21.13** 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 21.2 aufgelisteten Zugriffsmethoden sind aber auch Methoden zum Ersetzen und Entfernen von VBA-Code-Zeilen enthalten.

**Tabelle 21.3** 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 21.10 sucht in allen Projekten nach der Prozedur »subFindProzName« (siehe Listing 21.9) 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 21.14 ersichtlich; beachten Sie die beiden markierten Bereiche, wo die Änderungen vorgenommen wurden.



Die Beispiel-Prozeduren finden Sie auf der CD-ROM zum Buch in der Datei \Beispiele\Kap21\Bsp21\_01.doc im Modul modSearchAndReplaceLines.

**Listing 21.10** 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
            End If
        
```

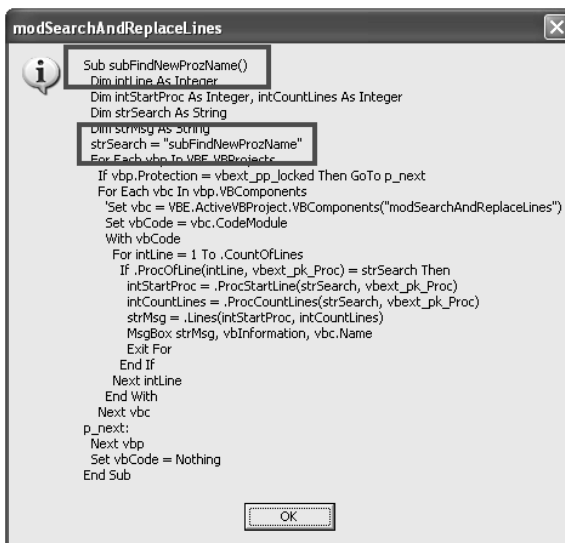
**Listing 21.10** Umbenennen und Ersetzen einer Zeile in einer Prozedur und Ausführen der geänderten Prozedur (*Fortsetzung*)

```

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. 21.14** Ergebnis nach der Ausführung von Listing 21.10



**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 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 21.11 löscht die ersten beiden Zeilen im Code-Modul, wenn in der ersten Zeile

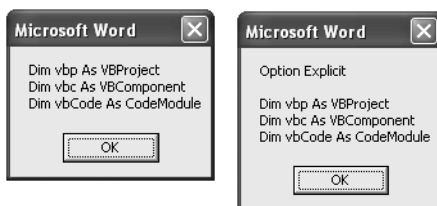
```
Option Explicit
```

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 21.15 ersichtlich.

**Listing 21.11** 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. 21.15** 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 zum Buch in der Datei `\Beispiele\Kap21\Bsp21_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 1) vorliegende Komponente (Benutzerformular, Modul oder Klassenmodul) mit dem VBA-Code.
- 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 21.4 stehen Ihnen zum Erstellen und Bearbeiten von Komponenten, Ereignissen und Code-Zeilen zur Verfügung:

**Tabelle 21.4** Übersicht über die Methoden zum Erstellen und Bearbeiten von Komponenten per VBA-Code

Zugriffsmethode	Beschreibung
<code>AddFromFile</code>	Fügt den Inhalt einer angegebenen Datei in ein Code-Modul ein (siehe den Abschnitt »Makro-Anweisungen per VBA-Code importieren«)
<code>AddFromString</code>	Fügt einen angegebenen Text in das Code-Modul in die erste Zeile ein (siehe den Abschnitt »Makro-Anweisungen per VBA-Code einfügen«)
<code>CreateEventProc</code>	Erstellt für ein angegebenes Objekt eine Ereignisprozedur (siehe den Abschnitt »Makro-Anweisungen per VBA-Code einfügen«)
<code>DeleteLines</code>	Löscht eine oder mehrere Zeilen (siehe den Abschnitt »Ersetzen und Entfernen von VBA-Code-Zeilen«)
<code>InsertLines</code>	Fügt eine oder mehrere Zeilen an einer bestimmten Stelle in das Code-Modul ein (siehe den Abschnitt »Makro-Anweisungen per VBA-Code einfügen«)
<code>ReplaceLine</code>	Ersetzt eine vorhandene Code-Zeile durch eine angegebene Code-Zeile (siehe den Abschnitt »Ersetzen und Entfernen von VBA-Code-Zeilen«)

## Eine vorhandene 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« in diesem Kapitel):

```
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.

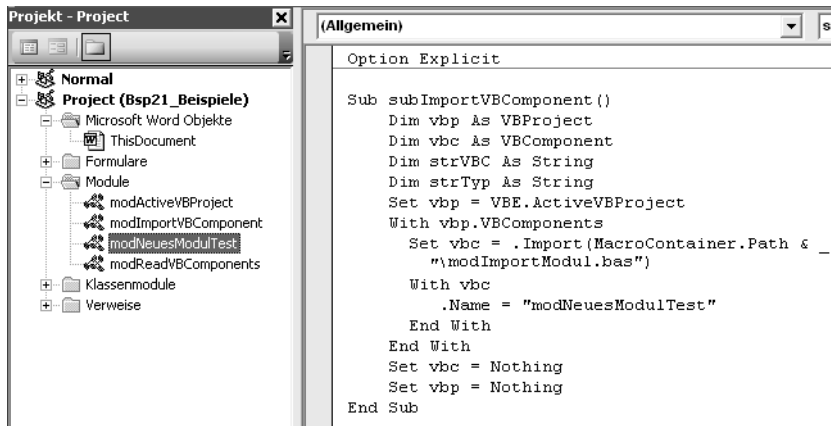
Als Komponenten-Namen wird der normalerweise in der importierten Datei als Attribut vermerkte Name verwendet. Sie können diesen aber nach dem Import über die Name-Eigenschaft ändern, wie Listing 21.12 veranschaulicht. Das Ergebnis ist in Abbildung 21.16 ersichtlich.

**Listing 21.12** 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
    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 *Bsp21\_01.doc* und *modImportModul.bas* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap21*.

**Abbildg. 21.16** Anzeige des importierten Moduls im Projekt-Explorer

**WICHTIG**

Existiert bereits eine Komponente mit dem intern vermerkten Komponenten-Namen, wird beim Importieren der Komponenten-Namen 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 21.12 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 21.1 aufgeführten *vboxt*-Konstantenwerte.

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 21.13 erstellt ein neues leeres Standardmodul mit dem Namen *modNeuesModul*.

**Listing 21.13** 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
```



Listing 21.13 Hinzufügen eines neuen Standardmoduls zum aktiven Projekt (Fortsetzung)

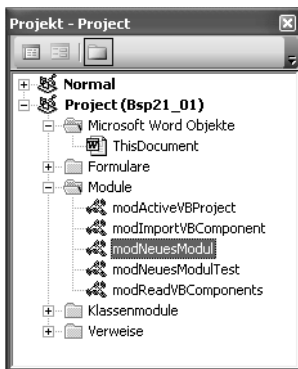
```

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 21.17 veranschaulicht.

Abbildg. 21.17 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.

Diese Option sollte unbedingt verwendet werden, da sich dadurch Fehler, die durch falsche Variablendeklaration 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 21.14 zeigt ein Beispiel hierfür, wenn sich die einzufügende Datei im selben Ordner wie die Beispieldatei *Bsp21\_01.doc* befindet.

Listing 21.14 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 *Bsp21\_01.doc* und *modSearchProz.bas* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap21*.

### 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 21.15 Versuch per *AddFromString*-Methode eine Prozedur zu erstellen

```

Sub subAddStringToModule1()
    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 21.15 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 21.15 dahingehend um, indem Sie die *InsertLines*-Methode verwenden:

Listing 21.16 Erstellen einer Prozedur mittels der *InsertLines*-Methode

```

Sub subAddStringToModule2()
    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.

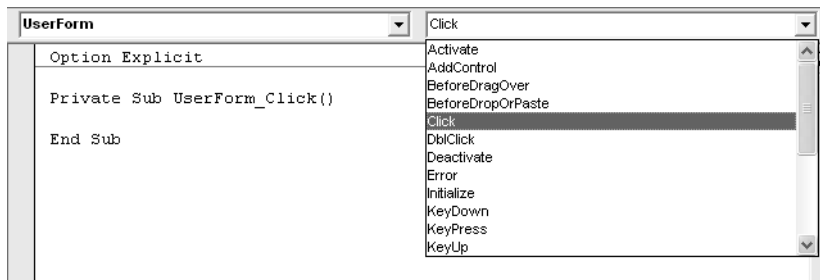
Für Ereignisse, wie sie z.B. in Benutzerformularen ausgeführt werden, steht Ihnen eine besondere Methode zur Verfügung: Die CreateEventProc-Methode.

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 21.18), wenn Sie sich in der Code-Anzeige des UserForms befinden.

**Abbildg. 21.18** Übersicht über die möglichen Ereignisse für Benutzerformulare



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.

Das Beispiel in Listing 21.17 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 21.17 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 21.18), die dann ausgegeben wird.

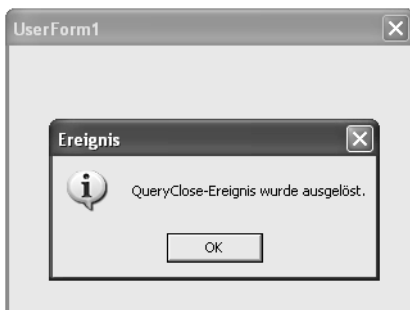
Listing 21.18 Mittels der *CreateEventProc*-Methode erzeugtes Ereignis

```

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 21.19 ausgegeben.

Abbildg. 21.19 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 zum Buch in der Datei \Beispiele\Kap21\Bsp21\_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 21.5 aufgelistet.

Tabelle 21.5 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 Eigenschaftensfenster des jeweiligen Steuerelementes vornehmen können.

Das Makro in Listing 21.19 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 21.19 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)

```

**Listing 21.19** Dynamisch erstelltes Benutzerformular mit Steuerelementen (Fortsetzung)

```

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 21.20 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. 21.20 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 21.20 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 zum Buch in der Datei `\Beispiele\Kap21\Bsp21_01.doc` im Modul `modRemoveVBComponent`.

Listing 21.20 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
    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
End Sub
```

**Listing 21.20** Entfernen einer neu angelegten Komponente (*Fortsetzung*)

```
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 die 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.

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 21.21 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 zum Buch in der Datei `\Beispiele\Kap21\Bsp21_01.doc` im Modul *modShowAllUserforms*.

**Listing 21.21** 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 21.21 erhalten Sie Zugriff auf das UserForm mit normalerweise einer Reihe von Steuerelementen. In Listing 21.19 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)
```

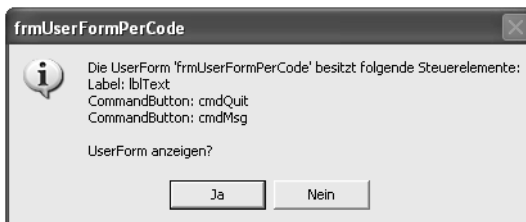
Die Prozedur *subShowUserFormControlInfos* in Listing 21.22 benutzt diese Funktion, um für alle UserForms die enthaltenen Steuerelemente mit Typ und Namen anzuzeigen (Abbildung 21.21), bevor das jeweilige UserForm angezeigt werden kann.

**Listing 21.22** 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. 21.21** 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. 21.22** 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 Objekte (.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.

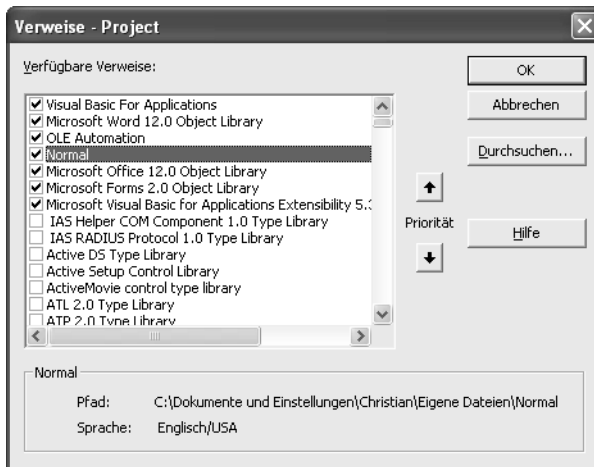


Die in diesem Abschnitt verwendeten Beispiel-Prozeduren finden Sie auf der CD-ROM zum Buch in der Datei \Beispiele\Kap21\Bsp21\_01.doc im Modul *modReferences*.

## 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.

Abbildg. 21.23 Übersicht über die gesetzten und verfügbaren Verweise eines Projektes



Das Listing 21.23 zeigt alle gesetzten Verweise (Abbildung 21.24), 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.

Listing 21.23 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. 21.24 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 21.6 listet die GUID für die wichtigsten Verweis-Bibliotheken auf. Das Listing 21.24 veranschaulicht, wie diese Angaben programmtechnisch zu ermitteln sind; das Ergebnis ist in Abbildung 21.25 ersichtlich.

Tabelle 21.6 GUIDs für die Standardverweise in Office 12 (Office 2007)

Name	GUID/Standardpfad/Beschreibung
VBA	{000204EF-0000-0000-C000-000000000046}
	C:\Programme\Gemeinsame Dateien\Microsoft Shared\VBA\VBA6\VBE6.DLL (Windows 2000 / Windows XP)

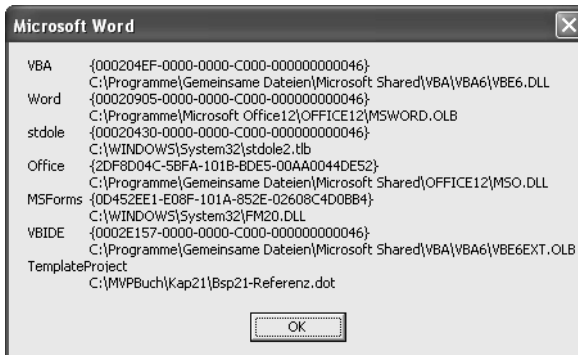
Tabelle 21.6 GUIDs für die Standardverweise in Office 12 (Office 2007) (Fortsetzung)

Name	GUID/Standardpfad/Beschreibung
	C:\Program Files\Common Files\Microsoft Shared\VBA\VBA6\VBE6.DLL (Windows Vista)
	Visual Basic For Applications
Word	{00020905-0000-0000-C000-000000000046}
	C:\Programme\Microsoft Office\OFFICE12\MSWORD.OLB (Windows 2000 / Windows XP)
	C:\Program Files\Microsoft Office\Office12\MSWORD.OLB (Windows Vista)
	Microsoft Word 12.0 Object Library
stdole	{00020430-0000-0000-C000-000000000046}
	C:\WINDOWS\system32\Stdole2.tlb (Windows 2000 / Windows XP)
	C:\Windows\system32\stdole2.tlb (Windows Vista)
	OLE Automation
Office	{2DF8D04C-5BFA-101B-BDE5-00AA0044DE52}
	C:\Programme\Gemeinsame Dateien\Microsoft Shared\OFFICE12\MSO.DLL (Windows 2000 / Windows XP)
	C:\Program Files\Common Files\Microsoft Shared\OFFICE12\MSO.DLL (Windows Vista)
	Microsoft Office 12.0 Object Library
MSForms	{0D452EE1-E08F-101A-852E-02608C4D0BB4}
	C:\WINDOWS\System32\FM20.DLL (Windows 2000 / Windows XP)
	C:\Windows\system32\FM20.DLL (Windows Vista)
	Microsoft Forms 2.0 Object Library
VBAIDE	{0002E157-0000-0000-C000-000000000046}
	C:\Programme\Gemeinsame Dateien\Microsoft Shared\VBA\VBA6\VBE6EXT.OLB (Windows 2000 / Windows XP)
	C:\Program Files\Common Files\Microsoft Shared\VBA\VBA6\VBE6EXT.OLB (Windows Vista)
	Microsoft Visual Basic for Applications Extensibility 5.3

Für die Office-Versionen »Office 2000 (Office 9)«, »Office 2002 (Office XP/Office 10)« und »Office 2003« (Office 11) ändern sich nur die Versionsnummern und Dateinamen der jeweiligen Versionsdateien.

**Listing 21.24** 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. 21.25** Ausgabe der gesetzten Verweise mit Pfad und GUID


Das Listing 21.25 setzt einen neuen Verweis auf die Datei *Bsp21\_Referenz.dot*, sofern dieser noch nicht in der Liste der gesetzten Verweise aufgeführt ist. Damit das Beispiel funktioniert, kopieren Sie bitte die Datei *Bsp21\_Referenz.dot* von der CD-ROM aus dem Ordner *\Beispiele\Kap21\* in den lokalen Ordner *C:\MVPBuch\Kap21*. Wenn Sie die Datei an einen anderen Ort speichern, passen Sie bitte den vollständigen Pfad in Listing 21.25 an den gewählten Speicherort an.

**Listing 21.25** Setzen eines Verweises auf eine Dokumentvorlage

```
Sub subSetReference()
    fktSetReferences "C:\MVPBuch\Kap21\Bsp21_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
```



Listing 21.25 Setzen eines Verweises auf eine Dokumentvorlage (Fortsetzung)

```

    If oRef.FullPath = strRefPath Then
        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

```

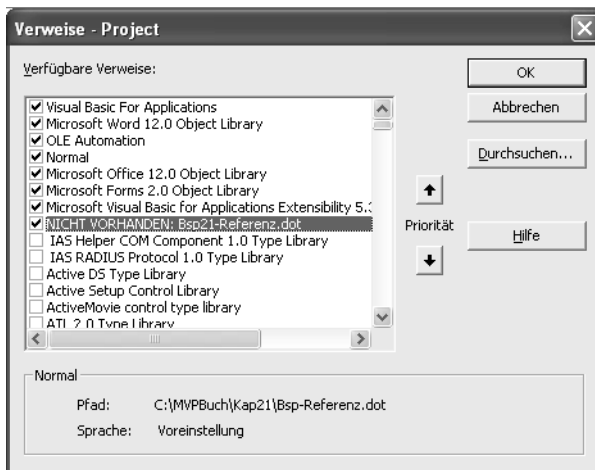
Wenn Sie die AddFromGuid-Methode verwenden möchten, müssen Sie diese GUID kennen, da diese eindeutig und unabhängig von der verwendeten Office-Version ist (siehe Tabelle 21.6).

## 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 die der 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 (siehe Abbildung 21.26). Diese ungültigen Verweise können zu Fehlern in den Makros führen und sogar die Stabilität von Word beeinträchtigen.

Abbildg. 21.26 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 21.26 überprüfen Sie die gesetzten Verweise. Wenn Sie die im Listing 21.25 hinzugefügte Verweisdatei *Bsp21\_Referenz.dot* entfernen oder umbenennen, und das Makro ausführen, erhalten Sie den Hinweis in Abbildung 21.27, dass der Verweis ungültig sei.

**Listing 21.26** Prüfen der Verweise auf Gültigkeit

```
Sub subProofReferences()
    Const c_Ref As String = "Verweis prüfen"
    Dim ret As Integer
    Dim oRef As Reference
    With VBE.ActiveVBProject.References
        For Each oRef In VBE.ActiveVBProject.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. 21.27** 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 21.27 überprüft alle gesetzten Verweise und versucht diese entweder anhand des Dateipfades oder der GUID wieder zu setzen.

Listing 21.27 Reparieren ungültiger Verweise

```

Sub subReSetReferences()
    Const c_Ref As String = "Verweis prüfen/setzen"
    Dim strGUID As String
    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 21.27 dahingehend modifizieren, dass Sie optional den neuen Pfad zur Datei mit angeben. In Listing 21.28 werden alle ungültigen Verweise dahingehend geprüft, ob die angegebene Verweisdatei *Bsp21\_Referenz.dot* betroffen ist. Ist dies der Fall, wird nach Bestätigung (Abbildung 21.28) 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 21.28 Ungültige Verweise ersetzen

```

Sub subTestAndReSetReference()
    fktReSetReferences "G:\MVPBuch\Kap21\Bsp21_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

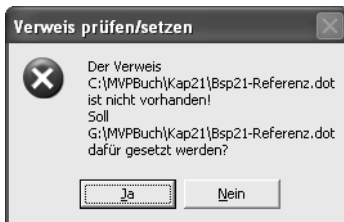
```

**Listing 21.28** Ungültige Verweise ersetzen (Fortsetzung)

```

If IsMissing(strRefPath) = False Then
    If Dir(strRefPath) = "" Then
        MsgBox "Die angegebene Verweisdatei existiert nicht:" & _
            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. 21.28** 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 818) und auf das aktive VBA-Projekt (Seite 819) mit seinen darin enthaltenen Modulen und Benutzerformularen (Seite 821) zugreifen können.
- Ein weiterer Schwerpunkt war der Zugriff auf den VBA-Code eines Projektes (Seite 824) und das Auslesen bestimmter Zeilen (Seite 825 und Seite 826).
- 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 831).
- 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 834), aber auch einzelne Code-Zeilen in ein Projekt einzufügen (Seite 838 und Seite 838).
- Neben dem Einfügen von VBA-Code haben Sie weiterhin gesehen, wie Sie Steuerelemente in ein UserForm einfügen (Seite 842) und ein so erstelltes UserForm zur Laufzeit anzeigen lassen können (Seite 846).
- Abschließend wurde Ihnen in diesem Kapitel gezeigt, wie Sie während der Bearbeitung von Makros benötigte Verweise auf Bibliotheken ermitteln (Seite 849) und ggf. neu setzen können (Seite 850).

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 dieses Buchs 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://msdn2.microsoft.com/en-us/library/default.asp>) im Bereich »Office Development«.



# Teil E

## XML und Smart-Technologien

### In diesem Teil:

<b>Kapitel 22</b>	Word und XML: Eine Einführung	861
<b>Kapitel 23</b>	Smarttags	909
<b>Kapitel 24</b>	Smart Documents	943





## Kapitel 22

# Word 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	884
XML in Word ab Version 2003	890
WordProcessingML außerhalb von Word generieren	907
Zusammenfassung	908

Word 2003 stellte einiges an neuer Funktionalität zur Verfügung, unter anderem Werkzeuge für die Arbeit mit XML. Diese wurden 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 23)
- Als Grundlage für die Smart Document-Funktionalität (siehe Kapitel 24)

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 2003 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.



Word 2007 baut auf die in Word 2003 eingeführte Funktionalität auf. Das standardmäßige Dateiformat ist ab sofort XML, allerdings entspricht es nicht genau dem Dateiformat eines in Word 2003 als XML gespeicherten Dokuments. Das in Word 2003 eingeführte »WordProcessingML« definiert nach wie vor den Dokumentinhalt. Strukturelle Teile wie eingebettete Objekte, Grafiken und Formatvorlagen, werden neu in getrennten »XML-Teilen« hinterlegt; die Verknüpfungen zwischen dem Dokument und diesen ausgelagerten Elementen werden in weiteren Teilen verwaltet und das ganze in einer »ZIP-Packung« zusammengefasst (siehe auch Kapitel 17).

Dieses neue Konzept heißt »OpenXML« und wurde für viele Office 2007-Anwendungen eingeführt. Wir werden das Thema »OpenXML« in diesem Buch nicht näher behandeln. Weitere Informationen dazu finden Sie bei OpenXMLDeveloper.org (<http://openxmldeveloper.org/default.aspx>) sowie im OpenXML-Software-Developers-Kit auf <http://www.microsoft.com/downloads/details.aspx?familyid=ad0b72fb-4a1d-4c52-bdb5-7dd7e816d046&displaylang=en>. Im Gegensatz zum Word 2003-XML-Dateiformat können – sofern das Kompatibilitätspack installiert wurde – alle Word-Versionen bis einschließlich Word 2000 das neue Dateiformat lesen und schreiben.

Anders als für Word 2003 unterstützen alle Versionen von Word 2007 die ganze Palette der in diesem Kapitel beschriebenen XML-Funktionalität. Neu dazu gesellen sich die Inhaltssteuerelemente, die in Kapitel 7 vorgestellt werden.

## 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 22.1 sowie in Listing 22.2 ersichtlich.

**Listing 22.1** Inhalt von *Bsp22\_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 22.2** Inhalt von *Bsp22\_02.xml*: Einige Kontinente mit ihren längsten Flüssen, ebenfalls 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 *Bsp22\_01.htm* sowie *Bsp22\_02.xml* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap22*.

### 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 abgedruckt (inklusive Großschreibung).
3. Führen Sie *Datei/Speichern* aus und wählen Sie einen Speicherort aus.
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 *"Bsp22\_01.htm"* – inklusive der Anführungszeichen. Betätigen Sie die Schaltfläche *Speichern*.
6. Word sollte daraufhin 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 benutzt, 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 22.1 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 22.2, 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 22.1 wurde mit dem Vokabular XHTML, das Listing 22.2 dagegen mit einem anderen – nennen wir es »KML« (K für Kontinente) – geschrieben.

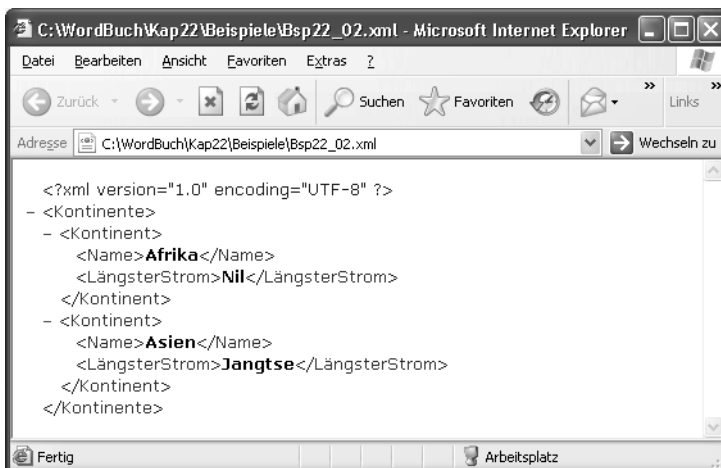
Beide sind also XML-Dateien, und XML-Werkzeuge können damit arbeiten. Unterschiedlich sind sie dennoch, wie die Interpretation der Vokabulare durch den Internet Explorer (siehe Abbildung 22.1 sowie Abbildung 22.2) veranschaulicht.

Abbildg. 22.1 Die XML-Datei *Bsp22\_01.htm*, interpretiert vom Internet Explorer



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. 22.2 Die XML-Datei *Bsp22\_02.xml*, interpretiert vom Internet Explorer



Die »KML«-Datei in Abbildung 22.2 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 hervor, 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 22.3).

**Listing 22.3** *Bsp22\_03.xml*: XML-Kontinente-Datei, die auf eine XSLT-Transform verweist

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="Bsp22_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 von Listing 22.4. Wird *Bsp22\_03.xml* im Internet Explorer geöffnet, muss es ähnlich aussehen wie die Abbildung 22.1.

Listing 22.4 Bsp22\_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>
            <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 Beispieldatei *Bsp22\_03.xml* sowie die Transformationsdatei *Bsp22\_03.xsl* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap22*.

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 also 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 *Bsp22\_01.htm* ein:

```
<?xml-stylesheet type="text/xsl" href="BSP22_04.xsl"?>
```

- Speichern Sie die Datei unter den Namen *Bsp22\_04.xml*.
- Geben Sie den Code aus Listing 22.5 in eine neue Datei ein und speichern Sie diese unter dem Namen *Bsp22\_04.xsl*.
- Öffnen Sie *Bsp22\_04.xml* im Internet Explorer. Das Resultat sollte wie in Abbildung 22.1 aussehen.

Der XSLT-Code transformiert XHTML in gewöhnliches, korrektes HTML.

**Listing 22.5** *Bsp22\_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 veranschaulichte 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 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 sowie 2007 unterstützt diese Konvertierung.)
- Es ist zudem möglich, 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. (Wird nicht von allen Versionen von Word 2003 unterstützt, wie in der Einleitung zu diesem Kapitel beschrieben.)



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 Word 2003 XML-Format kann wahlweise das ganze Dokument (als Word-ProcessingML) 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 24 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.
- In Word 2007 können Inhaltssteuerelemente mit einem XML-Teil verbunden werden.

**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://msdn2.microsoft.com/en-us/library/Aa223586\(office.11\).aspx](http://msdn2.microsoft.com/en-us/library/Aa223586(office.11).aspx))
- 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://www.microsoft.com/downloads/details.aspx?familyid=A56446B0-2C64-4723-B282-8859C8120DB6&displaylang=en>)

## XML-Bestandteile

Die XML-Sprache allein genügt nicht, um nützliche Lösungen bereitzustellen. Im Abschnitt »XML-Daten transformieren« in diesem Kapitel sahen wir beispielsweise, dass es einer Transformation bedarf, um die Daten auf benutzerfreundliche Art darzustellen. Tabelle 22.1 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 22.1** 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.
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 22.4 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 22.6 veranschaulicht.

**Listing 22.6** 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.«

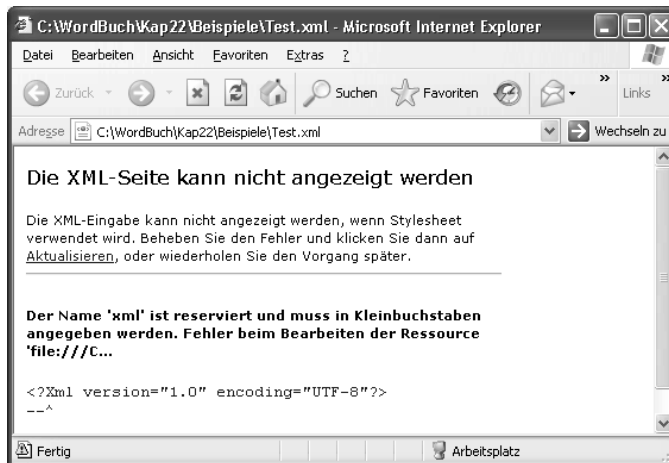
Ein Dokument, das allen Regeln der XML-Syntax nachkommt, wird als *wohlgeformt* betrachtet. Ist ein Dokument nicht wohlgeformt, ist es auch kein XML-Dokument und wird von einem XML-Parser abgelehnt, wie die Abbildung 22.3 veranschaulicht.

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. 22.3

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 22.1 ist wohlgeformt und gültig, weil sie die Regeln des

XHTML DTD befolgt. Die XML-Datei in Listing 22.2 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 22.7, das das Vokabular der »KML«-Datei aus Abschnitt »XML-Vokabulare« umschreibt. Es veranschaulicht viele Grundlagen der XSD.

**Listing 22.7** *Bsp22\_05.xsd*: XML-Schema für die »KML«-Datei (Listing 22.2). 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">
      <xsd:maxLength value="20" />
    </xsd:restriction>
  </xsd:simpleType>
</xsd:schema>
```



Die Beispieldatei *Bsp22\_05.xsd* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap22*.

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 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 22.8 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 22.8** *Bsp22\_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:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```



Die Beispieldatei *Bsp22\_06.xsd* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap22*.

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 Ägyptens, 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. 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 22.9 veranschaulichen.

**Listing 22.9** *Bsp22\_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.KAP22Beispiel.com/BSP07"
  xmlns:bu="http://www.KAP22Beispiel.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 *Bsp22\_07.xsd* finden Sie auf der CD-ROM zum Buch im Ordner `\Beispiele\Kap22`.

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` angehört:

```
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.KAP22Beispiel.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.KAP22Beispiel.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 22.10 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 22.10** *Bsp22\_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.KAP22Beispiel.com/BSP08"
  xmlns:cd="http://www.KAP22Beispiel.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>

  <xsd:simpleType name="CDIDTyp" >
    <xsd:restriction base="xsd:integer" >
      <xsd:minInclusive value="2000" />
      <xsd:maxInclusive value="2999" />
    </xsd:restriction>
  </xsd:simpleType>
```



Die Beispieldatei *Bsp22\_08.xsd* finden Sie auf der CD-ROM zum Buch im Ordner `\Beispiele\Kap22`.

Jetzt werden die zwei Schemas in einem »Überschema« vereint, wie Listing 22.11 veranschaulicht.

**Listing 22.11** *Bsp22\_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.KAP22Beispiel.com/BSP09"
  xmlns:bu="http://www.KAP22Beispiel.com/BSP07"
  xmlns:cd="http://www.KAP22Beispiel.com/BSP08"
  elementFormDefault="qualified">

  <xsd:import
    namespace="http://www.KAP22Beispiel.com/BSP07"
    schemaLocation="Bsp22_07.xsd"/>
  <xsd:import
    namespace="http://www.KAP22Beispiel.com/BSP08"
    schemaLocation="Bsp22_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 *Bsp22\_09.xsd* finden Sie auf der CD-ROM zum Buch im Ordner `\Beispiele\Kap22`.

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 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 (Schema), 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 22.12, die lediglich das Wurzelement *Sammlung* enthält. Diese Datei ist sowohl wohlgeformt als auch gültig.

**Listing 22.12** *Bsp22\_10.xml*: XML-Dokument mit einer leeren Sammlung

```
<?xml version="1.0" encoding="UTF-8" ?>
<Sammlung
  xmlns="http://www.KAP22Beispiel.com/BSP09" >
</Sammlung>
```

#### HINWEIS

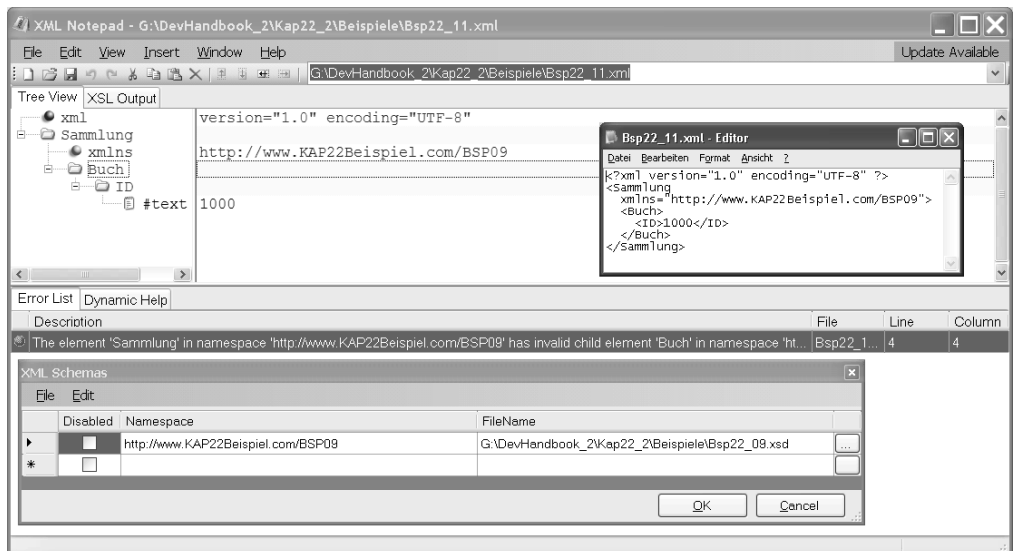
Da kein Namensraumpräfix Teil der *xmlns*-Deklaration ist, wird dieser Namensraum als standardmäßiger der XML-Datei in Listing 22.12 betrachtet. Der XML-Parser wird annehmen, dass alle Elemente ohne Namensraumpräfix (unqualified) diesem Namensraum angehören.

Wird diese Datei um ein *Buch*-Element und seine ID erweitert, wie in Listing 22.13, wird es ungültig, weil das Element *Buch* nicht dem standardmäßigen Namensraum (dem *Sammlung*-Schema) angehört (Abbildung 22.4).

**Listing 22.13** *Bsp22\_11.xml*: XML-Dokument mit einem *Buch*-Element; erster Versuch

```
<?xml version="1.0" encoding="UTF-8" ?>
<Sammlung
  xmlns="http://www.KAP22Beispiel.com/BSP09" >
  <Buch>
    <ID>1000</ID>
  </Buch>
</Sammlung>
```

**Abbildg. 22.4** Die XML-Datei (oben) ist nicht gültig (Fehlermeldung auf der Registerkarte *Error List*), wenn sie mit dem Schema (unten) geprüft wird



**HINWEIS      Nützliche Werkzeuge für die Arbeit mit Schemas**

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 die in diesem Kapitel abgebildeten »XML Notepad« (<http://www.microsoft.com/downloads/details.aspx?familyid=72d6aa49-787d-4118-ba5f-4f30fe913628&displaylang=en>) und Architag »XRay XML Editor« (<http://architag.com/xray/>). Sie prüfen dynamisch die Wohlgeformtheit eines Fensterinhalts und zeigen an, wo der Fehler liegt. Zudem prüfen sie die Gültigkeit einer XML-Datei mit einem geöffneten Schema, wie in Abbildung 22.4 und Abbildung 22.5 ersichtlich. (Das Architag-Produkt 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.)

Falls Sie Visual Studio 2005 (oder später) haben, lohnt es sich, die mitgelieferten Werkzeuge anzuschauen. Im Menü *XML* stehen Befehle, um ein Schema zu generieren, sowie das Ergebnis einer Transformation zu betrachten.

Der nächste Versuch, in Listing 22.14, 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 22.14** *Bsp22\_12.xml*: XML-Dokument mit Namensraum und -präfix für das *Buch*-Element. Diese Datei ist gültig.

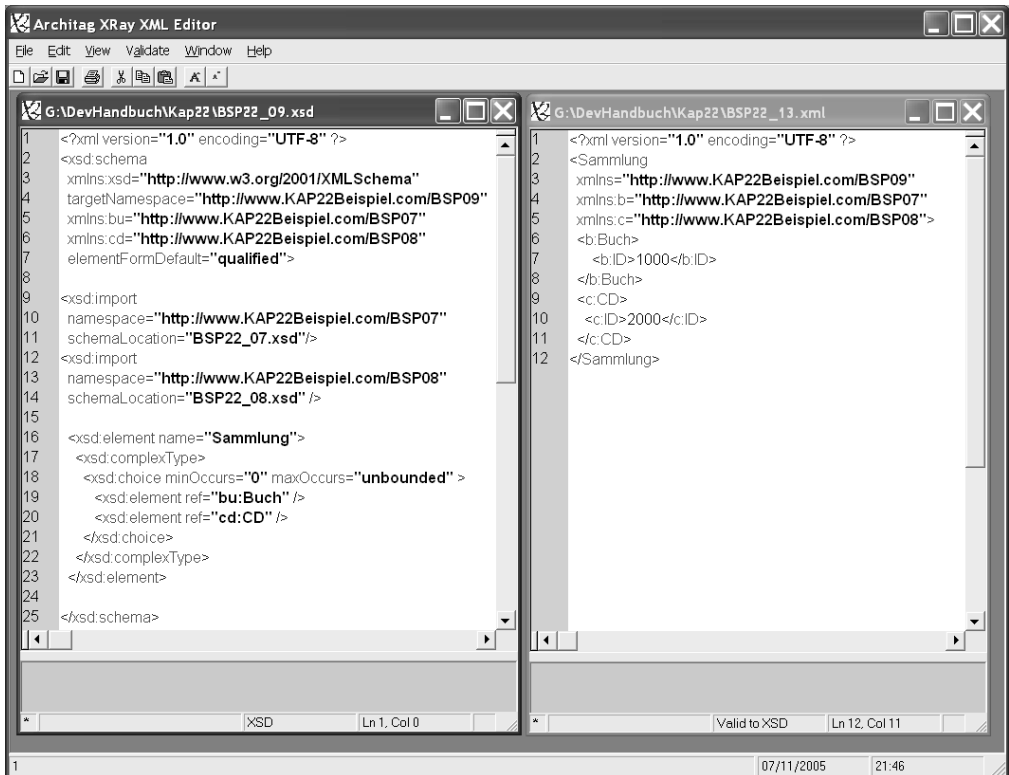
```
<?xml version="1.0" encoding="UTF-8" ?>
<Sammlung
  xmlns="http://www.KAP22Beispiel.com/BSP09"
  xmlns:b="http://www.KAP22Beispiel.com/BSP07">
  <b:Buch>
    <b:ID>1000</b:ID>
  </b:Buch>
</Sammlung>
```

Schließlich veranschaulichen Listing 22.15 und Abbildung 22.5 ein gültiges XML-Dokument mit *Buch*- sowie *CD*-Element.

**Listing 22.15** *Bsp22\_13.xml*: Gültiges XML-Dokument mit je einem *Buch*- sowie *CD*-Element

```
<?xml version="1.0" encoding="UTF-8" ?>
<Sammlung
  xmlns="http://www.KAP22Beispiel.com/BSP09"
  xmlns:b="http://www.KAP22Beispiel.com/BSP07"
  xmlns:c="http://www.KAP22Beispiel.com/BSP08" >
  <b:Buch>
    <b:ID>1000</b:ID>
  </b:Buch>
  <c:CD>
    <c:ID>2000</c:ID>
  </c:CD>
</Sammlung>
```

**Abbildg. 22.5** »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 durchgearbeitet 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>. Zudem befinden sich einfache Beispiele in Kapitel 13 und Kapitel 15. In den Beispieldateien zum Thema Inhaltssteuerelemente (Kapitel 7) finden Sie aufwändigere Codebeispiele.

---

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 und Word 2007 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://msdn2.microsoft.com/en-us/library/ms763742.aspx>. Die allgemeine Version des MSXML-SDK steht zum Herunterladen unter <http://www.microsoft.com/downloads/details.aspx?FamilyID=2cf40ae6-368c-4b6b-a185-2dfa92fb7993&DisplayLang=en> zur Verfügung.

---

Das .NET Framework stellt System.XML-Klassen für die Arbeit mit XML-Dokumenten, XML-Schemas, XSLT, XPath und XML-DOM zur Verfügung. Diese sind in der MSDN-Bibliothek dokumentiert.

Das Word-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 dyna-

misch 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, die gegenwärtigen .NET Framework-Klassen sowie Word 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 unter <http://www.obqo.de/w3c-trans/xpath-de>.

Es gibt im Internet ein kostenloses Werkzeug zum Herunterladen, das Sie bei der Arbeit mit XPath unterstützt: »SketchPath« auf <http://pgfearo.googlepages.com>.

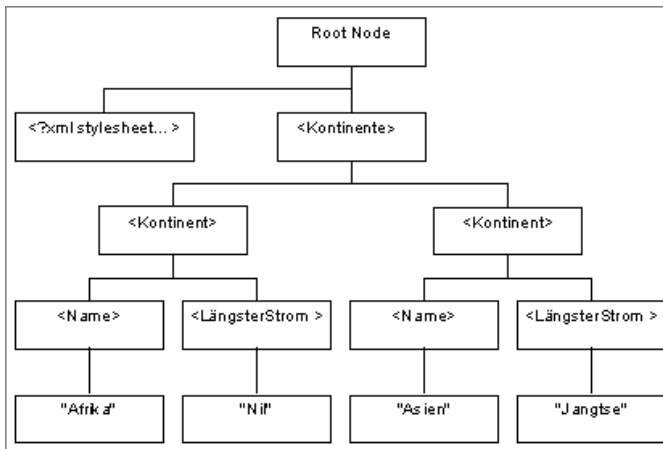
---

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 22.6).

Abbildg. 22.6 Hierarchie der Knotenpunkte des XML-Dokuments *Kontinente*

Die XSLT in Listing 22.3 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 22.16 veranschaulicht. Darin basiert die Transformation des Kontinente-XML-Dokuments auf Mustervergleichen statt auf programmtechnischen Konstrukten.

**Listing 22.16** *Bsp22\_14.xsl*: Eine Transformation des Kontinent-XML-Dokuments, basierend auf Muster vergleichen

```

<?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>

```

**Listing 22.16** *Bsp22\_14.xsl*: Eine Transformation des Kontinent-XML-Dokuments, basierend auf Muster vergleichen (Fortsetzung)

```
<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 *Bsp22\_14.xsl* mit der Transformation sowie *Bsp22\_15.xml*, die auf diese hinweist, finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap22*.

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 22.6), 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 22.1) 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://msdn2.microsoft.com/en-us/library/Aa203691\(office.11\).aspx](http://msdn2.microsoft.com/en-us/library/Aa203691(office.11).aspx). 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 anderem:

- »XML Pocket Consultant«, Autor William R. Stanek, Microsoft Press, ISBN 0-7356-1183-1 (englisch)

# XML in Word ab Version 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 die Word-Dokumentstruktur einzubetten. Dieser Abschnitt bietet einen Überblick des WordProcessingML-Vokabulars und stellt die Werkzeuge für die Arbeit mit Word-fremden XML-Vokabularen vor.



Die in diesem Abschnitt vorgestellten Grundlagen gelten ebenfalls für Word 2007, mit den in der Kapiteleinleitung aufgeführten Vorbehalten bezüglich des standardmäßigen Dateiformats. Word 2007 kann Dokumente in das XML-Dateiformat von Word 2003 speichern und solche Dateien auch öffnen.

## 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 22.17; wie Word dieses auf dem Bildschirm darstellt, sehen Sie in Abbildung 22.7.

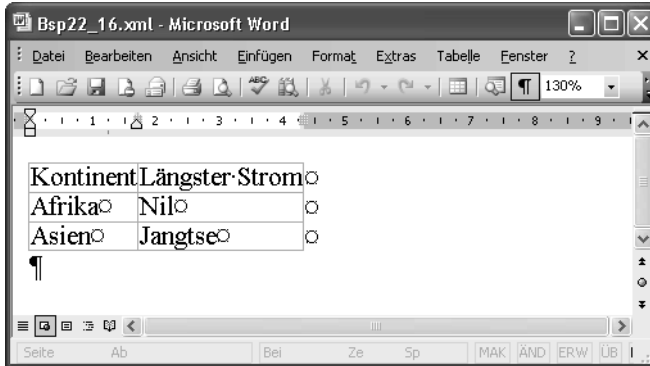
**Listing 22.17** Bsp22\_16.xml: Die *Kontinente*-Tabelle in WordProcessingML

```
<?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:p>
</w:body>
</w:wordDocument>
```



Die Beispieldatei *Bsp22\_16.xml* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap22*.

Abbildg. 22.7 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 22.18; 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 *Bsp22\_17.xml* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap22*.

Listing 22.18 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 22.17. Sie alle (siehe Tabelle 22.2) werden von Word automatisch hinzugefügt, egal ob das Dokument passende Objekte beinhaltet oder nicht.

**Tabelle 22.2** 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 Dateinhalt 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 Essentials« 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 22.17 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 Formatierungsbefehl 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

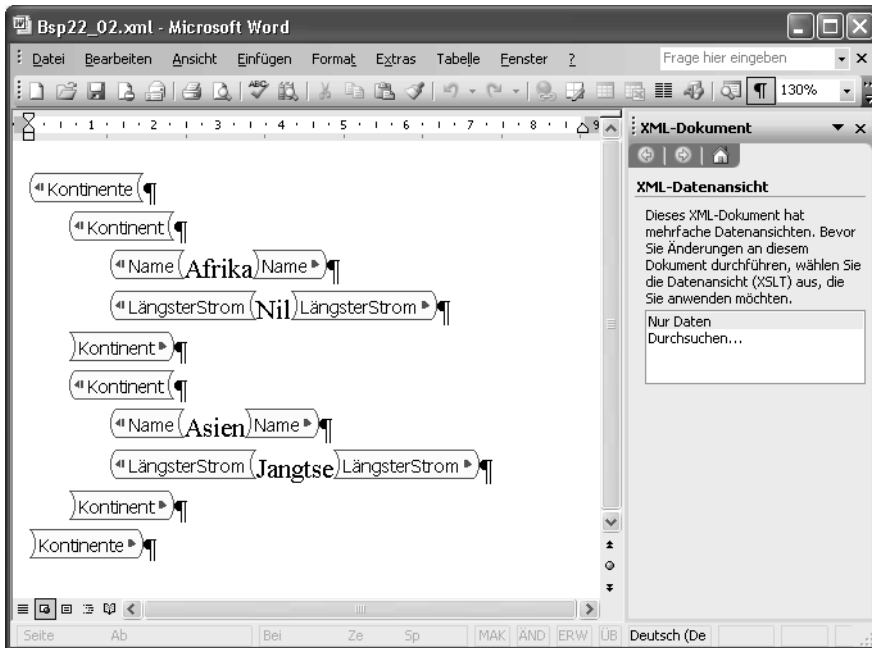
Zusätzlich zum WordProcessingML-Dateiformat enthält Word 2003 Schnittstellen, um andere XML-Vokabulare mit ihren Schemas, Namensräumen und Transformationen in Word-Dokumente zu integrieren. Diese Funktionalität ist zum größten Teil an den Entwickler gerichtet, um Smart Document-Lösungen (siehe Kapitel 24) bereitzustellen. Sie macht keinen benutzerfreundlichen XML-Editor aus Word.

Wenn Sie die Beispieldatei für Listing 22.2 (*Bsp22\_02.xml* auf der Buch-CD) in Word öffnen, wird das Ergebnis ähnlich wie in Abbildung 22.8 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.

Klicken Sie auf den Pfeil neben dem Titel des Aufgabenbereichs und wählen Sie in der Liste weiterer verfügbarer Aufgabenbereiche den Eintrag *XML-Struktur* aus.

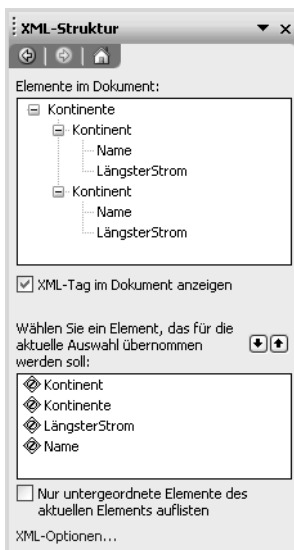
Dieser Aufgabenbereich (Abbildung 22.9) ist in zwei Abschnitte unterteilt: Der obere zeigt eine hierarchische Auflistung der im Dokument *enthaltenen* Elemente sowie ein Kontrollkästchen, womit diese ein- und ausgeblendet werden können; der untere listet die *verfügbaren* Elemente auf. Falls das XML-Dokument 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.)

Abbildg. 22.8 Die in Word geöffnete »KML«-Datei



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. 22.9 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 mithilfe 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 22.19 für das »KML«-Vokabular geladen und mit einem Dokument (im vorgestellten Szenario mit der Beispieldatei *Bsp22\_02.xml*) verbunden. (Dieses Schema unterscheidet sich von den vorangehenden durch die Deklaration eines `targetNamespace` und die Unterstützung von gemischtem Inhalt (`mixed="true"`).)

**Listing 22.19** *Bsp22\_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.KAP22Beispiel.com/BSP18"
  elementFormDefault="qualified">
  <xsd:element name="Kontinente">
    <xsd:complexType mixed="true">
      <xsd:sequence>
        <xsd:element name="Kontinent" maxOccurs="unbounded" >
          <xsd:complexType>
            <xsd:sequence>
```

Listing 22.19 Bsp22\_18.xsd: Ein XML-Schema für das »KML«-Vokabular (Fortsetzung)

```

        <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>

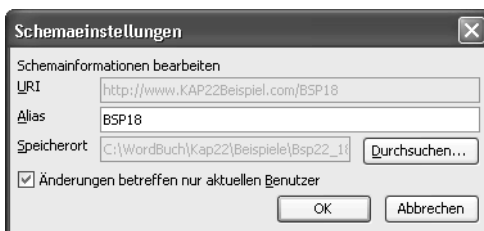
```



Die Beispieldatei *Bsp22\_18.xsd* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap22*.

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 *Bsp22\_02.xml* aus Listing 22.2). Klicken Sie rechts neben dem ersten Tag (Kontinente).
2. Aktivieren Sie nach Aufruf des Menübefehls *Extras/Vorlagen und Add-Ins* die Registerkarte *XML-Schema*. (In Word 2007 erreichen Sie das Dialogfeld über die Schaltfläche *Dokumentvorlage* auf der Registerkarte *Entwicklertools*.)
3. Klicken Sie auf die Schaltfläche *Schema hinzufügen*, um ein Schema in die Liste aufzunehmen (Abbildung 22.10). Navigieren Sie zum Ordner und wählen Sie die \*.xsd-Datei aus.

Abbildg. 22.10 Das Dialogfeld *Schemaeinstellungen*

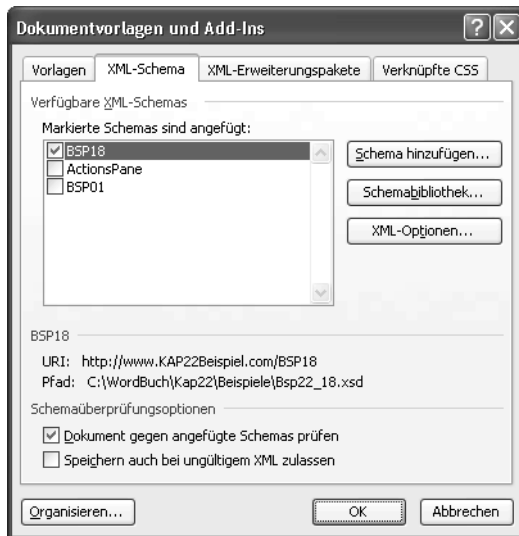
4. Bei der Aufforderung für ein »Alias« (Namensraumpräfix) geben Sie einen kurzen, beschreibenden Ausdruck ein.

**HINWEIS**

Word wird beim Speichern des XML-Dokuments das eingegebene Namensraumpräfix nicht berücksichtigen und wird ein eigenes generieren (wie beispielsweise *ns0*). Das von Ihnen eingegebene dient nur der Anzeige im Aufgabenbereich *XML-Struktur*.

5. Das Namensraumpräfix wird der Schemabibliothek (Abbildung 22.11) 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. 22.11 Schemas werden auf der Registerkarte *XML-Schema* mit einem Dokument verbunden



#### HINWEIS

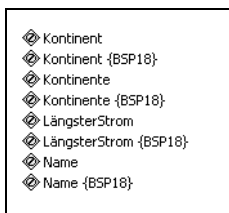
Die eigentliche Schemabibliothek, in der Schemas und Lösungen verwaltet werden, wird durch Anklicken der Schaltfläche *Schemabibliothek* erreicht (Abbildung 22.18). In der oberen Hälfte können Schemas hinzugefügt, entfernt sowie andere Namensraumpräfixe zugewiesen werden.

Beim Betrachten des Aufgabenbereichs *XML-Struktur* wäre zu erwarten, dass der Eintrag *Kontinent* in der unteren Liste steht, da laut Schema nach Kontinent 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 22.12 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 *Bsp22\_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.KAP22Beispiel.com/BSP18* in *Bsp18*).

Abbildg. 22.12 Liste der verfügbaren Elemente im Aufgabenbereich *XML-Struktur*



Es ist auch möglich, durch ein Makro wie in Listing 22.20 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 22.20** Den Namensraum *URI* in einem Dokument ersetzen

```
Sub NamensraumAnpassen()
    Const strURI = "http://www.KAP22Beispiel.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 *Bsp22\_19.doc* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap22*.

Wenn Sie nun das XML-Dokument unter einem anderen Namen speichern (beispielsweise *Bsp22\_02a.xml*), schließen und dann im Windows-Editor öffnen, erkennen Sie, dass der Namensraum *URI http://www.KAP22Beispiel.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 2003 können Sie beispielsweise die »Word XML Toolbox« benutzen (dieses Werkzeug funktioniert nicht in Word 2007). Es handelt sich um ein im .NET Framework 1.1 geschriebenes Add-In, das auf der Microsoft-Webseite unter <http://www.microsoft.com/downloads/details.aspx?FamilyID=a56446b0-2c64-4723-b282-8859c8120db6> heruntergeladen werden kann.

Eine Beschreibung der »Toolbox« (in englischer Sprache) finden Sie unter [http://msdn2.microsoft.com/en-us/library/bb226699\(office.11\).aspx](http://msdn2.microsoft.com/en-us/library/bb226699(office.11).aspx).

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. ▶

**PROFITIPP**

Das Schema *Bsp22\_20.xsd* wurde mit diesem Werkzeug für das XML-Dokument *Bsp22\_02.xml* erstellt. (Es handelt sich um die einzige Beispieldatei dieses Kapitels, die eine UTF-16-Codierung hat.) *Bsp22\_21.xml* ist die passende, von der Toolbox erstellte XML-Datei; die WordProcessingML-Version finden Sie in der Datei *Bsp22\_22.xml* (alle befinden sich im Ordner *\Beispiele\Kap22*).

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 22.13).

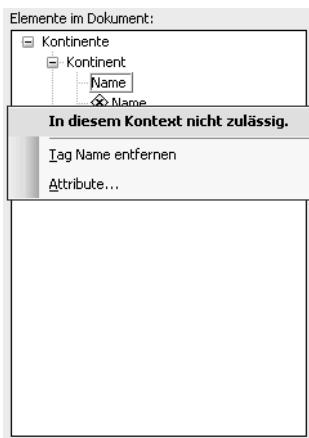
Standardmäßig kann ein ungültiges XML-Dokument nicht gespeichert werden. Wollen Sie es trotzdem speichern, muss entweder

- Die Verbindung zum Schema getrennt werden
- In *Extras/Vorlagen und Add-Ins/XML-Schema* das Kontrollkästchen *Speichern auch bei ungültigem XML zulassen* aktiviert werden
- 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. 22.13** Kontextfehler, weil ein »Name«-Element nicht in einem »Name« verschachtelt sein darf



## Transformationen und Lösungen (Solutions)

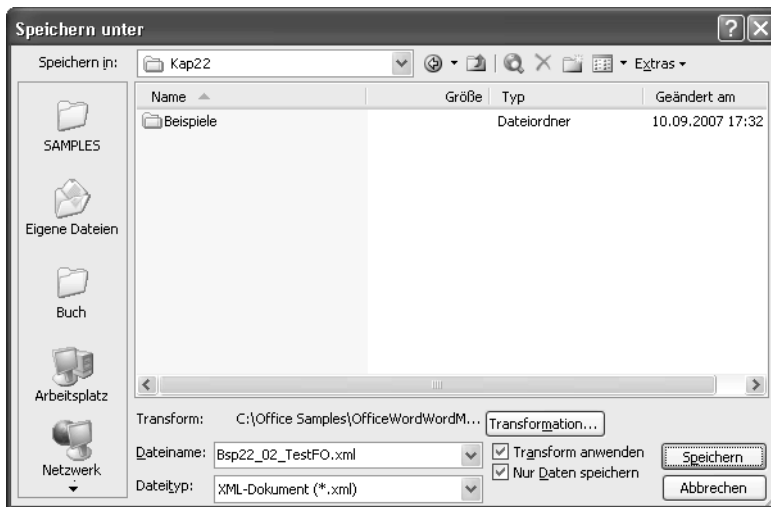
Beim Öffnen oder Speichern eines XML-Dokuments – sowohl eines im Format WordProcessingML wie auch »Nur Daten« – als XML-Dokument kann der Inhalt gleichzeitig transformiert werden. Das geöffnete Dokument sieht damit möglicherweise ganz anders aus als die ursprüngliche Datei, bzw. das gespeicherte Ergebnis 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 22.14)
5. Zur XSL-FO-Transformation (siehe den Abschnitt »XSLT« in diesem Kapitel) navigieren und sie anwählen

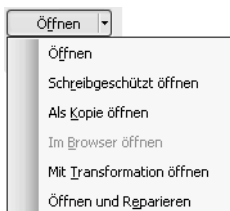
**Abbildg. 22.14** Im Dialogfeld *Speichern unter* kann ein Dokument als XML gespeichert und gleichzeitig transformiert werden



**WICHTIG** Um Datenverluste zu vermeiden, speichern Sie das Dokument als Word-Dokument ab, bevor Sie eine Transformation durchführen.

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 22.15). (Die Transformation eines WordProcessingML-Dokuments in ein anderes WordProcessingML-Dokument ist normalerweise recht komplex.)

**Abbildg. 22.15** 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:

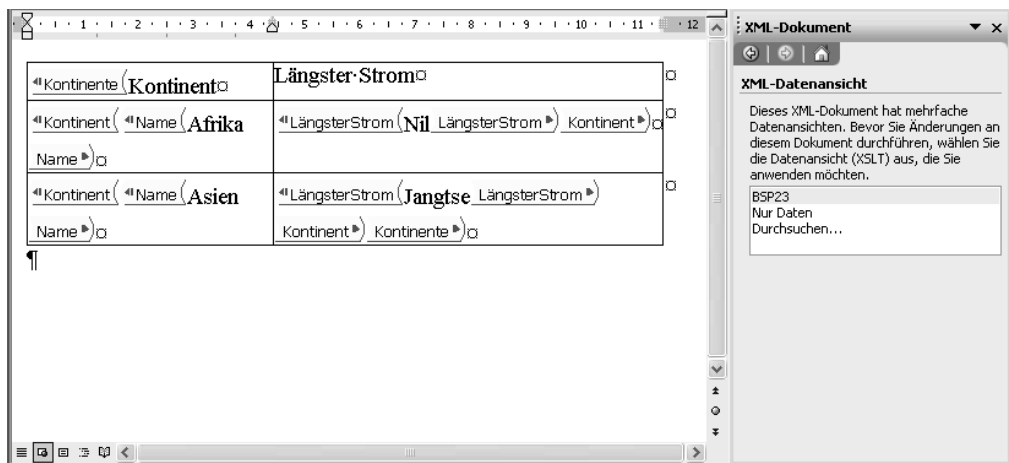
1. Öffnen Sie das XML-Dokument, das im Abschnitt »XML-Vokabulare« in diesem Kapitel vorgestellt wurde (*Bsp22\_02.xml*).
2. Im Aufgabenbereich *XML-Dokument* klicken Sie auf den Eintrag *Durchsuchen*.
3. Navigieren Sie zum Ordner mit den CD-Dateien und wählen Sie eine Transformation (beispielsweise *Bsp22\_03.xsl*), die im Abschnitt »XML-Daten transformieren« beschrieben wurde.

Sie sollten eine ähnliche Tabelle wie in Abbildung 22.1 sehen. Wenn Sie die gleiche Transformation mit dem im Abschnitt »Die Word-Schemabibliothek« weiter vorne in diesem Kapitel erstellten Dokument (*Bsp22\_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 22.21 veranschaulicht eine Transformation, die dieses XML-Dokument in ein WordProcessingML-Dokument transformiert, wie in Abbildung 22.16. Sie deklariert sowohl den Namensraum des *Bsp22\_18.xsd*-Schemas als auch den für das Word-Vokabular:

```
xmlns:w="http://schemas.microsoft.com/office/word/2003/wordml"
xmlns:ns2="http://www.KAP22Beispiel.com/BSP18"
```

**Abbildg. 22.16** Ein in WordProcessingML transformiertes XML-Dokument



**Listing 22.21** 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.KAP22Beispiel.com/BSP18">
  <xsl:output method="xml" encoding="UTF-8" standalone="yes" />
```

Listing 22.21 Transformation, um aus einem »KML«- ein WordProcessingML-Dokument zu erzeugen (Fortsetzung)

```

<xsl:template match="/">
  <w:wordDocument
    xmlns:w="http://schemas.microsoft.com/office/word/2003/wordml"
    xmlns:ns2="http://www.KAP22Beispiel.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>
    <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>

```

**Listing 22.21** Transformation, um aus einem »KML«- ein WordProcessingML-Dokument zu erzeugen (Fortsetzung)

```
<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 *Bsp22\_23.xsl* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap22*.

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.KAP22Beispiel.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 die -Taste. Word müsste eine neue Tabellenzeile generieren, die die Elemente für einen Kontinent enthält.

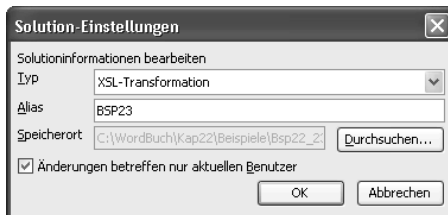
Um dem Anwender die Auswahl und die Zuweisung einer Transformation zu erleichtern, können Transformationen mit Schemas verbunden werden. Diese werden im Abschnitt *Solution zum Schema* der Schemabibliothek verwaltet. Die mit einem Schema assoziierten Transformationen erscheinen automatisch in der Liste *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 *Bsp22\_23.xsl* wie folgt der Schemabibliothek hinzu:

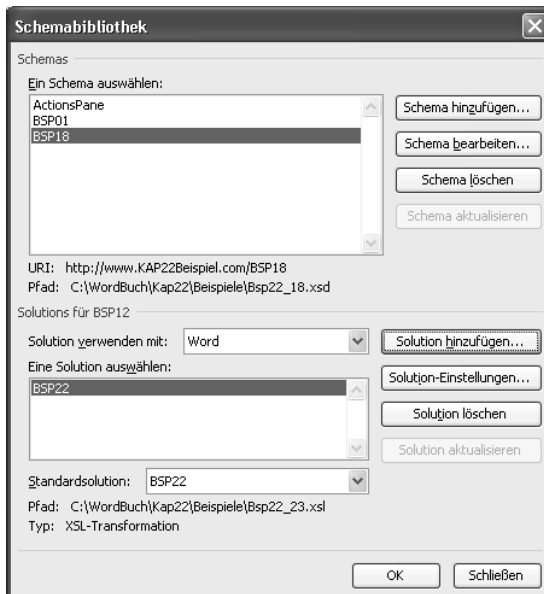
1. Blenden Sie das Dialogfeld *Schemabibliothek* über die Menüfolge *Extras/Vorlagen und Add-Ins/XML-Schema* ein.
2. Wählen Sie das Schema (*Bsp18*) aus der Liste.
3. Klicken Sie die Schaltfläche *Solution hinzufügen* an.
4. Navigieren Sie zum Beispiel-Ordner, wählen Sie die Transformation (*Bsp22\_23.xsl*) und betätigen Sie dann *Öffnen*. Das Dialogfeld *Solution-Einstellungen* (Abbildung 22.17) wird eingeblendet.
5. Geben Sie ein Namensraumpräfix (»Alias«) in das *Alias*-Feld ein (falls nicht, erstellt Word eine GUID und benutzt diese).

**Abbildg. 22.17** Beim Einbinden einer Transformation in die Schemabibliothek ein Namensraumpräfix zuweisen



Beachten Sie in Abbildung 22.18, 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. 22.18** 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 »Word-ProcessingML 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 *Bsp22\_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 (*Bsp22\_24.xml* beispielsweise) und schließen
4. Das Fenster der Eingabeaufforderung über die Befehlsfolge *Start/Alle Programme/Zubehör* einblenden
5. Das »Inference Tool« auf die Datei (mit voller Pfadangabe) ausführen

```
wml2xslt c:\WordBuch\Beispiele\KAP22\Bsp22_24.xml
```

Das Resultat wird eine XSL-Datei sein (im beschriebenen Fall *Bsp22\_24.xsl*). Bitte beachten Sie, dass dieses Werkzeug eher als Hilfsmittel zu betrachten ist. Oft werden Sie nicht herumkommen, die Transformation manuell 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 Codefragment lädt das Beispiel-Schema mit dem Namensraum »BSP18« in die Schemabibliothek:

```
Dim objSchema As Word.XMLNamespace
Set objSchema = Application.XMLNamespaces.Add(
    Path:="C:\CD-ROM\Beispiele\Kap22\Bsp22_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 22.22 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 *Bsp22\_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. 22.19 XML in WordProcessingML transformieren – ohne Word

```
Sub XML2WML()
    'Passen Sie die Pfadangaben Ihrem System an
    Const strPath = "C:\CD-ROM\Beispiele\Kap22\"
    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 & "\Bsp22_23.xsl") Then
        MsgBox "XSL-Transformationsdatei konnte nicht gefunden werden"
```

**Abbildg. 22.19** 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 & "\Bsp22_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 & "\Bsp22_26.xml"
            End If
        End If
    End If
End If

Set objWordMLDocument = Nothing
Set objXMLDocument = Nothing
Set objXSLTransform = Nothing
End Sub

```



Die Beispieldatei *Bsp22\_25.doc* sowie *Bsp22\_25.txt* mit dem Code finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap22*.

## 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 907 ff.) veranschaulicht, wie ein WordProcessingML-Dokument unabhängig von der Word-Anwendung erstellt werden kann.



## Kapitel 23

# Smarttags

### **In diesem Kapitel:**

Die Arbeit mit Smarttags	910
Die Entwicklung von Smarttags	914
Zusammenfassung	942

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 und Word 2007. Zudem wird anhand von Beispielen gezeigt, wie Sie Smarttags für diese Word-Versionen 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 Liste« (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 23.1 vorgestellte XML-Dokument enthält die Information für das Smarttag-Beispiel. Speichern Sie es im folgenden Ordner:

*C:\Programme\Gemeinsame Dateien\Microsoft Shared\Smart Tag\LISTS*

### HINWEIS

Informationen zum Erstellen und Umgang mit XML-Dokumenten finden Sie in Kapitel 22.

Listing 23.1

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>
```



Benutzen Sie die Beispieldatei *planet.xml* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap23*.

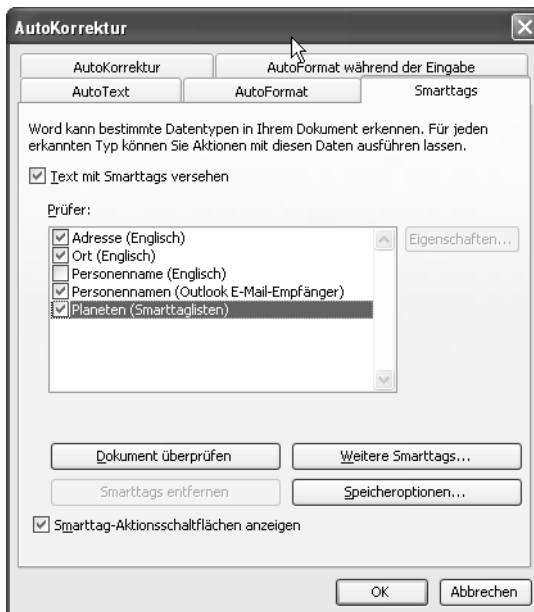
Starten Sie Word und stellen Sie sicher, dass ein neues, leeres Dokument geöffnet ist. Rufen Sie dann den Menübefehl *Extras/AutoKorrektur-Optionen* auf und holen Sie die Registerkarte *Smarttags* in den Vordergrund (Abbildung 23.1). Aktivieren Sie den Eintrag »Planeten (Smarttaglisten)« in der Liste *Prüfer* und klicken Sie auf *OK*.



2007

In Word 2007 wird das Dialogfeld über die *Word-Optionen* in der Kategorie *Dokumentprüfung* erreicht. Klicken Sie dort auf die Schaltfläche *AutoKorrektur-Optionen*.

Abbildg. 23.1 Das Dialogfeld, um Smarttag-Optionen zu verwalten



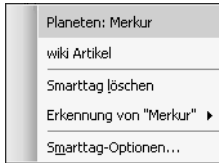
Im Dokument geben Sie den Text »Merkur« ein, gefolgt von einem Leerzeichen. Nach einem kurzen Augenblick wird das Wort von einer feinen, gepunkteten Linie unterstrichen. Wenn Sie den Mauszeiger über dem Wort positionieren oder hineinklicken, erscheint ein Smarttag-Symbol, wie in Abbildung 23.2 ersichtlich.

Abbildg. 23.2 Text, der von einem Smarttag erkannt und markiert wurde



Ein Klick auf das Symbol (oder Drücken der Tastenkombination  $\boxed{\text{Alt}} + \boxed{\text{⇧}} + \boxed{\text{F10}}$ ) blendet die Liste der Aktionen und weitere Optionen für das Smarttag ein (Abbildung 23.3).

Abbildg. 23.3 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

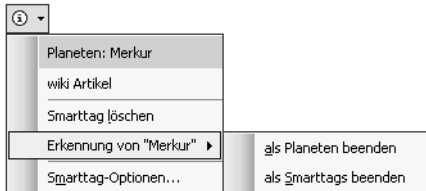
Mehr als ein Satz Aktionen kann mit einem Smarttag-Typ verbunden sein. Würden Sie beispielsweise *planet.xml* kopieren, umbenennen und in den gleichen Ordner mit der ursprünglichen Datei speichern, enthielte das Menü zwei identische Aktionen-Einträge: für jede der XML-Dateien einen.

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 23.4 ersichtlich.

Abbildg. 23.4 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:

Unter Windows XP

```
<Laufwerk>:\Dokumente und Einstellungen\<Benutzername>\Anwendungsdaten\Microsoft\Smart Tags\Exceptions\
```

Unter Windows Vista

```
<Laufwerk>:\Users\<Benutzername>\AppData\Roaming\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*



2007

In Word 2007 befinden sich diese Befehle in der Kategorie *Erweitert* der *Word-Optionen*, und zwar im Abschnitt *Genauigkeit beim Freigeben dieses Dokuments beibehalten*.

Smarttags werden standardmäßig im Dokument eingebettet, wenn dieses im Format *\*.doc*, *\*.docx* 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://msdn2.microsoft.com/en-us/library/Aa169576\(office.11\).aspx](http://msdn2.microsoft.com/en-us/library/Aa169576(office.11).aspx). Die Installationsdatei *mstagsdk.msi* befindet sich auch auf der CD-ROM zum Buch im Ordner `\Beilagen\SmartTagSDK`.

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://de.wikipedia.org/wiki/Kontextfreie\\_Grammatik](http://de.wikipedia.org/wiki/Kontextfreie_Grammatik) näher erklärt. Informationen zur Syntax für Smarttags finden Sie im Smarttag-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
- 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. Zudem kann eine VSTO 2005-Lösung dokumentspezifische Smarttags definieren, wie der Abschnitt »VSTO-Smarttags« veranschaulicht. 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 und 2007.

## 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.mspx>.

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

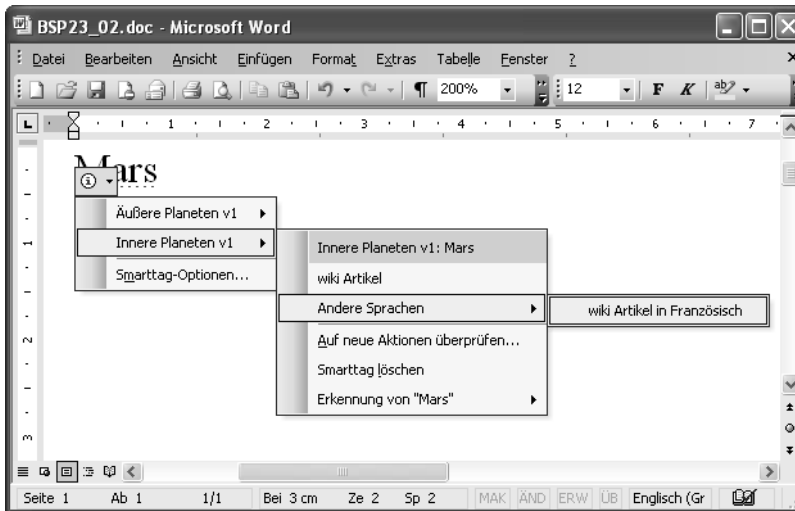
Wenn dieser Smarttag aktiv ist, werden Planetennamen wie »Mars« als Smarttag erkannt und markiert. Das Aktionen-Menü wird dem in Abbildung 23.5 ähnlich sein. Das Beispiel umfasst zwei »Recognizers«, einen für »äußere Planeten« sowie einen für »innere Planeten«. (In der Ausgangslage steht der Ausdruck »Mars« absichtlich »irrtümlich« in beiden Listen).

### Untermenüs

Für jeden »Recognizer« steht ein Menü in der obersten Ebene. Die Einträge für die Aktionen, die in Abbildung 23.3 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.

Abbildg. 23.5 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 23.2. 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 23.2 MOSTL mit zwei »Recognizers« und automatischer Aktualisierung (*BSP23\_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.KAP23Bsp.com/planet/Bsp23_02_mi.htm</FL:moreInfoURL>
  <FL:downloadURL>http://www.KAP23Bsp.com/planet/Bsp23_02_d1.htm</FL:downloadURL>
  <FL:updateable>true</FL:updateable>
  <FL:autoUpdate>true</FL:autoUpdate>
  <FL:lastCheckpoint>1</FL:lastCheckpoint>
  <FL:updateURL>http://www.KAP23Bsp.com/planet/Bsp23_02_up_v2.xml</FL:updateURL>
  <FL:updateFrequency>1</FL:updateFrequency>
```



Listing 23.2 MOSTL mit zwei »Recognizers« und automatischer Aktualisierung (*BSP23\_02.xml*) (Fortsetzung)

```

<FL:smartTag type="urn:schemas-Bsp:astronomisch#innereplaneten">
  <FL:propertyPage>http://www.KAP23Bsp.com/planet/Bsp23_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: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.KAP23Bsp.com/planet/Bsp23_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 Sprachenerkennung 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*

Unter Windows XP

*C:\Dokumente und Einstellungen\<Benutzername>\Anwendungsdaten\Microsoft\Smart Tag Lists*

C:\Dokumente und Einstellungen\<Benutzername>\Anwendungsdaten\Microsoft\Smart Tag Lists\1031

Unter Windows Vista:

C:\Users\<Benutzername>\AppData\Roaming\Microsoft\Smart Tag Lists


C:\Users\<Benutzername>\AppData\Roaming\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 LCIDs 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 kommasetrennte 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 (*Bsp23\_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\Kap23.

Darauf wird momentan in der Hauptliste mit dem Fantasie-URL – <http://www.KAP23Bsp.com/planet/> – hingewiesen. Um der Diskussion zu folgen, sollten die folgenden Dateien auf einem Webserver gespeichert werden:

*Bsp23\_02\_mi.htm*

*Bsp23\_02\_dl.htm*

*Bsp23\_02\_ip.htm*

*Bsp23\_02\_äp.htm*

*Bsp23\_02\_up\_v2.xml*

Bsp23\_02\_up\_v3.xml

Bsp23\_02\_v2.xml

Bsp23\_02\_v3.xml

Die URLs in diesen Dateien sind entsprechend anzupassen und wieder mit der Codierung UTF-8 zu speichern.

Bsp23\_02.xml (die Hauptliste)

Bsp23\_02\_v2.xml

Bsp23\_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. 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.

Für Windows Vista, starten Sie *Programme* in der Systemsteuerung. In diesem Dialogfenster in der Gruppe *Programme und Funktionen* klicken auf den Eintrag *Windows-Funktionen ein- oder ausschalten*. In dem daraufhin angezeigten Dialogfeld *Windows-Funktionen* öffnen Sie den Eintrag *Internetinformationsdienst* über das +-Zeichen, sodass die Untereinträge angezeigt werden. Der IIS (Internet Information Service) verbirgt sich im Untermenü *WWW-Dienste*. Durch Anklicken des Kontrollkästchens vor dem Eintrag *Internetinformationsdienste* werden automatisch die notwendigen Standard-Komponenten in den Menüpunkten *Webverwaltungstools* und *WWW-Dienste* markiert, die für den Betrieb des IIS notwendig sind.

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 *Kap23ST*) und kopieren Sie die Dateien dort hinein. In diesem Fall lautet der URL: *http://localhost/Kap23ST/*.

## Die automatische Aktualisierung einer MOSTL

Damit die Smarttag-Funktionalität prüft, ob die Liste zu aktualisieren ist, werden der XML-Datei die Elemente aus Tabelle 23.1 hinzugefügt.

Tabelle 23.1 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.

**Tabelle 23.1** MOSTL-Elemente für die automatische Aktualisierung der Liste (Fortsetzung)

Elemente für die Aktualisierung	Beschreibung
<updateURL>	Enthält den URL zur Spezifikation für die Aktualisierung (im Beispiel handelt es sich um die Datei <i>Bsp23_02_up_v2.xml</i> , in Listing 23.3 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>Bsp23_02_v2.xml</i> ), wird keine Aktualisierung durchgeführt.

**Listing 23.3** 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>Bsp23_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
```

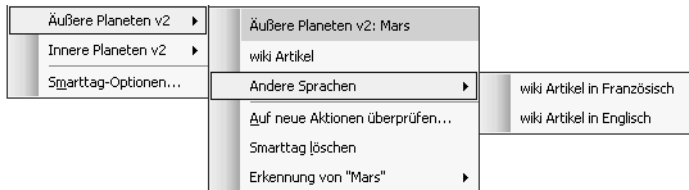
Bei erfolgter Aktualisierung ersetzt der Inhalt der neuen MOSTL den der gegenwärtigen. Der Dateiname bleibt jedoch gleich (*Bsp23\_02.xml*).

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 *Bsp23\_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 *Bsp23\_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 23.6 ersichtlich. Die Beschriftungen führen »v2« auf, und ein englischer Artikel wird neu angeboten.

**Abbildg. 23.6** 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

Das Listing 23.2 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 Beispielpcode 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 die *Suchen und Ersetzen*-Funktionalität. Smarttags unterstützen Perl-Dialekt, leicht modifiziert für den Gebrauch mit XML (die Zeichen »<« und »>« werden mit den Zeichenfolgen &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.

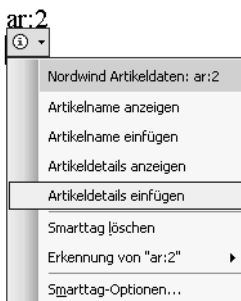
## Ein Smarttag mit VB6 entwickeln

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.

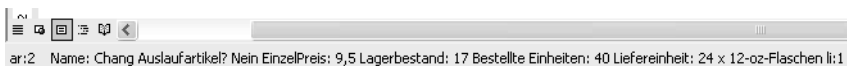
Wie die Abbildung 23.7 veranschaulicht, wird ein Kürzel eingegeben und vom Smarttag erkannt. Das Aktionen-Menü bietet mehrere Optionen an.

Abbildg. 23.7 Das Smarttag erkennt das Kürzel und zeigt entsprechende Optionen an

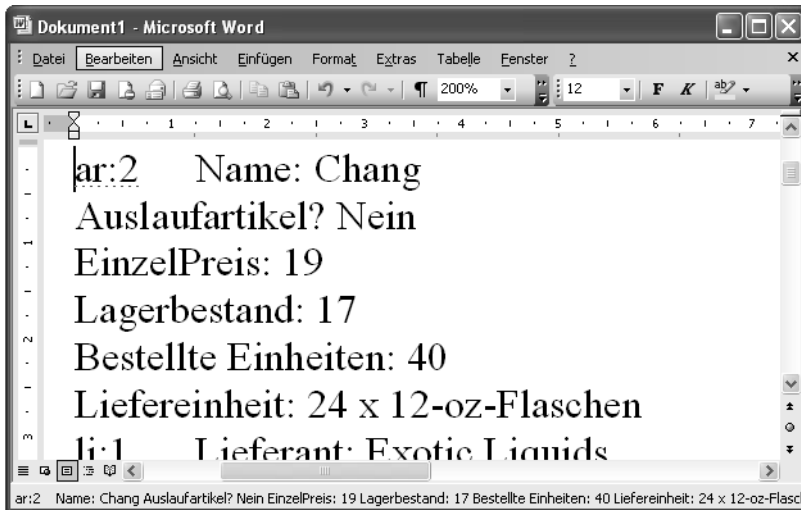


Informationen zum Artikelnamen oder die Details können in der Statusleiste angezeigt (Abbildung 23.8) oder ins Dokument eingefügt (Abbildung 23.9) werden. Der Anwender kann nicht benötigte Informationen löschen oder nach Ausführen des Befehls *Rückgängig machen* die Aktion erneut starten, wenn er einen anderen Artikel will.

Abbildg. 23.8 Artikeldetails werden in der Statusleiste angezeigt



Abbildg. 23.9 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 23.4 wiedergegeben.



Die Dateien *Bsp23\_03\_Actions.cls*, *Bsp23\_03\_DBAccess.bas*, *Bsp23\_03\_Recognizer.cls* sowie *Bsp23\_03\_SharedCode.bas* finden Sie auf der CD-ROM im Ordner *\Beispiele\Kap23\Bsp23\_03\_VB6.zip*, zusammen mit den restlichen VB6-Projektdateien. Es handelt sich um reine Textdateien, die im Windows-Editor sowie in VB6 geöffnet werden können.

**Listing 23.4** »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
```

**Listing 23.4** »Recognizer«-Code dieses VB6-Beispiels (Fortsetzung)

```

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
' 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 _

```

Listing 23.4 »Recognizer«-Code dieses VB6-Beispiels (Fortsetzung)

```

        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 & vbCr & 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 23.2 ein.

Tabelle 23.2 *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>Bsp23_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:Bsp2303#Artikel</b> sowie <b>urn:Bsp2303#Kunden</b> festgelegt.

Die einzige implementierte Methode ist *Recognize*, mit den Parametern in Tabelle 23.3.

**Tabelle 23.3** Parameter der *ISmartTagRecognize.Recognizer*-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 Zei-

chenketten – *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 23.4 ein.

Tabelle 23.4 *ISmartTagAction*-Eigenschaften

Eigenschaft	Beschreibung
ProgID	Legt die Bezeichnung für die Windows-Registrierung fest, in diesem Fall <b>Bsp23_03.clsActions</b>
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 <b>urn:Bsp2303#Artikel</b> sowie <b>urn:Bsp2303#Kunden</b> festgelegt
SmartTagCaption	Die oberste Beschriftung des Aktionen-Menüs sowie der erste Teil der Beschreibung im Dialogfeld <i>Smarttags</i> in den <i>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 sprachunenabhängige Verbnamen für eine bestimmte Verb-ID

Das Action-Interface implementiert eine einzige Methode – **InvokeVerb**. Sie hat die in Tabelle 23.5 aufgelisteten Parameter.

Tabelle 23.5 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 zurückzuschreiben 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)

**Tabelle 23.5** Die Parameter der Methode *ISmartTagAction.InvokeVerb* (Fortsetzung)

Parameter	Beschreibung
XML	Der XML-Text, der von der aufrufenden Anwendung übergeben wurde (nur lesbar)

## Eine Smarttag-DLL erstellen und installieren

Die Datei *Bsp23\_03.dll* sowie die Ressourcendatei *Bsp23\_03.udl* befinden sich auf der CD-ROM im Ordner *\Beispiele\Kap23\Bsp23\_03\_VB6.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:\Word-Buch\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\Kap23\Bsp23_03.dll
```

3. Die folgenden Schlüssel werden in der Windows-Registrierung erstellt:

```
HKEY_CLASSES_ROOT\Bsp23_03.clsActions
HKEY_CLASSES_ROOT\Bsp23_03.clsRecognizer
```

4. Zeigen Sie den Inhalt des Unterschlüssels *CLSID* für *Bsp23\_03.clsActions* an, klicken Sie rechts auf den *Standard*-Eintrag, wählen Sie *Ändern* und kopieren Sie den Wert (er soll aussehen wie *{B0FED99F-4B75-46CF-8B70-3FB50ED0D598}*). Anschließend betätigen Sie *Abbrechen*, um das Dialogfeld auszublenden.
5. Suchen Sie den Schlüssel


```
HKEY_CURRENT_USER\Software\Microsoft\Office\Common\Smart Tag\Actions
```

6. 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.
7. Wiederholen Sie die Schritte 4 bis 6 für den »Recognizer«, um einen neuen Schlüssel mit dem Wert unter *HKEY\_CLASSES\_ROOT\Bsp23\_03.clsRecognizer* zu erstellen:

```
HKEY_CURRENT_USER\Software\Microsoft\Office\Common\Smart Tag\Recognizer
```

### HINWEIS

Sie können alternativ die Datei *Bsp23\_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.

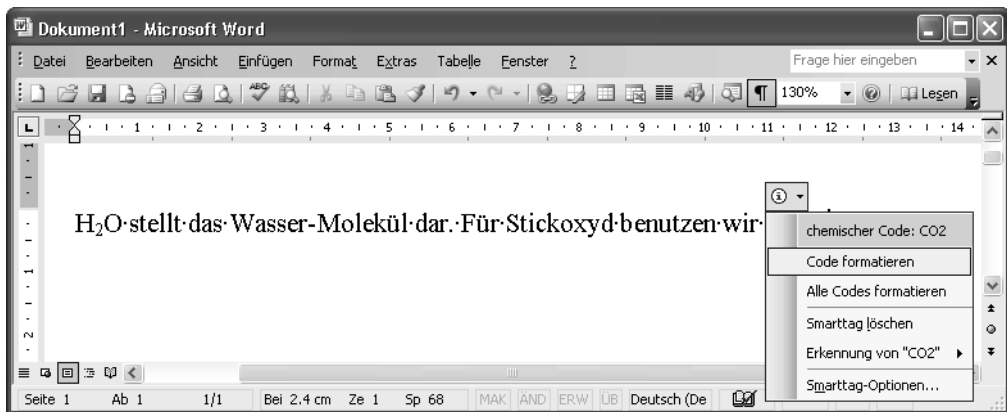
## VSTO-Smarttags

Wie in Kapitel 10 beschrieben, stellt VSTO 2005 verschiedene Office-Schnittstellen (Interfaces) auf vereinfachter Art zur Verfügung. Der Entwickler kann sich somit voll seiner Lösung widmen und muss sich nicht um »die Verkabelung« (die Implementierung) kümmern. So auch für die ISmartTag-Schnittstelle.

Im Gegensatz zu anderen Smarttags sind diejenigen einer VSTO-Lösung dokumentenspezifisch. Die rot gepunkteten Unterstriche erscheinen nur im VSTO-Dokument oder in einem auf Basis einer VSTO-Dokumentvorlage erstellten Dokument.

Word wird auch dazu benutzt, um wissenschaftliche Berichte zu schreiben, sei es in einem Forschungslabor oder als Teil der Schularbeit. Die Formatierung von chemischen Formeln und Molekülen gestaltet sich als zeitraubend, da Ziffern tiefgesetzt werden müssen. Das vorliegende Beispiel zeigt eine Möglichkeit, diesen Vorgang zu beschleunigen. Erkannt werden Zeichenfolgen, die als Wörter im Text stehen, mit einem Großbuchstaben anfangen, gefolgt von weiteren Großbuchstaben oder einer Ziffer. Die Lösung definiert zwei Smarttag-Aktionen, wie in Abbildung 23.10 ersichtlich. Die erste formatiert nur den gegenwärtigen Code, die zweite bearbeitet alle Codes im Dokument. Bei der Ausführung werden die SmartTags aus dem Dokument entfernt.

Abbildg. 23.10 Ein in einer VSTO-Lösung definierter Smarttag



Die wenigen Codezeilen, die hierfür gebraucht werden, entnehmen Sie bitte dem Listing 23.5. Vergleichen Sie diese mit dem Inhalt der Dateien *Bsp23\_03\_Recognizer.cls* und *Bsp23\_03\_Actions.cls* des im Abschnitt »Ein Smarttag mit VB6 entwickeln« beschriebenen Smarttag. Die erforderliche Festlegung aller Eigenschaften der Interface entfällt; VSTO erledigt diese mühsame Arbeit.

Zudem bietet es die Funktionalität in einem dem .NET-Entwickler vertrauten Format an. Der Smarttag wird als eine Objektvariable deklariert und die benötigten Eigenschaften (SmartTagName und SmartTagCaption) als Parameter übergeben:

```
Dim stMolekul As New VSTO.Word.SmartTag(m_stNameSpace, "chemischer Code")
```

Die Aktionen (ChemCodeFormatieren sowie AlleChemCodeFormatieren) werden als Ereignisse und die Erkennungskriterien als eine Eigenschaft des Smarttags behandelt. Letztlich wird das erstellte Objekt der SmartTags-Auflistung des Projekts hinzugefügt.

```
WithEvents ChemCodeFormatieren As VSTO.Word.Action
WithEvents AlleChemCodeFormatieren As VSTO.Word.Action
Private Sub SmartTagsErstellen()
    Dim stMolekul As New VSTO.Word.SmartTag(m_stNameSpace, "chemischer Code")
    stMolekul.Expressions.Add(New Regex("\b[A-Z] [A-Z0-9]*\b"))
    ChemCodeFormatieren = New VSTO.Word.Action("Code formatieren")
    AlleChemCodeFormatieren = New VSTO.Word.Action("Alle Codes formatieren")
    stMolekul.Actions = New VSTO.Word.Action()
    {ChemCodeFormatieren, AlleChemCodeFormatieren}
    Me.VstoSmartTags.Add(stMolekul)
End Sub
```

In den Ereignis-Prozeduren stellt VSTO Eigenschaften des Smarttag-Objekts als Argumente (ActionEventArgs) zur Verfügung. Somit können sein Bereich und der Text problemlos angesprochen werden.

#### Listing 23.5 Code für ein VSTO-Smarttag

```
Imports System.Text.RegularExpressions
Imports VSTO = Microsoft.Office.Tools
Imports Word = Microsoft.Office.Interop.Word

Public Class ThisDocument
    Private Sub ThisDocument_Startup(ByVal sender As Object, ByVal e As System.EventArgs) _
        Handles Me.Startup
        SmartTagsErstellen()
        Me.AttachedTemplate.Saved = True
    End Sub

    Const m_stNameSpace As String = "www.Kap23.com/Beispiele#ChemCode"
    WithEvents ChemCodeFormatieren As VSTO.Word.Action
    WithEvents AlleChemCodeFormatieren As VSTO.Word.Action

    Private Sub SmartTagsErstellen()
        Dim stMolekul As New VSTO.Word.SmartTag(m_stNameSpace, "chemischer Code")
        stMolekul.Expressions.Add(New Regex("\b[A-Z] [A-Z0-9]*\b"))
        ChemCodeFormatieren = New VSTO.Word.Action("Code formatieren")
        AlleChemCodeFormatieren = New VSTO.Word.Action("Alle Codes formatieren")
        stMolekul.Actions = New VSTO.Word.Action()
        {ChemCodeFormatieren, AlleChemCodeFormatieren}
        Me.VstoSmartTags.Add(stMolekul)
    End Sub

    Private Sub ChemCodeFormatieren_Click(
        ByVal sneder As Object, ByVal e As VSTO.Word.ActionEventArgs) _
        Handles ChemCodeFormatieren.Click
        ChemCodeBearbeiten(e.Range.SmartTags(1))
    End Sub
```



Listing 23.5 Code für ein VSTO-Smarttag (Fortsetzung)

```

Private Sub AlleChemCodeFormatieren_Click(
    ByVal sender As Object, ByVal e As VSTO.Word.ActionEventArgs) _
    Handles AlleChemCodeFormatieren.Click
    Dim doc As Word.Document = e.Range.Document
    Dim st As Word.SmartTag
    For Each st In doc.SmartTags
        ChemCodeBearbeiten(st)
    Next
    'Der "aufrufende" Smarttag wird durch die Schleife nicht bearbeitet.
    ChemCodeBearbeiten(e.Range.SmartTags(1))
End Sub

Private Sub ChemCodeBearbeiten(ByVal st As Word.SmartTag)
    Dim zeichen As Word.Range
    Try
        For Each zeichen In st.Range.Characters
            If IsNumeric(zeichen.Text) Then
                zeichen.Font.Subscript = -1 'Wahr
                st.Delete()
            End If
        Next
    Catch ex As Exception
        MessageBox.Show(ex.Message)
    End Try
End Sub

Private Sub ThisDocument_Shutdown(
    ByVal sender As Object, ByVal e As System.EventArgs) Handles Me.Shutdown
End Sub
End Class

```



Die VSTO-Smarttag-Lösung befindet sich in der Datei *VSTO\_ST\_VB.zip* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap23*.

## 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 23.6 aufgelistet) sind nur relevant, wenn mit Smarttags im Aufgabenbereich *Dokumentaktionen* einer Smart Document-Lösung gearbeitet wird; diese werden im folgenden Kapitel 24 vorgestellt.

Tabelle 23.6 Die *SmartTag*-Objekte des Word 2003-Objektmodells

Name	Beschreibungen und Bemerkungen
<i>SmartTagType</i>	Stellt einen bestimmten Smarttag-Namensraum-URI dar. Eigenschaft des Word <i>Application</i> -Objekts.

**Tabelle 23.6** 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 23.7 und Tabelle 23.8 aufgelistet, gruppiert und kommentiert.

**Tabelle 23.7** 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 23.8** Smarttag-Namen, Beschriftungen und nicht numerische ID-Werte (Notizen zu Tabelle 23.7)

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-ID</b> -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 23.9 aufgelistet.

**Tabelle 23.9** 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 ins HTML- oder XML-Format beizubehalten sind. 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 23.9 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.DisplaySmartTagOptions	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-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 23.11.

Abbildg. 23.11 Ein transformiertes XML-Dokument mit Informationen über alle sich im Quelldokument befindenden Smarttags

Smarttags verwendet in: \BSP23\_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:/KAP23/BSP23_02_download.htm
Eigenschaften:	
Name: Value	
typ: gestirne	
monde: 2x	
SmartTag.Name: → urn:schemas-beispiel:astronomisch#innereplaneten	SmartTag.FriendlyName: → innereplaneten
SmartTag.Text: marsx	Download-URL: file:///c:/KAP23/BSP23_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 *Bsp23\_04.doc* mit dem VBA-Code sowie die Transformationsdatei *Bsp23\_04.xsl* finden Sie auf der CD-ROM im Ordner *\Beispiele\Kap23*. Die Konstantwerte für die Pfadangaben in der Prozedur *ListSmartTagsByType* müssen vor der ersten Ausführung angepasst werden.

Der Beispielcode in Listing 23.6, Listing 23.7 sowie Listing 23.8 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 2003/2007 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 22 vorgestellten »Microsoft Office 2003 Word-ProcessingML 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 23.6 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 23.7 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 23.8 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 *Bsp23\_04.doc* enthält zudem eine alternative Version dieses Codes – *ListSmartTagByTypeDebug* – die die Smarttag-Informationen in das Direktfenster statt in eine XML-Datei ausgibt.

**Listing 23.6** 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.KAP23.com/BSP04"

' Passen Sie die folgenden Konstantwerte wo nötig an
Const strXMLPathName = "C:\WordBuch\Beispiele\KAP23\Bsp23_04.xml"
```

Listing 23.6 Beispielcode, um alle Smarttag-Information eines Dokuments aufzulisten (Fortsetzung)

```

Const strXSLTPathName = "c:\WordBuch\Beispiele\KAP23\Bsp23_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
            End If
        End With
    Next

```

**Listing 23.6** Beispielcode, um alle Smarttag-Information eines Dokuments aufzulisten (Fortsetzung)

```

        Call ListSmartTagActions(objSmartTagActions, objXMLDocument, _
                                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 23.7** 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

```



Listing 23.7 Untergeordnete Prozedur, die die Aktionen eines Smarttags zusammenträgt (Fortsetzung)

```

    Err.Clear
    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 & vbCr & Err.Description

End Sub

```

Listing 23.8 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 24).

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 910 ff.)
- Konzeptgrundlagen (Seite 912 ff.)
- Die Entwicklung von MOSTL-Smarttags (Seite 915 ff.)
- Die Entwicklung einer Smarttag-DLL in VB6 (Seite 923 ff.)
- Die Erstellung eines Smarttags in einer VSTO-Dokumentlösung (Seite 931 ff.)
- Die Smarttag-Funktionalität im Word-Objektmodell (Seite 933 ff.)

## Kapitel 24

# Smart Documents

### **In diesem Kapitel:**

Was ist ein Smart Document?	944
Ein MOSTL-Smart Document	946
Eine Smart Document-DLL	970
Das Erweiterungspaket-Manifest digital signieren	997
Zusammenfassung	998

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. Sie funktioniert teilweise auch unter Office 2007, allerdings mit erhöhter Sicherheit. In diesem Buch stellen wir jedoch nur die Smart Document-Technologie für Word vor.

#### HINWEIS

Smart Documents sind für Office 2003 die einzige Möglichkeit, eigene Aufgabenbereiche mit COM-Anwendungen zu erstellen. Visual Studio for Office 2005 (VSTO 2005) besitzt zwar eine vereinfachte Funktionalität zur Erstellung von Aufgabenbereichen, steht jedoch nur .NET-Entwicklern zur Verfügung. Mehr dazu lesen Sie in Kapitel 10.

Anhand des großen Echos gibt es für Nicht-VBA-Entwickler (also auch für VB6 sowie VB.NET) in Office 2007 das *CustomTaskPane*-Interface. Dieses wurden ebenfalls in Kapitel 10 im Abschnitt zu VSTO-COM-Add-Ins kurz vorgestellt.

Mehr Informationen zu den CustomTaskPane-Schnittstellen finden sie auf

<http://www.microsoft.com/germany/msdn/library/office/ErstellenVonBenutzerdefiniertenAufgabenbereichenInOffice2007.aspx?mfr=true> sowie

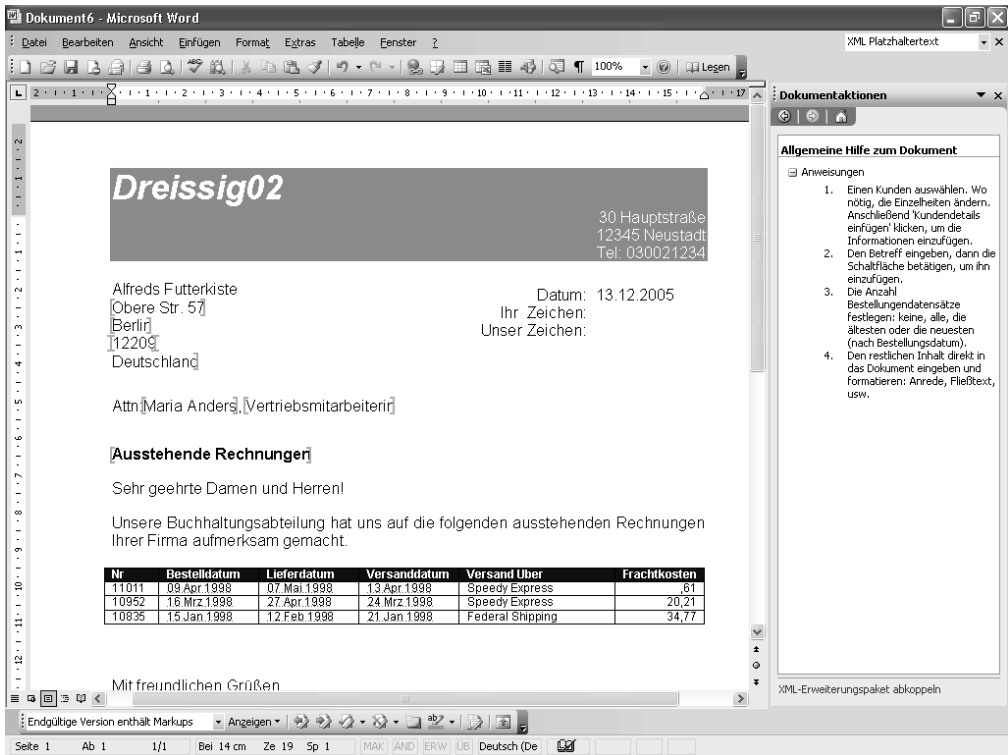
<http://blogs.msdn.com/andrew/archive/2006/12/23/low-level-support-for-icustomtaskpaneconsumer-iribbonextensibility-formregionstartup-etc.aspx>.

## Was ist ein Smart Document?

In Kapitel 23 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 22) erkannt, an denen der Anwender gegenwärtig arbeitet bzw. in denen sich die Einfügemarke gerade befindet.

**Abbildg. 24.1** 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 23 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\SmartDocumentSDK` 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 24.2). 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 22 beschrieben, werden XML-Elemente in Word-Tabellen beim Einfügen neuer Zeilen automatisch dupliziert, so dass der Anwender problemlos Datensätze hinzufügen kann.

Abbildg. 24.2 In eine Word-Tabelle exportierte XML-Daten aus einer Access-Datenbank

The screenshot shows a Microsoft Word window titled 'BSP24\_01B.xml - Microsoft Word'. The main content is a table with the following data:

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	Martín 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.

On the right side, the 'Dokumentaktionen' task pane is visible. It contains sections for 'Nordwind Kunden dataroot-Element', 'Element Kunde', and 'Element Position', each with instructions on how to use the data in the document.

## 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 24.1 benötigten Beispieldateien befinden sich auf der CD-ROM zum Buch im Ordner `\Beispiele\Kap24`. Die Beispiel-Datenbank *Nordwind.mdb* finden Sie im Ordner `\Beispiele\Datenbank`.

Tabelle 24.1

Beispieldateien für den Abschnitt über MOSTL-Smart Documents

Beispieldatei	Beschreibung
<i>Bsp24_01A.xml</i>	Ein Word-Dokument mit einem <b>Kunden</b> -Datensatz in einem benutzerdefinierten XML-Vokabular
<i>Bsp24_01B.xml</i>	Eine erweiterte Version von <i>Bsp24_01A.xml</i> , die mehr Datensätze enthält und mit einem Erweiterungspaket verbunden ist

**Tabelle 24.1** Beispieldateien für den Abschnitt über MOSTL-Smart Documents (Fortsetzung)

Beispieldatei	Beschreibung
<i>Bsp24_01.xsd</i>	Das Schema für <i>Bsp24_01A.xml</i> und <i>Bsp24_01B.xml</i>
<i>Bsp24_01.xsl</i>	Eine Transformation für <i>Bsp24_01A.xml</i> und <i>Bsp24_01B.xml</i>
<i>Bsp24_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>Bsp24_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>Bsp24_01S.xml</i>
<i>Bsp24_01N.xml</i>	Enthält Informationen zur MOSTL-Datei, zum Schema sowie zur Transformation
<i>Bsp24_01.reg</i>	Schaltet die Sicherheit für Erweiterungspakete aus

Gehen Sie zur Installation folgendermaßen vor:

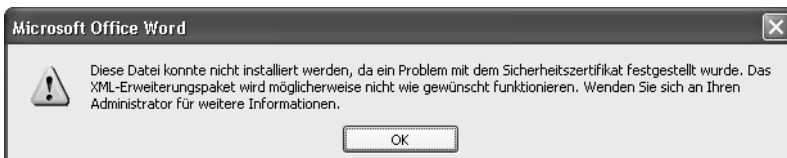
1. Kopieren Sie alle genannten Beispieldateien – mit Ausnahme der *Nordwind.mdb* – in einen Ordner auf Ihrem Rechner. Im Beispiel wird der Ordner *C:\Beispiele\Kap24* verwendet, den Sie auch anlegen sollten. Die Datenbank *Nordwind.mdb* kopieren Sie bitte in den Ordner *C:\Beispiele\Datenbank*.
2. Starten 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*.



In Word 2007 finden Sie auf der Registerkarte *Entwicklertools* der Multifunktionsleiste einzelne Schaltflächen für alle in diesem Kapitel erwähnten Registerkarten des Dialogfelds *Dokumentvorlagen und Add-ins*.

3. Klicken Sie auf die Schaltfläche *Schema hinzufügen* und wählen Sie aus dem Ordner *C:\Beispiele\Kap24* (bzw. dem von Ihnen verwendeten Ordner) die Datei *Bsp24\_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 *Bsp24\_01M.xml*.

**ACHTUNG** Word wird vermutlich die in Abbildung 24.3 angezeigte Meldung einblenden, dass die Datei wegen Problemen mit der Sicherheitsprüfung nicht geöffnet werden konnte. Bestätigen Sie den Warnhinweis über *OK*.

**Abbildg. 24.3** 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 *Bsp24\_01.reg* aus. Bestätigen Sie die Frage, ob die Informationen in der Datei *Bsp24\_01.reg* der Registrierung hinzugefügt werden sollen, über die Schaltfläche *Ja*.

Mit dieser Datei wird folgender Eintrag der Registrierung hinzugefügt bzw. geändert:

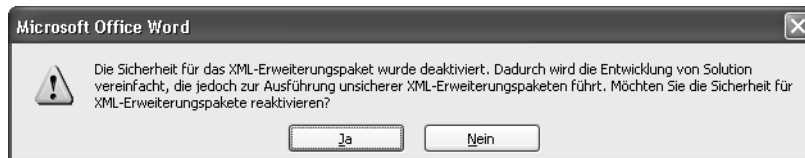
[HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Office\Common\Smart Tag]

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 *Bsp24\_01M.xml* weist eine Warnmeldung (Abbildung 24.4) darauf hin, dass die Sicherheit für XML-Erweiterungspakete ausgeschaltet ist, und fragt, ob die Sicherheit wieder eingeschaltet werden soll.

Abbildg. 24.4

Standardmäßig ist die Schaltfläche *Ja* aktiviert, was die Festlegung des Registry-Schlüssels rückgängig machen würde

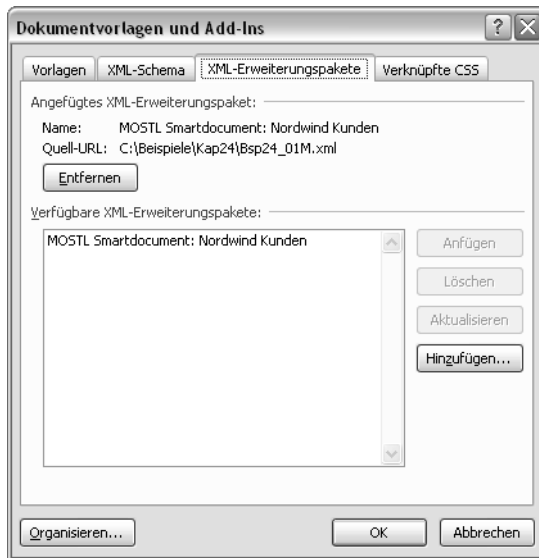


6. Lehnen Sie die Reaktivierung der Sicherheit unbedingt über die Schaltfläche *Nein* ab.
7. Um das Erweiterungspaket anschließend dem aktuellen Dokument anzufügen, klicken Sie auf die Schaltfläche *Anfügen*. Dies ist nur möglich, weil das Schema mit dem gleichen Namensraum in der Registerkarte *XML-Schema* aktiviert ist. Nachdem Sie das Erweiterungspaket der Datei angefügt haben, sollte das Dialogfeld wie in Abbildung 24.5 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.

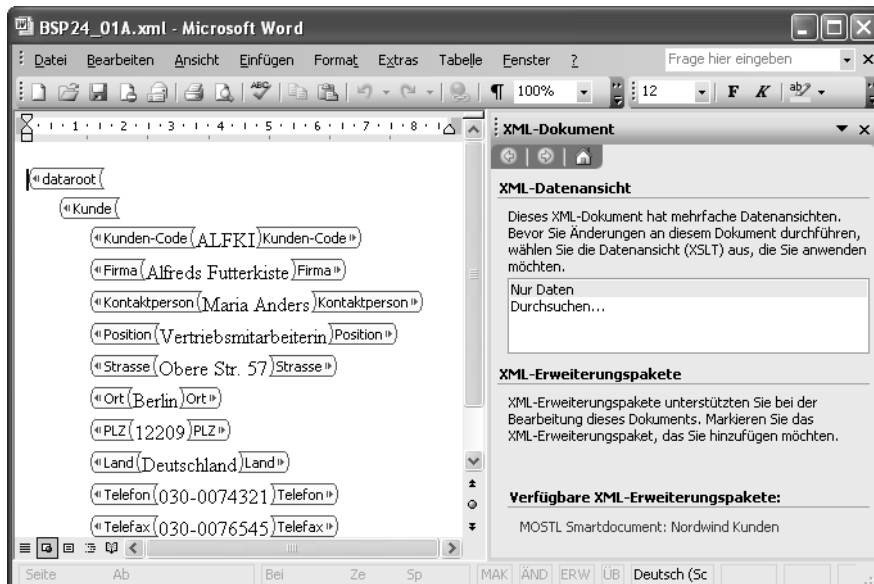
Abbildg. 24.5 Das XML-Erweiterungspaket wurde dem aktuellen Dokument angefügt



8. Schließen Sie das Dialogfeld *Dokumentvorlagen und Add-Ins* mit *OK*.

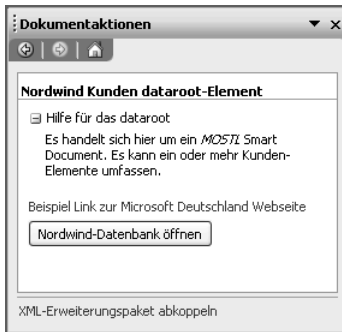
Wird nun eine XML-Datei (*Bsp24\_01A.xml*) geöffnet, die durch ihren URI auf das Schema (aber nicht das Erweiterungspaket) verweist, erscheint sie in Word wie in Abbildung 24.6 ersichtlich. Ein Link zum Erweiterungspaket befindet sich im unteren Teil des Aufgabenbereichs *XML-Dokument*. Wird er angeklickt, erscheint der Aufgabenbereich *Dokumentaktionen*.

Abbildg. 24.6 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 24.7. Die Angaben in diesem Dokumentaktionen-Abschnitt gelten für das ganze XML-Dokument und bleiben sichtbar, egal, wo man sich im XML befindet.

Abbildg. 24.7 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 24.8 aus. Unter dem Abschnitt für das Wurzelement findet sich einer für dieses Element. Er enthält lediglich Informationen zum erwarteten Elementinhalt.

Abbildg. 24.8 Der Aufgabenbereich *Dokumentaktionen* mit zwei eingeblendeten Hierarchie-Ebenen



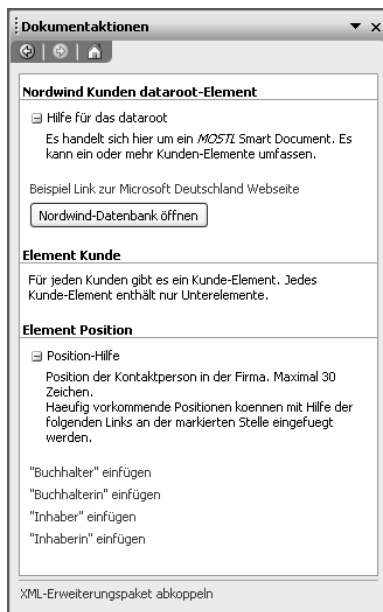
Wenn Sie der Reihe nach die weiteren, sich in der nächsten Ebene befindenden Elemente anwählen, wird im Aufgabenbereich ein dritter Abschnitt eingeblendet. Dessen Inhalt variiert je nach Element; meistens handelt es sich um Hilfe-Texte oder Beschriftungen. Die Abbildung 24.9 zeigt die Angaben für das 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.

#### HINWEIS

Die Abbildung 24.9 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. 24.9 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 24.2 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 22 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 22: Die benutzerdefinierten Tags müssen sich außerhalb der WordProcessingML-Elemente für die entsprechenden Tabellentile 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).

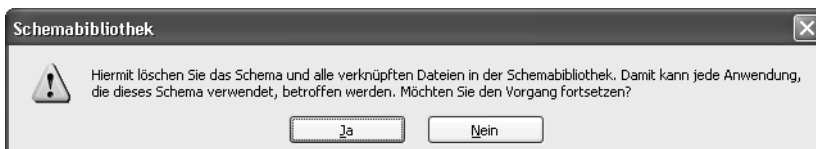
Beim automatischen Erstellen einer neuen Tabellenzeile durch Drücken der -Taste am Tabellenende wird ein zusätzlicher Satz des Elements Kunde erzeugt, mit allen Unterelementen, wie in der vorherigen Zeile definiert.

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 Arbeitserleichterung 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 24.10 bestätigen Sie mit *Ja*. Damit werden auch die mit dem Schema verbundenen Transformationen und Erweiterungspakete gelöscht.

Abbildg. 24.10 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 *Bsp24\_01B.xml*. Word blendet das Dialogfeld aus Abbildung 24.11 ein.

**Abbildg. 24.11** Erweiterungspaket, um alle verbundenen Schemas und Transformationen in einem Schritt zu installieren



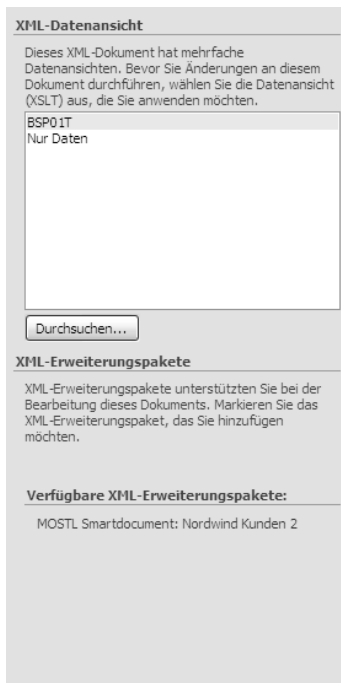
5. Aktivieren Sie das Kontrollkästchen *Als Standardauswahl für dieses Dokument festlegen* und bestätigen Sie die Frage mit *Ja*. Die Sicherheitsmeldung in Abbildung 24.4 müsste erscheinen; beantworten Sie diese mit *Nein*. Es werden kurz verschiedene Dialogfelder eingeblendet, bevor das Dokument schließlich 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 2*. Sie sehen das Dokument, wie es in Abbildung 24.2 vorliegt, mit dem Aufgabenbereich *Dokumentaktionen* in Abbildung 24.7.

**Abbildg. 24.12** In Word 2007 wird eine der Lösung hinzugefügte Transformation automatisch erkannt



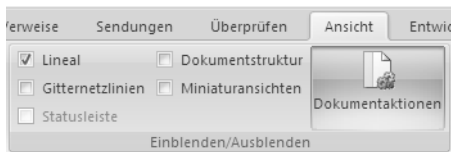


In Word 2007 wurde der Ablauf geändert. Beim Öffnen des Dokuments wird die Transformation automatisch geladen; der in Abbildung 24.12 abgebildete Aufgabenbereich *Datenansicht* wird eingeblendet. Im untersten Abschnitt kann der Benutzer auf den Link zum Erweiterungspaket (*MOSTL Smartdocument: Nordwind Kunden 2*) klicken. Anschließend erscheint der Aufgabenbereich *Dokumentaktionen*.

**HINWEIS**

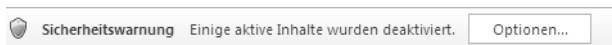
Liegt eine Smart Document-Lösung vor, erscheint auf der Registerkarte *Ansicht* eine Schaltfläche mit der Bezeichnung *Dokumentaktionen* (Abbildung 24.13). Mit dieser kann der Benutzer den gleichnamigen Aufgabenbereich beliebig ein- und ausblenden.

**Abbildg. 24.13** Die für Smart Document-Lösungen dem Benutzer zur Verfügung stehende Schaltfläche

**ACHTUNG**

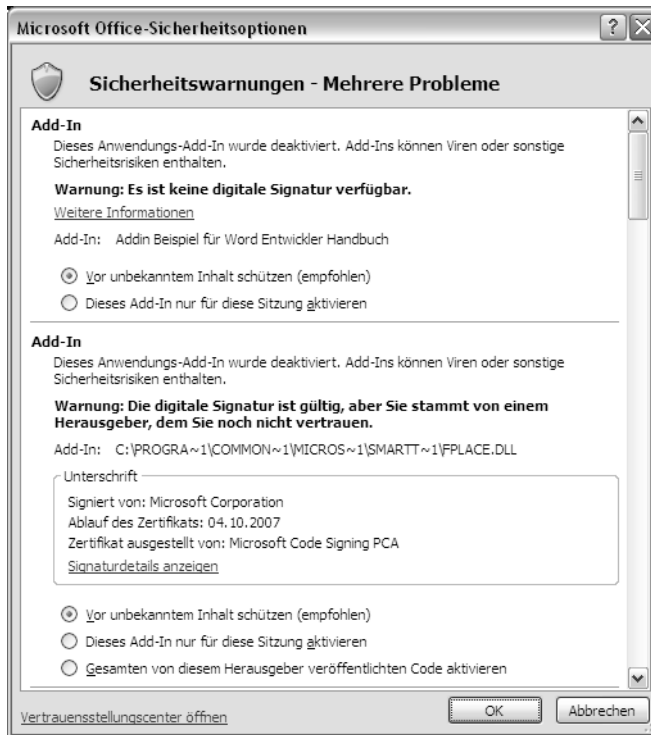
Ist in Word 2007 die Sicherheitseinstellung *Anwendungs-Add-Ins müssen von einem vertrauenswürdigen Herausgeber signiert sein* im Vertrauensstellenscenter in der Kategorie *Add-Ins* aktiviert, so können beim erstmaligen Laden eines Erweiterungspakets Probleme entstehen, obwohl der Teil digital signiert wurde. Falls das Kontrollkästchen *Benachrichtigung für nicht signierte Add-Ins deaktivieren (Code bleibt deaktiviert)* eingeschaltet ist, sollte dieses deaktiviert werden. Beim erneuten Öffnen der XML-Datei wird oberhalb des Dokuments ein Balken mit einer Sicherheitswarnung (siehe Abbildung 24.14) erscheinen. Klicken Sie auf *Optionen*, um mehr Informationen zu erhalten.

**Abbildg. 24.14** Die Sicherheitswarnung erscheint beim Laden von nicht signierten Add-Ins



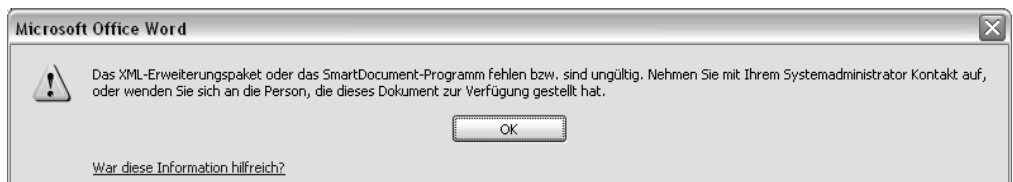
Im Dialogfeld *Microsoft Office-Sicherheitsoptionen* (Abbildung 24.15) werden die Probleme (die nicht signierten Add-Ins) aufgelistet. Unter anderem könnten mehrere für den Aufgabenbereich *Dokumentaktionen* benötigte *.dll*-Dateien aufgelistet sein. Um dieses Problem zu beseitigen, schalten Sie die Option *Gesamten von diesem Herausgeber veröffentlichen Code aktivieren* ein und bestätigen mit *OK*.

Abbildg. 24.15 Nicht signierte COM-Add-Ins werden bei erhöhter Sicherheit nicht geladen




Wird das Dialogfeld in Abbildung 24.16 eingeblendet, muss Word beendet und erneut gestartet werden. Beim nochmaligen Öffnen der XML-Datei wird das Erweiterungspaket anstandslos geladen.

Abbildg. 24.16 Mögliche Warnmeldung beim Laden eines Erweiterungspakets



Wenn Sie in die Tabelle klicken und die Einfügemarke durch die Zelle bewegen, werden Sie die verschiedenen Inhalte des Aufgabenbereichs wiedererkennen.

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 Tabellenzelle 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 Klick mit der rechten Maustaste auf den fehlerhaften 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 Tabellenzeile 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 24.17 wird eingeblendet.

Abbildg. 24.17 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* bzw. *.docx*) 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 24.18 eingeblendet. Das Microsoft Office 2003 Smart Document SDK erklärt, unter welchen Umständen Word ein Erweiterungspaket nicht automatisch anfügt.

**Abbildg. 24.18** Dieses Dialogfeld wird angezeigt, wenn mehr als ein installiertes Erweiterungspaket für ein Dokument benutzt werden kann



## So funktioniert's

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.

Für *Bsp24\_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 24.1 veranschaulicht den Inhalt der Datei.

**Listing 24.1** Inhalt der Beispieldatei *Bsp24\_01A.xml* mit einem *Kunde*-Datensatz

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<dataroot
  xmlns="http://www.KAP24Beispiel.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>
```

Das zweite Beispieldokument *Bsp24\_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.KAP24Beispiel.com/BSP01" manifestPath="BSP24_01N.xml"?>
```

*Bsp24\_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.

*Bsp24\_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 22) 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 24.2 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 24.3 zu entnehmen ist, werden alle Kindelemente identisch verarbeitet. Fehlt in der XML-Datei ein Element dieser Liste, wird dafür keine Tabellenzelle erstellt.

**Listing 24.2** 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 24.3** 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>
  </ns2:Kunden-Code>
</xsl:template>
```

**Listing 24.3** Der vom »Inference Tool« erstellte `<xsl:template>`-Code für alle Kindelemente des *Kunde*-Elements (*Fortsetzung*)

```

</xsl:attribute>
</xsl:for-each>
<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 24.2 und Listing 24.3) befindet sich in Listing 24.4. Im Gegensatz zur vorangehenden Transformation wird die Tabellenzeile zuerst erstellt und danach die Zellen gefüllt, Element für Element (nur der Teil für das Element *Kunden-Code* wird angezeigt; alle weiteren Elemente werden ähnlich behandelt).

**Listing 24.4** 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>

```

**Listing 24.4** Die angepasste Transformation erzeugt, unabhängig von der Anzahl der Kindelemente eines *Kunde*-Elements, eine gleichmäßige Tabelle (*Fortsetzung*)

```

        <w:sz w:val="20" />
        <w:sz-cs w:val="20" />
    </w:rPr>
    <w:t><xsl:apply-templates select="ns2:Kunden-Code" /></w:t>
</w:r>
</w:p>
</w:tc>
</ns2:Kunden-Code>

```

*Bsp24\_01S.xml* (die MOSTL-Solution). Der Code dieser Datei befindet sich in Listing 24.5. Eine Smart Document MOSTL-Solution muss das gleiche Schema respektieren wie eine Smarttag-MOSTL (siehe Kapitel 23). 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 ein 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.KAP24Beispiel.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 den 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 24.2 aufgelistet.

Tabelle 24.2 Aktionen für Smart Document MOSTL-Lösungen

Aktion	Beschreibung/Bemerkung
<i>Separator</i>	Waagrechte 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 24.5).
<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 24.5 Inhalt der MOSTL-Solution-Datei *Bsp24\_01S.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.KAP24Beispiel.com/BSP01#dataroot">
    <caption>Nordwind Kunden dataroot-Element</caption>
    <actions>
      <action id="dataroot_help">
        <actionType>Help</actionType>
        <caption>Hilfe für das dataroot</caption>
      </action>
    </actions>
  </smartDoc>
</smartTagList>
```

Listing 24.5 Inhalt der MOSTL-Solution-Datei *Bsp24\_01S.xml* (Fortsetzung)

```

    <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>

  <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.KAP24Beispiel.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.KAP24Beispiel.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.KAP24Beispiel.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>

```

**Listing 24.5** Inhalt der MOSTL-Solution-Datei *Bsp24\_01S.xml* (Fortsetzung)

```

    </action>
  </actions>
</smartDoc>

<smartDoc type="http://www.KAP24Beispiel.com/BSP01#Kontaktperson">
  <caption> Element Kontaktperson</caption>
  <actions>
    <action id="Kontaktperson_label">
      <actionType>Label</actionType>
      <caption>Name der Hauptkontaktperson der Firma. Maximal 30 Zeichen.</caption>
    </action>
  </actions>
</smartDoc>

<smartDoc type="http://www.KAP24Beispiel.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>

```



Listing 24.5 Inhalt der MOSTL-Solution-Datei *Bsp24\_01S.xml* (Fortsetzung)

```

        <insertXML><w:p><w:r><w:t>InhaberIn</w:t></w:r></w:p></insertXML>
    </elementAction>
</action>
</actions>
</smartDoc>

<smartDoc type="http://www.KAP24Beispiel.com/BSP01#Strasse">
    <caption> Element Strasse</caption>
    <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.KAP24Beispiel.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.KAP24Beispiel.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.KAP24Beispiel.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.KAP24Beispiel.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>

```

**Listing 24.5** Inhalt der MOSTL-Solution-Datei *Bsp24\_01S.xml* (Fortsetzung)

```

</smartDoc>

<smartDoc type="http://www.KAP24Beispiel.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>

<smartDoc type="http://www.KAP24Beispiel.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>

```

*Bsp24\_01M.xml* sowie *Bsp24\_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 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« in diesem Kapitel.

Wir zeigen hier in Listing 24.6 nur den Inhalt des Manifests *Bsp24\_01N.xml* an, da dieses auf dem Grundgerüst von *Bsp24\_01M.xml* aufbaut. Weitere Anmerkungen zu diesem Manifest:

- Das Element `<uri>` bezieht sich auf den URI des Schemas (*Bsp24\_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 »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 (*Bsp24\_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 Ordnernamen ungültige 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\_KAP24Beispiel\_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 24.6

Die Manifest-Datei *Bsp24\_01N.xml* des MOSTL-Smart Document-Beispiels

```
<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.KAP24Beispiel.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>BSP24_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>
```

**Listing 24.6**

 Die Manifest-Datei *Bsp24\_01N.xlm* des MOSTL-Smart Document-Beispiels (Fortsetzung)

```

<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>BSP24_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>
    <SD:type>solutionList</SD:type>
    <SD:version>1.0</SD:version>
    <SD:filePath>BSP24_01S.xml</SD:filePath>
  </SD:file>
</SD:solution>
</SD:manifest>
    
```

### Globally Unique Identifiers (GUIDs)

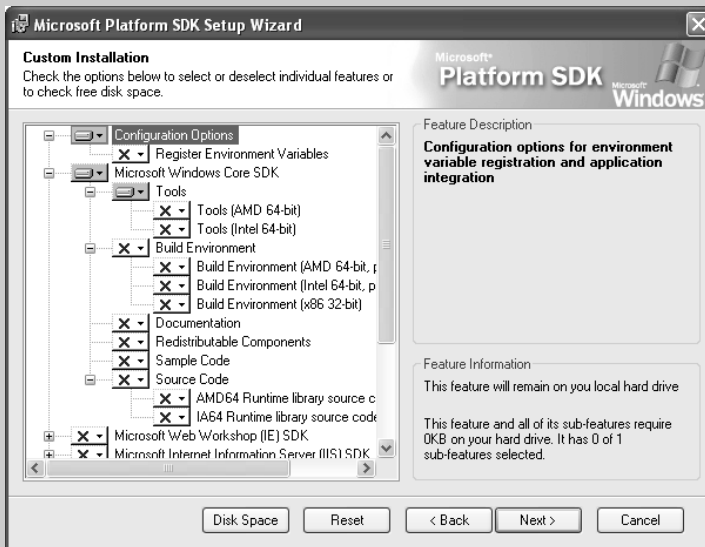
Wie im letzten Abschnitt erwähnt, müssen Smart Document Solutions eine eindeutige SolutionID haben. Meistens wird eine GUID (»Globally Unique Identifier«) verwendet.

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 zu 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 24.19 eingeblendet. Deaktivieren Sie alle Optionen außer den Haupteinträgen »Configuration Tools« und »Microsoft Windows Core SDK/Tools«.
4. Führen Sie die Installation durch. Bitte beachten Sie, dass die Komponenten über das Internet heruntergeladen werden; die Installation kann also dauern. ►

Abbildg. 24.19 Installation des Teils des Platform SDK mit dem GUIDGEN-Werkzeug

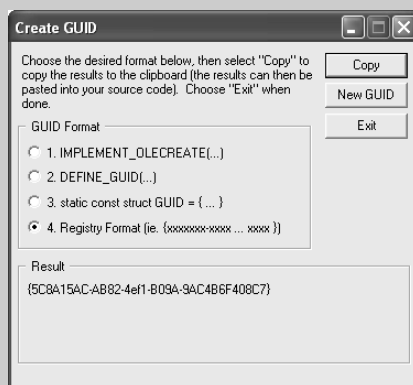


Suchen Sie nun die Datei *GUIDGEN.EXE* auf Ihrem Rechner. Sie müsste sich im festgelegten Speicherort, im Unterordner *Bin* befinden. Führen Sie sie aus; das Dialogfeld aus Abbildung 24.20 sollte erscheinen.

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. 24.20 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

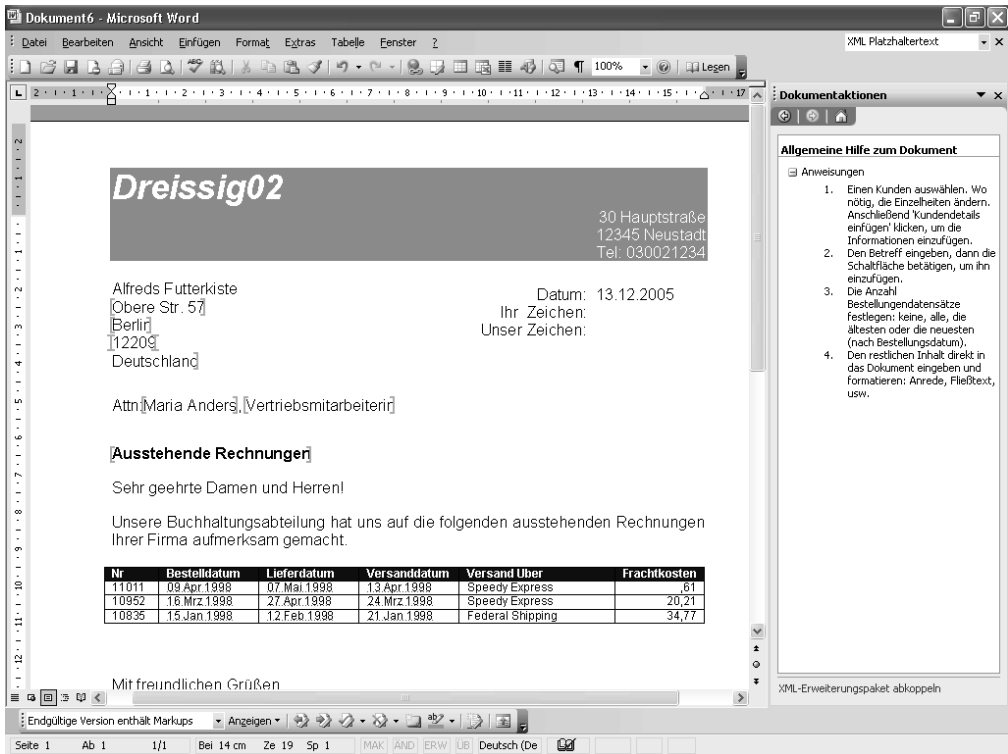
Das Beispiel dieses Abschnitts stellt einige, aber längst nicht alle Möglichkeiten einer DLL-Lösung vor. Es soll vor allem den Aufbau einer Smart Document-DLL verdeutlichen. Die Code-Beispiele sind in der klassischen Visual Basic 6-Sprache gehalten.

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 24.21 dargestellt.

Abbildg. 24.21 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 24.3 benötigten Beispieldateien befinden sich auf der CD-ROM im Ordner `\Beispiele\Kap24`. Die Beispiel-Datenbank *Nordwind.mdb* finden Sie im Ordner `\Beispiele\Datenbank`.

Tabelle 24.3 Beispieldateien für den Abschnitt über eine Smart Document-DLL

Beispieldatei	Beschreibung
<i>Bsp24_02.dot</i>	Eine Word-Vorlage mit Briefkopf und Standardtext sowie Elemente eines benutzerdefinierten XML-Vokabulars
<i>Bsp24_02.xsd</i>	Das mit der Vorlage <i>Bsp24_02.dot</i> verbundene Schema
<i>Bsp24_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>Bsp24_02.udl</i>	Eine Data-Link-Datei, die der DLL die Verbindungsangaben für die Datenbank bereitstellt

**Tabelle 24.3** Beispieldateien für den Abschnitt über eine Smart Document-DLL (Fortsetzung)

Beispieldatei	Beschreibung
<i>Bsp24_02.xsl</i>	Wird vom »action handler« zur Transformation der Bestelldaten bei deren Einfügung benutzt
<i>Bsp24_02M.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 *Bsp24\_02.zip* alle Quelldateien, um das Projekt in Visual Basic 6 zu öffnen und bearbeiten.

1. Kopieren Sie alle Dateien im Ordner *\Beispiele\Kap24* auf der Buch-CD, deren Namen mit den Zeichen »Bsp24\_02« beginnen, in einen lokalen Ordner mit der Pfadangabe *C:\Beispiele\Kap24*.
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 *Bsp24\_02.udl* mit dem Windows-Editor entsprechend anpassen.)
3. Navigieren Sie im Windows-Explorer zum Ordner mit den Beispieldateien und klicken Sie doppelt auf *Bsp24\_02.dot*, um ein neues Dokument zu erzeugen.

Falls das damit verbundene Erweiterungspaket noch nicht installiert ist, erscheint die Meldung in Abbildung 24.22. Aktivieren Sie, wie beim MOSTL-Beispiel, das Kontrollkästchen und bestätigen Sie die Installation mit *Ja*.

**Abbildg. 24.22** Das Erweiterungspaket installieren


Falls die Sicherheit für Erweiterungspakete ausgeschaltet ist, wird auch diese Meldung (Abbildung 24.4) 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 24.22. Ansonsten wird das Dokument wie ein normales Dokument, ohne den Aufgabenbereich *Dokumentaktionen*, geöffnet. (Mehr darüber können Sie im SDK nachlesen.)

Um eine Lösung aus einem anderen Pfad installieren zu lassen, ändern Sie einfach in der Vorlage (*Bsp24\_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



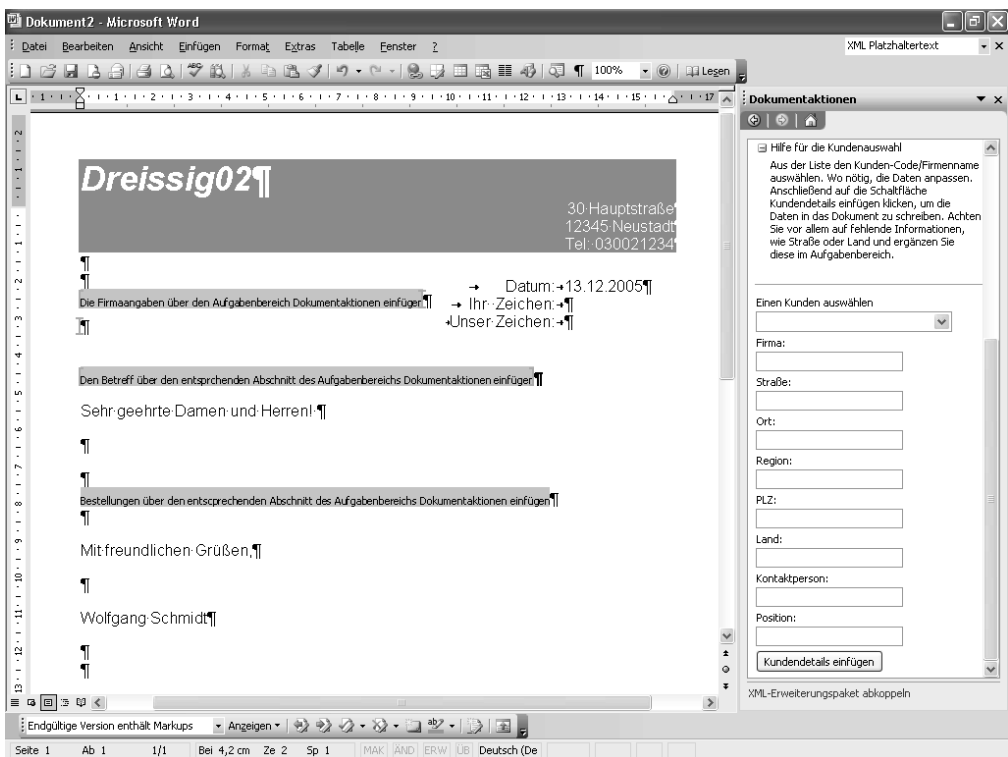
3. Den Schema-Eintrag in der Liste markieren und *Schema löschen* betätigen
4. Die darauf folgende Frage, ob Sie das Schema samt alle verknüpften Dateien löschen möchten, mit *Ja* beantworten

Nun wird beim Erstellen eines neuen Dokuments die Meldung in Abbildung 24.22 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 Dokumenteneigenschaft festgehalten.

Auf dem Bildschirm erscheint das Gerüst des Briefes mit Platzhaltertexten an drei Stellen (Abbildung 24.23). 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.

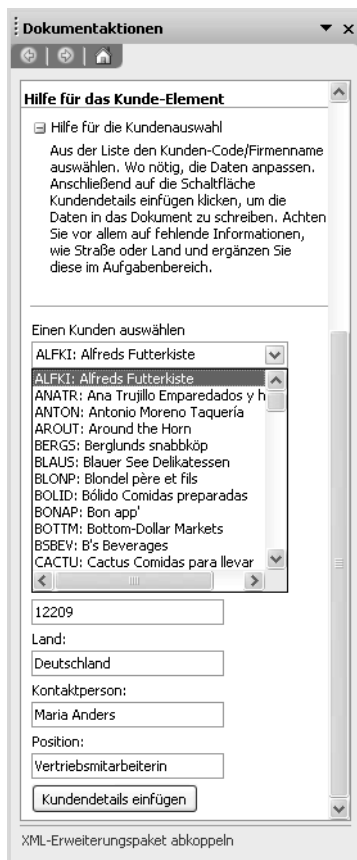
Abbildg. 24.23 Das Gerüst für einen Kundenbrief



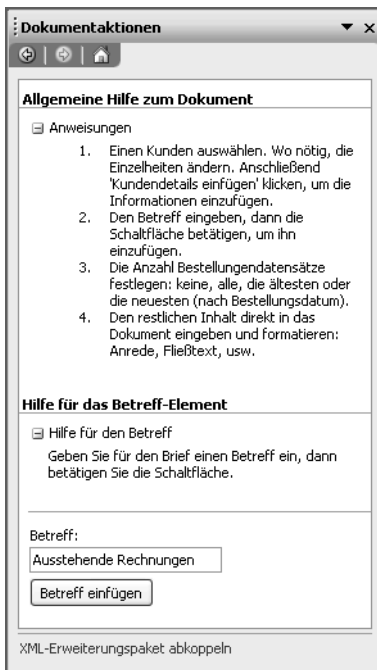
Das Smart Document stellt eine Verbindung zur Datenbank her, um die Dropdownliste in Abbildung 24.24 zu füllen. Nach Auswahl eines Eintrags erscheinen die Informationen in den darunter liegenden Textfeldern, wo sie nach Bedarf bearbeitet werden können. Durch Betätigung der Schaltfläche *Kundendetails einfügen* werden die Daten in die (unsichtbaren) XML-Elemente des Dokuments übertragen. Die Erläuterung dazu finden Sie im Abschnitt »Steuerelementereignisse« in diesem Kapitel.

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 24.24 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« in diesem Kapitel beschrieben.

**Abbildg. 24.24** Der kundenspezifische Teil des Aufgabenbereichs stellt aktuelle Daten aus der Datenbank zur Verfügung



Abbildg. 24.25 Der Aufgabenbereich mit der Hilfe zum ganzen Dokument sowie für das Element *Betreff*

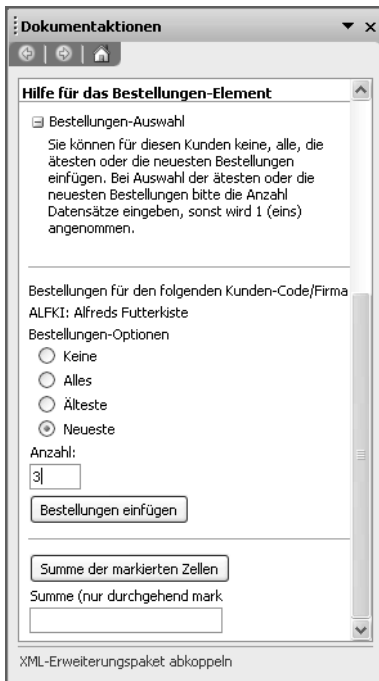


Nach dem Einfügen des Betreffs bietet die Lösung die Möglichkeit, Bestellinformationen für den Kunden auszuwählen und einzufügen. Der Anwender kann entweder keine, alle oder eine Auswahl der jüngsten bzw. ältesten Bestellungen einfügen, wie in Abbildung 24.26 dargestellt. Diese werden in Form einer Tabelle 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.

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 24.24 wieder ein, und es kann ein anderer Kunde ausgewählt werden.

Abbildg. 24.26 Über die Schaltfläche *Summe der markierten Zellen* können Werte addiert 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\Kap24* 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.

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:

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
```

Um die Steuerelemente eindeutig zu identifizieren, verwendet das *SmartDocument*-Interface wie die MOSTL den **SmartDocXMLTypeName** – eine Kombination des Namensraums plus Elementname: *namespace#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
```

Die Abschnittsüberschrift im Aufgabenbereich für ein bestimmtes Element (»Hilfe für das Kunde-Element« beispielsweise, siehe Abbildung 24.24) bestimmt die Eigenschaft **SmartDocXmlCaption**.

Diese kann sprachspezifisch nach LCID sein, um mehrsprachige Lösungen zu ermöglichen. Das folgende 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
```

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
```

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 24.24 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 Konstantwert 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

```

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 umgewandelt. Ein Beispiel für dessen Einsatz finden Sie in Listing 24.12.

```

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.KAP24beispiel.com/BSP24#Kunde202>«).

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 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.

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 24.4 zusammen. Dem *Type-Namen* müsste der Namensraum vorangestellt werden, Beispiel: »<http://www.KAP24Beispiel.com/BSP02#Kunde>«

**Tabelle 24.4** In diesem Beispiel verwendete Elementnamen, Element-IDs, Controls, ControlIndex-Werte, 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"
		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.

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.

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

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 24.7).

Beachten Sie, dass der Hilfetext über den Parameter Content, der durch ByRef an die Methode übergeben wurde, zurückgegeben wird.

**Listing 24.7** 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;rtig," & _
                        " 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>" & _
                        "<li>Die Anzahl Bestelldatens&uuml;tze festlegen: keine, alle," & _
                        " die &uuml;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 &uuml;testen oder die neuesten Bestellungen " & _
                        " einf&uuml;gen. Bei Auswahl der &uuml;testen oder die neuesten " & _
                        " bitte die Anzahl Datens&uuml;tze eingeben, sonst wird 1 (eins)" & _
                        " angenommen.</p></body></html>"
            Case Else
            End Select
        Case Else ' Englisch
    End Select
End Sub
```

Die Methode **PopulateListOrComboContent** in Listing 24.8 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 24.8** 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.

                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

```

**Listing 24.8** Füllt die Dropdownliste im Abschnitt für das Element *Kunde* mit Kundeninformationen aus einer Datenbank (*Fortsetzung*)

```
Finally:
Exit Sub

ErrorHandler:
    MsgBox Err.Description
    Resume Finally
End Sub
```

**PopulateRadioGroup.** Optionsschaltflächen können auch sprachspezifisch bezeichnet werden, wie das Listing 24.9 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 24.13.

**Listing 24.9** 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, _
    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.

Die Methode **OnListOrComboSelectChange** wird ausgeführt, wenn der Anwender einen Eintrag aus der Liste anwählt. Der Code in Listing 24.10 ruft die Hilfsprozedur *SetTextboxText* auf, die die Datenbankdaten des ausgewählten Kunden in die entsprechenden Textfelder schreibt.

Listing 24.10 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

```

Da alle Schaltflächen über die Methode **InvokeControl** gesteuert werden, wurden die eigentlichen Handlungen der Methode in Listing 24.11 in Hilfsprozeduren ausgelagert. Zwei davon, die wichtige Aspekte veranschaulichen, werden weiter unten vorgestellt.

Listing 24.11 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
    Exit Sub

    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
    End Select

```

**Listing 24.11** Die Methode *InvokeControl* wird durch Anklicken einer Schaltfläche im Aufgabenbereich *Dokumentaktionen* ausgelöst (Fortsetzung)

```
Case Else
End Select
End Sub
```

**Invoke\_KUNDE\_INSERT\_BUTTON.** Die Prozedur in Listing 24.12 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 24.12** 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
```



**Listing 24.12** Beim Anklicken der Schaltfläche *Kundendetails einfügen* werden Daten aus dem Aufgabenbereich in das Dokument geschrieben (Fortsetzung)

```

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

    ' 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))

```

**Listing 24.12** Beim Anklicken der Schaltfläche *Kundendetails einfügen* werden Daten aus dem Aufgabenbereich in das Dokument geschrieben (*Fortsetzung*)

```
' 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\_BESTELLUNGEN\_INSERT\_BUTTON.** Der Großteil dieser Prozedur beschäftigt sich damit, Werte zu verifizieren; diese Codezeilen haben wir in Listing 24.13 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 24.13** 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
```

Listing 24.13 Ein Klick auf die Schaltfläche *Bestellungen einfügen* führt diese Methode aus (Fortsetzung)

```

'''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`,`" & _
" 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 INCLUDETTEXT-Feldfunktion in das Element "Bestellungen" einfügen,
' um die XML-Datei in Bestellungen.xml zu transformieren. Die Feldfunktion
' anschließend auflösen.
strPad1 = Replace(strSolutionPath & strBestellungenDatei, "\", "\\")

```

**Listing 24.13** Ein Klick auf die Schaltfläche *Bestellungen einfügen* führt diese Methode aus (Fortsetzung)

```
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 7 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
- \n Stellt 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 ein 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\\Kap24\\Bsp24_02.xml"
\t "c:\\Beispiele\\Kap24\\Bsp24_02.xsl" \c XML }
```

### Die Transformation

Wie in der Diskussion zu Listing 24.13 erwähnt, werden die Informationen zu den Bestellungen über eine *IncludeText*-Feldfunktion ins Dokument geholt. Eine Transformation wandelt diese in eine Tabelle um.

Diese Transformation (*Bsp24\_02.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.
- Überflüssiges 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 *Bsp24\_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 einbauen (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.KAP24Beispiel.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 (*Bsp2402\_th*).

```
<w:styles>
  <w:style w:type="paragraph" w:styleId="Bsp2402_th">
    <w:name w:val="Bsp2402_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-00AA00C14882'
      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 (*Bsp24\_02.xsd*) befindet sich in Listing 24.14. 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 24.14 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.KAP24Beispiel.com/BSP02"
  xmlns="http://www.KAP24Beispiel.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">
                  <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 (*Bsp24\_02M.xml*) des Erweiterungspakets befindet sich in Listing 24.15. Der Großteil der Elemente ist im Abschnitt »So 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



Um die Lösung unter Word 2007 lauffähig zu machen, vergessen Sie nicht, den Inhalt des Elements `targetApplication` anzupassen. Für Word 2007 muss die Angabe lauten: *Word.Application.12*. Ansonsten müssen Sie auf die gleichen Sicherheitsmaßnahmen wie für eine MOSTL-Lösung achten.

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].[Klassename]« (in diesem Fall *Bsp02.SmartDoc*) im Abschnitt *HKEY\_CLASSES\_ROOT* und kopieren Sie den Wert seines *Clsid*-Schlüssels.

**Listing 24.15 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.KAP24Beispiel.com/BSP02</SD:uri>
  <SD:solution>
    <SD:solutionID>{CAA3479D-3E91-4b80-A56E-8A1CEFC50123}</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:filePath>Bsp24_02.xsd</SD:filePath>
    </SD:file>
  </SD:solution>
  <SD:solution>
    <SD:solutionID>{CAA3479D-3E91-4b80-A56E-8A1CEFC50123}</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>Bsp24_02.DLL</SD:filePath>
      <SD:CLSID>{913B7A1B-B324-4537-B2CB-9781C945E557}</SD:CLSID>
      <SD:regsvr32/>
    </SD:file>
    <SD:file>
      <SD:type>other</SD:type>
      <SD:version>1.0</SD:version>
      <SD:filePath>Bsp24_02.UDL</SD:filePath>
    </SD:file>
    <SD:file>
      <SD:type>other</SD:type>
      <SD:version>1.0</SD:version>
      <SD:filePath>Bsp24_02.XSL</SD:filePath>
    </SD:file>
  </SD:solution>
</SD:manifest>
```



## Die DLL in VB6 erstellen

Folgen Sie den in Kapitel 21 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-Lösungen befinden sich in der ZIP-Datei *Bsp24\_02.zip* auf der CD-ROM im Ordner *\Beispiele\Kap24*.

## 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://msdn2.microsoft.com/en-us/library/ms995347.aspx>.

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 *Bsp24\_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\Kap24\Bsp24_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:

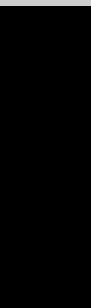
- Wie der Aufgabenbereich *Dokumentaktionen* dem Anwender kontextbezogene Hilfe und Steuerelemente zur Verfügung stellt (Seite 944 ff.).
- Die Entwicklung einer Smart Document-Lösung mit MOSTL (Seite 946 ff.).
- Das Erstellen eines Globally Unique Identifiers (GUID) (Seite 968 ff.).
- Die Realisierung einer Smart Document-Lösung mit einer in VB6 erstellten COM-DLL (Seite 970 ff.).

# Teil F

## Anhang

### In diesem Teil:

<b>Anhang A</b>	Namenskonventionen für Word-VBA	1001
<b>Anhang B</b>	Feldfunktionen	1005
<b>Anhang C</b>	Bezeichnungen von Symbolleisten	1013
<b>Anhang D</b>	Startoptionen von Word	1025
<b>Anhang E</b>	Bei Problemen – Word zurücksetzen	1029
<b>Anhang F</b>	Nützliche Links ins Internet	1033
<b>Anhang G</b>	Inhalt der CD-ROM	1035
<b>Anhang H</b>	Über die Autoren	1041



## Anhang A

# Namenskonventionen für Word-VBA

### In diesem Anhang:

Variablen	1002
Benutzerdefiniertes Dialogfeld	1002
Entwicklungsumgebung	1003
Wordobjekte	1003

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
Drehfeld	spn		spnAnzahl
Kontrollkästchen	chk		chkKopieErstellen
Listenfeld	lst		lstVerteiler ▶

Steuerelement	Präfix	Deklaration	Beispiel
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
Dokumenteigenschaft	prop	Dim prop As Office.DocumentProperty	Set prop = doc.BuiltInDocument Properties(1)
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	fttr	Dim ftr As Word.HeaderFooter	Set ftr = doc.Sections(1).Headers(1)
InlineShape	ils	Dim ils As Word.InlineShape	Set ils = doc.InlineShapes(1) ▶

Objekt	Präfix	Deklaration	Beispiel
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)
Symbolleistenlistenfeld	ccbx	Dim ccbx As Office.CommandBarComboBox	Set ccbx = cbr.Controls(25)
Symbolleiste	cbr	Dim cbr As Office.CommandBar	Set cbr = app.CommandBars("File")
Symbolleisten-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

# Feldfunktionen

**In diesem Anhang:**

Allgemeines zum Thema Feldfunktionen

1006

# 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 Klammerspaars 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>	GOTOBUTTON oder MACROBUTTON 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 Semikolon eingetragen, verwenden Sie dieses, 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.	{ = SUM(A1;A2) \# "0,00" } 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:

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
wdFieldFileSize	FileSize	Dateigrösse

**Tabelle B.3**    Gegenüberstellung der deutschen und englischen Feldfunktionsnamen *(Fortsetzung)*

<b>WdFieldType</b>	<b>Englischer Ausdruck</b>	<b>Alter deutscher Ausdruck</b>
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
wdFieldSaveDate	SaveDate	SpeicherDat

Tabelle B.3 Gegenüberstellung der deutschen und englischen Feldfunktionsnamen (Fortsetzung)

WdFieldType	Englischer Ausdruck	Alter deutscher Ausdruck
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-Hilfedateien 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
Initial	Caps

Tabelle B.4 Deutsche und englische Feldschalternamen (Fortsetzung)

Deutscher Name	Englischer Name
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 16 beschrieben, dienen englische Bezeichnungen als Indexwerte bei den Symbolleisten (CommandBars). In Tabelle C.1 sowie Tabelle C.2 sind 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
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

**Tabelle C.1** Sortiert nach den deutschen Begriffen (Fortsetzung)

Deutsche Bezeichnung	Englische Bezeichnung	Indexwert
Inline-ActiveX-Steuerelement	Inline ActiveX Control	112
Inlinebild	Inline Picture	66
Japanische Grußformeln	Japanese Greetings	40
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
Pop-upmenü: Horizontale Linie	Horizontal Line Popup	68
Positionsrahmen	Frames	61
Präzisionsausrichtung	Nudge	134
Rahmen	Borders	135
Rechtschreibung	Spelling	83
Reihenfolge	Order	133

Tabelle C.1 Sortiert nach den deutschen Begriffen (Fortsetzung)

Deutsche Bezeichnung	Englische Bezeichnung	Indexwert
Schatteneinstellungen	Shadow Settings	25
Schattierungsfarbe	Shading Color	137
Schriftfarbe	Font Color	136
Seitenansicht	Print Preview	16
Seriendruck	Mail Merge	11
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

**Tabelle C.1**    Sortiert nach den deutschen Begriffen (Fortsetzung)

Deutsche Bezeichnung	Englische Bezeichnung	Indexwert
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
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

Deutsche Bezeichnung	Englische Bezeichnung	Indexwert
3D-Einstellungen	3-D Settings	24
ActiveX-Steuerelement	ActiveX Control	101
Address Block Popup	Address Block Popup	110
Ausrichten oder verteilen	Align or Distribute	131
Anmerkungsstifte	Annotation Pens	128
Anmerkungsstifte	Annotation Pens	130
AutoFormen	AutoShapes	116
AutoSignaturmenü	AutoSignature Popup	91
AutoZusammenfassen	AutoSummarize	31
AutoText	AutoText	21
Statusleiste für Rechtschreibprüfung im Hintergrund	Background Proofing Status Bar	87
Standardformen	Basic Shapes	122
Blockpfeile	Block Arrows	119

Tabelle C.2 Sortiert nach den englischen Begriffen (Fortsetzung)

Deutsche Bezeichnung	Englische Bezeichnung	Indexwert
Rahmen	Borders	135
Legenden	Callouts	117
Canvas Popup	Canvas Popup	99
Zellausrichtung	Cell Alignment	138
Chinesische Schriftumwandlung	Chinese Translation	109
Zwischenablage	Clipboard	140
Kommentar	Comment	104
Nebeneinander vergleichen	Compare Side by Side	47
Verbindung	Connector	107
Verbindungen	Connectors	123
Steuerelement-Toolbox	Control Toolbox	33
Krümmen	Curve	95
Knoten krümmen	Curve Node	96
Kurvenabschnitt	Curve Segment	97
Datenbank	Database	4
Diagramm	Diagram	29
Diagramm	Diagram	106
Felder anzeigen	Display Fields	57
Dokumentlayout	Document Layout	46
Dokumentstruktur	Document Map	93
Zeichnen	Drawing	5
Zeichnungs- und Schreibstifte	Drawing and Writing Pens	127
Zeichnungs- und Schreibstifte	Drawing and Writing Pens	129
Zeichnungsbereich	Drawing Canvas	27
Initial	Drop Caps	54
Grafik bearbeiten	Edit Picture	8
E-Mail	E-mail	44
Endnoten	Endnotes	55
Umschlag	Envelope	141
Entwurfsmodus beenden	Exit Design Mode	32
Erweiterte Formatierung	Extended Formatting	20
AutoText-Feld	Field AutoText	92

**Tabelle C.2**    Sortiert nach den englischen Begriffen (Fortsetzung)

Deutsche Bezeichnung	Englische Bezeichnung	Indexwert
Feldanzeige für Listenzahlen	Field Display List Numbers	58
Felder	Fields	56
Füllfarbe	Fill Color	124
Unverankerte Grafik	Floating Picture	98
Flussdiagramm	Flowchart	118
Schriftfarbe	Font Color	136
Absatzschriftart	Font Paragraph	80
Untermenü Schrift	Font Popup	79
Fußnoten	Footnotes	60
Formularfelder	Form Fields	59
Konsistenz formatieren	Format consistency	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
Format	Formatting	2
Formular	Forms	6
Frameeigenschaften	Frame Properties	89
Frames	Frames	43
Positionsrahmen	Frames	61
Ganzer Bildschirm	Full Screen	7
Funktionstastenanzeige	Function Key Display	37
Grammatik	Grammar	84
Grammatik (2)	Grammar (2)	85
Greeting Line Popup	Greeting Line Popup	111
Kopf- und Fußzeile	Header and Footer	14
Kopfzeilen	Headings	62
Popupmenü: Horizontale Linie	Horizontal Line Popup	68
Hyperlink-Kontextmenü	Hyperlink Context Menu	90
Freihandanmerkungen	Ink Annotations	49
Freihandkommentar	Ink Comment	50
Freihandkommentar	Ink Comment	115
Freihandzeichnung und -schrift	Ink Drawing and Writing	48



Tabelle C.2 Sortiert nach den englischen Begriffen (Fortsetzung)

Deutsche Bezeichnung	Englische Bezeichnung	Indexwert
Inline-ActiveX-Steuerelement	Inline ActiveX Control	112
Inline Zeichnungsbereich	Inline Canvas	67
Inlinebild	Inline Picture	66
Form einfügen	Insert Shape	125
Japanische Grußformeln	Japanese Greetings	40
Linienfarbe	Line Color	126
Linien	Lines	121
Verknüpfte Überschriften	Linked Headings	63
Verknüpfte Tabelle	Linked Table	76
Verknüpfter Text	Linked Text	78
Listen	Lists	65
Macros book	Macros book	53
Seriendruck	Mail Merge	11
Zentraldokument	Master Document	12
Menüleiste	Menu Bar	41
Microsoft	Microsoft	13
Präzisionsausrichtung	Nudge	134
OLE-Objekt	OLE Object	100
Onlinebesprechung	Online Meeting	143
Reihenfolge	Order	133
Organigramm	Organization Chart	28
Organization Chart Popup	Organization Chart Popup	105
Gliederung	Outlining	15
Outlook-E-Mail lesen	Outlook Read Mail	35
Outlook-E-Mail senden	Outlook Send Mail	36
Grafik	Picture	26
Seitenansicht	Print Preview	16
Nachricht lesen	Read Mail	18
Lesemoduslayout	Reading Layout	45
Aktualisieren	Refresh	42
Überarbeiten	Reviewing	30
Drehungsmodus	Rotate Mode	103

**Tabelle C.2**    Sortiert nach den englischen Begriffen *(Fortsetzung)*

Deutsche Bezeichnung	Englische Bezeichnung	Indexwert
Drehen oder kippen	Rotate or Flip	132
Skriptanchor-Popupmenü	Script Anchor Popup	64
E-Mail senden	Send Mail	19
Schattierungsfarbe	Shading Color	137
Schatteneinstellungen	Shadow Settings	25
Formen	Shapes	94
Rechtschreibung	Spelling	83
Standard	Standard	1
Sterne & Banner	Stars & Banners	120
Aufzeichnung beenden	Stop Recording	10
System	System	142
Tabellenzellen	Table Cells	70
Tabellenüberschriften	Table Headings	71
Tabellenlisten	Table Lists	72
Tabellengrafiken	Table Pictures	73
Tabellentext	Table Text	74
Tabellen	Tables	69
Tabellen und Rahmen	Tables and Borders	3
Aufgabenbereich	Task Pane	51
Text	Text	77
Textfeld	Text Box	34
Textfluss	Text Wrapping	139
Änderungen nachverfolgen	Track Changes	88
Überarbeitungsanzeige	Track Changes Indicator	108
Visual Basic	Visual Basic	9
Web	Web	22
Webtools	Web Tools	38
Ganze Tabelle	Whole Table	75
Wörter zählen	Word Count	39
Word für Windows 2.0	Word for Windows 2.0	17

Tabelle C.2 Sortiert nach den englischen Begriffen (Fortsetzung)

Deutsche Bezeichnung	Englische Bezeichnung	Indexwert
WordArt	WordArt	23
WordArt-Kontextmenü	WordArt Context Menu	102
XML-Fehleroptionen	XML Error Options	114
XML Strukturknoten-Popup	XML Structure Node Popup	113



## Anhang D

# Startoptionen von Word

**In diesem Anhang:**

Die Befehlszeilenargumente von Word

1026

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 folgenden 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
<i>winword.exe</i>	Ein neues Word-Fenster mit einem neuen leeren Dokument wird erzeugt, wobei die bestehende Instanz von Word genutzt wird.
<i>winword.exe /q</i>	Startet eine neue Instanz von Word ohne den so genannten Splashscreen anzuzeigen.
<i>winword.exe "Dateiname1" "Dateiname2" "DateinameX"</i>	Ein neues Word-Fenster wird für jede Datei erzeugt. Nach erfolgreichem Start werden die entsprechenden Dokumente geöffnet.
<i>winword.exe /n</i>	Startet eine neue Instanz von Word ohne ein leeres Dokument zu generieren.
<i>winword.exe /w</i>	Startet eine neue Instanz von Word und generiert ein leeres Dokument.
<i>winword.exe /t"Vorlage"</i>	Startet eine neue Instanz von Word. Anschließend wird ein neues Dokument basierend auf der angegebenen Dokumentvorlage generiert. Wobei weder das Marko <b>AutoNew</b> abgearbeitet noch das Ereignis <b>DocumentNew</b> eintritt.
<i>winword.exe /z"Vorlage"</i>	Startet eine neue Instanz von Word. Anschließend wird ein neues Dokument basierend auf der angegebenen Dokumentvorlage generiert. Zusätzlich wird das Marko <b>AutoNew</b> abgearbeitet und das Ereignis <b>DocumentNew</b> wird gefeuert. Das z-Argument steht erst ab Word 2007 zur Verfügung.
<i>winword.exe /m</i>	Startet eine neue Instanz von Word. Es werden keine <b>AutoExec</b> -Makros ausgeführt.
<i>winword.exe /m"Makroname"</i>	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 <b>AutoExec</b> -Makros ausgeführt. Das definierte Makro kann sich jedoch auch auf einen internen Word-Befehl beziehen (beispielsweise <i>/Datei1</i> bzw. <i>/mFile1</i> ).
<i>winword.exe /a</i>	Startet eine neue Instanz von Word ohne die hinterlegten Add-Ins zu aktivieren.
<i>winword.exe /safe</i>	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).
<i>winword.exe /r</i>	Word wird nicht gestartet, sondern führt eine Neuregistrierung in der Windows-Registrierung durch.

Tabelle D.1 Zusammenfassung der Startoptionen von Word (Fortsetzung)

Syntax	Bedeutung
<i>winword.exe /l"Addinname"</i>	Startet eine neue Instanz von Word, zusätzlich wird das entsprechende Add-In geladen und aktiviert.
<i>winword.exe /</i>	Startet eine neue Instanz von Word, wenn nur ein "/" oder eine unbekannte Befehlszeilenoption angegeben wird.

Grundsätzlich gelten für alle Argumente die nachstehenden Punkte:

- Lange Dateinamen mit oder ohne Pfadangaben, welche Leerschläge enthalten, müssen in Anführungszeichen eingebettet 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 des abgesicherten 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>)





## 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 schlechter Trost.

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 bis zur Word-Version 2003 den Menübefehl *Extras/Optionen* an. Wechseln Sie auf die Registerkarte *Speichern* und aktivieren Sie das Kontrollkästchen *AutoWiederherstellen-Info*. Unter Word 2007 klicken Sie zunächst auf die *Office-Schaltfläche* und anschließend auf die Schaltfläche *Word-Optionen*. Öffnen Sie dann die Kategorie *Speichern* und aktivieren Sie das Kontrollkästchen *AutoWiederherstellen-Informationen speichern alle xx Minuten*.

Die Problematik liegt jedoch ganz anders, wenn sich Word bei jedem Programmstart gleich wieder verabschiedet oder dies regelmäßig während dem Arbeiten mit dem Programm geschieht. Zwei Aspekte sind meistens 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 Absturz bringen.

Wie ein neues Original der Standardvorlage *Normal.dot* erzeugt wird, wurde bereits in Kapitel 13 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 Datei 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-Key in der Windows-Registrierung



regedit.exe

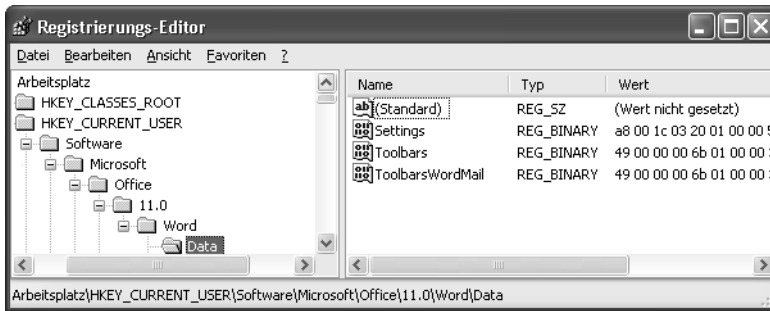
Word hinterlegt viele Einstellungen zu den gewählten Optionen und zur aktuellen Programmumgebung in der Windows-Registrierung. Diese Daten werden im Schlüssel *Data* in ein paar wenigen Einträgen mit entsprechend großen binären Datenfeldern abgespeichert.

Können diese Daten nicht sauber in der Windows-Registrierung gespeichert werden, was zum Beispiel bei einem Programmabsturz der Fall sein kann, besteht die Möglichkeit, dass diese nicht mehr konsistent sind. Diese Inkonsistenzen können wiederum dazu führen, dass Word regelmäßig abschmiert.

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 Ordner und wählen Sie den Menübefehl *Datei/Exportieren*, um eine Sicherheitskopie zu erstellen. In einem zweiten Schritt wird der Schlüssel entfernt. Dazu muss der Menübefehl *Bearbeiten/Löschen* angewählt werden.

Abbildg. 24.27 Data-Key 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, können die alten Werte jederzeit wieder aktiviert werden.



## 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://msdn2.microsoft.com/en-us/library/aa167879.aspx>

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 reiche Anzahl an 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://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 rundum Word.

<http://www.aboutvb.de/vba/vba.htm>

Eine deutschsprachige Internet-Seite mit qualitativ hoch stehenden Beiträgen, die sich an den professionellen Entwickler richten. Nebst 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.

<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 Anhang:**

Die Beispieldateien der CD-ROM zum Buch	1036
Zusatz-Software: Die Dateien im Anhang	1038

# 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, Kap01_CS.zip
2	\Beispiele\Kap02	Bsp02_01.doc, Bsp02_02.doc, Bsp02_03.doc, Bsp02_04.doc, Bsp02_05.doc, Bsp02_06.doc, Bsp02_07.doc, DebugTest.dot, Kap02_CS.zip
3	\Beispiele\Kap03	Bsp03_1.doc, Bsp03-1.ini
4	\Beispiele\Kap04	Bsp04_01.doc
5	\Beispiele\Kap05	Bsp05_01_Document.doc, Bsp05_01_Find.dot, Bsp05_01_System.doc, Bsp05_01_Template.doc, Bsp05_01App.doc, Bsp05_01Range.doc, Bsp05_01Range_Test.doc, Bsp05_02_Find.doc, Bsp05_02_Find.dot, Bsp05_02_Find.txt, Bsp05_02_Template.doc, Bsp05_02Range.doc, Bsp05_03_Find.doc, Bsp05_03Range.doc, Bsp05_04_Find.doc, Bsp05_04Range.doc, Bsp05_05_Find.doc, Bsp05_06_Find.doc, Bsp05_07_Find.doc, Seifenblasen.bmp, Kap05_CS.zip
6	\Beispiele\Kap06	Bsp06_01_Graf.doc, Bsp06_01_Num.doc, Bsp06_01_Style.doc, Bsp06_01_Zentraldokument.doc, Bsp06_01a.doc, Bsp06_02_Graf.doc, Bsp06_02_Num.doc, Bsp06_02_Zentraldokument.doc, Bsp06_02a.doc, Bsp06_03_Graf.doc, Bsp06_03_Num.doc, Bsp06_03_Zentraldokument.doc, Bsp06_04_Graf.doc, Bsp06_04_Zentraldokument.doc, Bsp06_05_Graf.doc, Bsp06_05_Zentraldokument.doc, Bsp06_06_Graf.docm, Bsp06_07_Graf.docm, Kap06-BuildingBlocks.docm, Kap06_CS.zip
7	\Beispiele\Kap07	anreden.xml, Brief.xml, Brief_Offerte.xml, Bsp07_01_Bookmark.doc, Bsp07_01_CC.dotm, Bsp07_01_Field.doc, Bsp07_01_Form.doc, Bsp07_01_SD.doc, Bsp07_01_Table.doc, Bsp07_02_CC.docx, Bsp07_02_CC.dotx, Bsp07_02_CC.xml, Bsp07_02_Field.doc, Bsp07_02_Field.xls, Bsp07_02_SD.doc, Bsp07_02_Table.doc, Bsp07_03_CC.dotm, Bsp07_03_SD.doc, Bsp07_03_Table.doc, Bsp07_04_CC.docm, Bsp07_04_SD.doc, Bsp07_04_Table.doc, Bsp07_05_CC.docm, Bsp07_05_Table.doc, Bsp07_06_CC.dotm, grussformel.xml, Kap07_CS.zip



Tabelle G.1 Übersicht über die Beispieldateien auf der CD-ROM zum Buch (Fortsetzung)

Kapitel	Speicherort	Name der Dateien
8	\Beispiele\Kap08	Bsp_08-1.dot, Bsp08_03.doc, Bsp08_Beispiel.doc
9	\Beispiele\Kap09	Bsp09_01.doc, Bsp09_02.doc, Bsp09_03.doc, MSCOMCTL.OCX, Kap09_VB.zip
10	\Beispiele\Kap10	Bsp10_01.doc, Bsp10_02.xls, Bsp10_03.dot, Bsp10_04.dot, Bsp10_05.dot, Bsp10_06.doc, Bsp10_07.xls, Bsp10_Briefdot, Bsp_Addin_VB.zip, Bsp_VSTO_VB.zip, Kap10_VB.zip, Kap10_CS.zip
11	\Beispiele\Kap11	Bsp_Demo.ppt, Bsp_Demo.vsd, Bsp_Demo.xls, Bsp11_01.doc, Bsp11_01a.doc, Bsp11_02a.dot, Bsp11_02b.doc, Bsp11_03.doc, Bsp11_04.dot, Bsp11_05.dot
12	\Beispiele\Kap12	Bsp12_01.doc, Bsp12_01.xls, Bsp12_02.doc, Bsp12_02_2007.docm, Bsp12_03.doc, Bsp12_04.doc, Bsp12_05.doc, Bsp12_06.doc, Kap12.zip
13	\Beispiele\Kap13	Bsp13_01.doc, Das Handbuch.xml, Kap13_CS.zip
15	\Beispiele\Kap15	Bsp15_01.doc, Bsp15_02.doc, Bsp15_03.doc, Bsp15_03.ini, Bsp15_03.xml, Bsp15_03a.txt, Bsp15_03b.txt, Bsp15_04.doc, wdDialogsKonstanten-Word2003.doc, wdDialogsübersicht_Word2003.doc
16	\Beispiele\Kap16	Bsp16_01.doc, Bsp16_02.doc, Bsp16_03.doc, Bsp16_04.doc, Bsp16_05.doc, Bsp16_06.doc
17	\Beispiele\Kap17	Beispiel1_customUI.xml, Beispiel2_customUI.xml, Beispiel3_customUI.xml, Bei-spiel4_customUI.xml, bleistift.jpg, Eightball.png, heft.jpg, klammer.jpg, kugel-schreiber.jpg, ordner.jpg, papier.jpg, Produkte.xml, radiergummi.jpg, Rib-bon_Beispiel1.dotm, Ribbon_Beispiel2.dotm, Ribbon_Beispiel3.dotm, Rib-bon_Beispiel4.dotm, Ribbon_Beispiel5.docm, umschlag.jpg
18	\Beispiele\Kap18	AddinBeispiel.dll, AddinBeispiel_AlleVersionen.dll, Bsp18_01.dot, Bsp18_02.dot, Bsp18_03.dot, COMAddin_Test.dot
19	\Beispiele\Kap19	Bsp19_01.doc, Bsp19_02.doc, Bsp19_03.doc, Bsp19_04.dot, Bsp19_05.dotm
20	\Beispiele\Kap20	Bsp20_01.dot
21	\Beispiele\Kap21	Bsp21_01.doc, Bsp21_02.doc, Bsp21_Referenz.dot, modImportModul.bas, mod-SearchProz.bas
22	\Beispiele\Kap22	Bsp22_01.htm, Bsp22_02.xml, Bsp22_02a.xml, Bsp22_02b.xml, Bsp22_03.xml, Bsp22_03.xsl, Bsp22_04.xml, Bsp22_04.xsl, Bsp22_05.xsd, Bsp22_06.xsd, Bsp22_07.xsd, Bsp22_08.xsd, Bsp22_09.xsd, Bsp22_10.xml, Bsp22_11.xml, Bsp22_12.xml, Bsp22_13.xml, Bsp22_14.xsl, Bsp22_15.xml, Bsp22_16.xml, Bsp22_17.xml, Bsp22_18.xsd, Bsp22_19.doc, Bsp22_20.xsd, Bsp22_21.xml, Bsp22_22.xml, Bsp22_23.xsl, Bsp22_25.doc, Bsp22_25.txt
23	\Beispiele\Kap23	Bsp23_02.doc, Bsp23_02.xml, Bsp23_02_.p.htm, Bsp23_02_dl.htm, Bsp23_02_ip.htm, Bsp23_02_mi.htm, Bsp23_02_up_v2.xml, Bsp23_02_up_v3.xml, Bsp23_02_v2.xml, Bsp23_02_v3.xml, Bsp23_04.doc, Bsp23_04.xsl, planet.xml, Bsp23_03_VB6.zip, VSTO_ST_VB.zip

Tabelle G.1 Übersicht über die Beispieldateien auf der CD-ROM zum Buch (Fortsetzung)

Kapitel	Speicherort	Name der Dateien
24	\Beispiele\Kap24	Bsp24_01.reg, Bsp24_01.xsd, Bsp24_01.xsl, Bsp24_01A.xml, Bsp24_01B.xml, Bsp24_01M.xml, Bsp24_01N.xml, Bsp24_01S.xml, Bsp24_02.dll, Bsp24_02.dot, Bsp24_02.udl, Bsp24_02.xsd, Bsp24_02.xsl, Bsp24_02M.xml, smartdoc.cls, Bsp24_02.zip
I	\Beispiele\Kapl	Bspl_01.dot
II	\Beispiele\KapII	BsplI_Symbolleiste.dot
III	\Beispiele\KapIII	BsplIII_01.doc, BsplIII_02.doc
IV	\Beispiele\KapIV	BsplIV_01.dot, BsplIV_02.dot, BsplIV_03.dot, BsplIV_04.dot, BsplIV_05.dot, BsplIV_06.dot, BsplIV_Logo.bmp, BsplIV_Wasserzeichen.bmp
V	\Beispiele\KapV	BspV_01.doc
VI	\Beispiele\KapVI	BspVI_01.doc, BspVI_01.dot, KapVI_Beispiel.doc
VII	\Beispiele\KapVII	BspVII_01.dot
VIII	\Beispiele\KapVIII	BspVIII_01.doc, BspVIII_02.doc, WordArt.doc
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 Thema sind noch weitere Information 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* finden Sie eine Zusammenstellung aller internen Word-Befehle (deutsch und englisch).
- 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.
- Ein Editor, um den Ribbon auf die Bedürfnisse des Anwenders anzupassen, befindet sich zusammen mit weiteren Dateien zum Thema im Ordner *\Beilagen\Ribbon*.
- Microsoft stellt verschiedene Software Development Kits (SDK) auf MSDN zum Herunterladen bereit. In diesem Buch wurden die *Smart Tag*- sowie *SmartDocument*-SDKs 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.
- 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.
- Das Content Control Toolkit ist ein eigenständiges Programm zum Verknüpfen von Content Controls mit XML-Datensätzen. Die Dokumentation und das Installationsprogramm befinden sich im Ordner *\Beilagen\Wd2007ContentCtrolToolkit*.
- 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 zu gegriffen werden kann. Die Datei befindet sich im Ordner *\Beilagen\Word95 Wordbasic Hilfe*.
- Zur Bearbeitung von XML-Daten kann ein spezieller Editor verwendet werden. Als mögliches Produkt kann der XML Notepad 2007 erwähnt werden. Das Installationsprogramm befindet sich im Ordner *\Beilagen\XML Notepad*.
- 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.
- 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, Interne Word-Befehle.xls
Namenskonventionen	<i>\Beilagen\Namenskonventionen für VBA 5.0-6.0</i>	./.
Ribbon	<i>\Beilagen\Ribbon</i>	2007OfficeControlIDsExcel2003.exe, 2007OfficeControlIDsExcel2007.exe, imageMso.xlsx, Office2007IconsGallery.xlsm, OfficeCustomUIEditorSetup.msi
Smart Document	<i>\Beilagen\SmartDocument SDK</i>	sdocsdk.msi
Smarttag	<i>\Beilagen\SmartTagSDK</i>	mstagsdk.msi
Suchen und Ersetzen	<i>\Beilagen\SuchenErsetzen</i>	SuchenErsetzen.pdf
True Type-Schriften	<i>\Beilagen\TrueTypeFont Eigenschaften</i>	setup.exe
Content Control Toolkit	<i>\Beilagen\Word2007ContentControlToolkit</i>	documentation.pdf, Setup.msi
Wordbasic Hilfe	<i>\Beilagen\Word95 Wordbasic Hilfe</i>	Readme.txt, Wrdbasic.hlp

**Tabelle G.2** Übersicht zu den weiteren Beilagen auf der CD-ROM *(Fortsetzung)*

Thema	Speicherort	Name der Dateien
XML Notepad	\Beilagen\XML Notepad	OfficeWordWordMLtoXSL-FOSample.exe, XmlNotepad.msi
XML Reference Schemas	\Beilagen\XML Reference Schemas	xsdref.msi
Seriendruck	\Beilagen\Zusatzmaterial Seriendruck	MSQuery.pdf, Odbc.pdf, Oledb.pdf, SD_Verbindungen.pdf, SD_Verbindungen.zip, Sql.pdf

## Anhang H

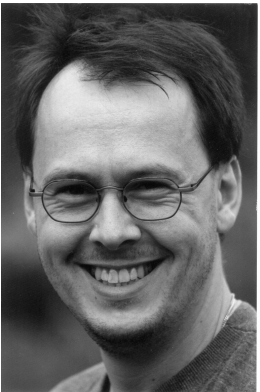
# Über die Autoren



**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 rundum 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, Smart Documents und VSTO (was zugleich die ersten Schritte in die Welt von .NET Framework bedeutete). Ihre Lieblingsprojekte sind Gesamtlösungen, in welchen sie als Mitglied eines Entwicklerteams für die Word-Integration sorgt.

Das vorliegende Werk ist nicht die erste Veröffentlichung Cindy Meisters. Nebst 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 Webseite <http://homepage.swissonline.ch/cindymeister>.

Cindy Meister wurde im Jahr 1996 erstmals in das MVP-Programm aufgenommen und erhielt seither jährlich diese Auszeichnung. Am 1. Oktober 2007 wurde sie erstmals als MVP für VSTO anerkannt. Sie kann über die E-Mail-Adresse [cindymeister@swissonline.ch](mailto:cindymeister@swissonline.ch) erreicht werden.



**Christian Freßdorf** arbeitet seit mehr als 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 mit WordBasic Funktionen zur Steuerung und Generierung von Onlinehilfe-Projekten im Vordergrund, konnten mit Einführung von VBA auch komplexere Abläufe realisiert werden.

Seit sieben 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 über [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 sieben Jahren sein Wissen auf *microsoft.public.de.word.vba* weiter und ist seit 2002 aktiver Microsoft MVP. Fragen zum vorliegenden Buch können ihm auf *wurzel2@bluemail.ch* gestellt werden.



**Peter Jamieson** hat seit 1979 in vielen Bereichen der Informatik, meistens in Großbritannien, aber auch in Benelux und anderswo gearbeitet. Es 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 ist 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* kontaktieren.





# Verzeichnis zum Objektmodell

## A

ActiveDocument 189  
Add-In-Objekt  
    Add-Methode 205, 504  
    Delete-Methode 205, 504  
Application-Objekt 177  
    ActivePrinter 184, 201  
    AutomationSecurity 179, 450, 580  
    BackgroundPrintingStatus 198  
    ButtonClicks 384  
    CentimetersToPoints 324  
    CustomizationContext 659  
    DisplayAlerts 179  
    Exists 598  
    KeysBoundTo 770  
    MacroContainer 835  
    MailingLabel 439  
    Options *siehe* Options-Objekt 180  
    PathSeparator 114, 180, 192  
    Quit 597  
    RestrictLinkedStyles 296  
    Run-Methode 205, 500  
    ScreenUpdating 178, 219  
    Visible 178  
    WordBasic *siehe* WordBasic 183  
AttachedTemplate *siehe* Document-Objekt 192  
AutomationSecurity *siehe* Application-Objekt 179  
AutoTextEntry-Objekt  
    Add-Methode 103  
    Insert-Methode 204, 230  
    Name 103  
    StyleName 103

## B

BackgroundPrintingStatus *siehe* Application-Objekt  
    BackgroundPrintingStatus 198  
Bold 209  
Bookmark-Objekt 340  
    Add 341  
    Exists 344  
    Range.Text 344  
BuildingBlockEntries-Objekt 274  
    Add-Methode 274

BuildingBlocks-Objekt 275, 277  
    BuildingBlockType-Objekt 275  
    BuildingBlockTypes, *siehe* BuildingBlockTypes-Objekt 277  
    Category 274, 277  
    Description 274  
    Eigenschaften 274  
    Insert-Methode 277  
    LoadBuildingBlocks-Methode 270  
    Name 274  
    Type 274, 275  
    Value 274  
    wdBuildingBlockTypes-Konstanten 268  
BuildingBlockTypes-Objekt 277  
    wdBuildingBlockTypes-Konstanten 278

## C

Collapse *siehe* Range-Objekt 210  
CommandBar-Objekt 708  
    ActionControl 716  
    AdaptiveMenu 713  
    AdaptiveMenus 713  
    Add-Methode 719  
    Control 713  
    DisableAskAQuestionDropdown 713  
    DisableCustomize 712  
    Enabled 712  
    Height 709  
    Index 709  
    Left 709  
    Name 709  
    NameLocal 709  
    Protection 712  
    RowIndex 710  
    Top 709  
    Type 709  
    Visible 712  
    Width 709  
ComputeStatistics *siehe* Document-Objekt 191  
ContentControl-Objekt 399  
    Add 419  
    ContentControlAfterAdd-Ereignis 420  
    ContentControlBeforeContentUpdate-Ereignis 420  
    DateCalendarType 403

ContentControl-Objekt (*Fortsetzung*)

DateDisplayFormat 403  
 DateDisplayLocale 403  
 DateStorageFormat 403  
 DefaultTextStyle 402  
 DropDownListEntries 403  
 ID 417, 424  
 LockContentControl 419  
 LockContentControl 402  
 LockContents 402  
 Multiline 402  
 PlaceholderText 401  
 SelectContentControlsByTag-Methode 402, 417  
 SelectContentControlsByTitle-Methode 402, 417  
 SetPlaceholderText-Methode 401  
 ShowingPlaceholderText 401  
 Temporary 402  
 WdContentControlType 401

Control-Objekt 713

Add-Methode 715  
 Caption 713  
 CommandBar 714  
 CommandBarComboBox 717  
 CommandBarComboBox, AddItem-Methode 718  
 Enabled 714  
 Execute-Methode 713  
 FindControl 713  
 ID 713  
 OnAction 715  
 Priority 713  
 Visible 714

CustomXMLPart-Objekt

Add 415  
 CustomXMLNode.AppendChildSubtree-Methode 425  
 LoadXML 415  
 NamespaceManager.LookupPrefix-Methode 424  
 SelectSingleNode 415  
 SetMapping 415  
 SetMappingByNode 415

## D

DefaultFliePath *siehe* Options-Objekt 182

Designer-Objekt

Controls 842  
 Controls.Add-Methode 842

Dialogfelder

DefaultTab-Eigenschaft 695  
 Dialogs-Auflistung 690

Dialogs-Objekt

Display-Methode 691  
 Execute-Methode 692  
 Show-Methode 691  
 Timeout-Parameter 691

Update-Methode 692  
 wdDialogBuildingBlockOrganizer 270  
 wdWordDialog-Konstanten 695

DisplayAlerts *siehe* Application-Objekt 179

Document-Objekt 189

Add-Methode 189, 192, 203  
 AttachedTemplate 192  
 AutoFormatOverride 286  
 CheckNewSmartTag 936  
 ComputeStatistics 191  
 DocumentProperty 650  
 EmbedSmartTags 936  
 FormattingShowClear-Methode 802  
 FullName 192  
 MailMerge 428  
 MailMerge.MailMergeType 428  
 Name 192  
 Open-Methode 189, 507  
 Path 192  
 PrintOut-Methode 196, 507  
 RecheckSmartTags 936  
 RemoveSmartTag 936  
 SaveAs-Methode 194  
 Saved 194  
 Save-Methode 194  
 SmartTagsAsXMLProps 936  
 StyleSort 804  
 Type 191, 192  
 Unprotect-Methode 395  
 Variable 290, 653

## E

Ereignisse

ContentControlBeforeDelete 420  
 ContentControlBeforeStoreUpdate 420  
 ContentControlOnEnter 420  
 ContentControlOnExit 420

Events 448

Document\_Close 451  
 Document\_New 451  
 Document\_Open 451  
 DocumentBeforeClose 467  
 DocumentBeforePrint 469  
 DocumentBeforeSave 468  
 DocumentBeforeSync 471  
 DocumentChange 466  
 DocumentOpen 466  
 NewDocument 465  
 WindowActivate 458  
 WindowBeforeDoubleClick 460  
 WindowBeforeRightClick 461  
 WindowDeactivate 459  
 WindowSelectionChange 463  
 WindowSize 459  
 WithEvents 455

**F**

Field-Objekt 378  
  Add-Methode 380  
  Code 381  
  DoClick 384  
  Kind 379, 383  
  Lock 379  
  Result 381  
  Update-Method 378  
  UpdateSource 267  
FileDialog-Objekt 696  
  DialogType-Eigenschaft 697  
  Execute-Methode 698  
  FileDialogFilters.Add-Methode 700  
  FileDialogFilters.Clear-Methode 700  
  FileDialogFilters-Auflistung 699  
  Filters-Eigenschaft 699  
  msoFileDialogFilePicker-Dialogfeld 704  
  msoFileDialogFolderPicker-Dialogfeld 704  
  msoFileDialogOpen-Dialogfeld 701  
  msoFileDialogOpen-Konstanten 697  
  msoFileDialogSaveAs-Dialogfeld 702  
  Show-Methode 698  
Find  
  ClearFormatting 222  
  Execute-Methode 230  
  Format 222, 287  
  Forward 222, 223  
  Found 228  
  HitHighlight-Methode 222  
  IgnorePunct 222  
  IgnoreSpace 222  
  MatchAllWordForms 222  
  MatchCase 222  
  MatchPrefix 222  
  MatchSoundsLike 222  
  MatchSuffix 222  
  MatchWholeword 222  
  MatchWildcards 222  
  Replacement.Text 222  
  Style 287, 348  
  Text 222  
  Wrap 222, 223  
Formfield-Objekt 392  
  EntryMacro 397  
  ExitMacro 397  
  Result 393  
  Valid 393  
Frames-Objekt  
  Add 319

**G**

GotFocus 618

**H**

HeaderFooter-Objekt 244, 256  
  Exists 260  
  IsHeader, IsFooter 262  
  LinkToPrevious 259  
  PageNumbers 260

**I**

IIF-Funktion 702  
InlineShape-Objekt 309  
  AddOLEObject 591  
  AddPicture-Method 310  
  ConvertToInlineShape-Methode 316  
  ConvertToShape-Methode 316  
  LinkFormat 316  
  OLEFormat 591  
  OLEFormat.DoVerb-Methode 591  
Italic 209

**K**

KeyBinding-Objekt 763  
  Add-Methode 772  
  BuildKeyCode 765  
  ClearAll-Methode 771  
  Clear-Methode 771  
  Count 763  
  Disable-Methode 772  
  FindKey 765  
  Key 765  
  KeyString 766  
  Rebind-Methode 773

**L**

LostFocus 618

**M**

MailMerge-Objekt  
  DataSource 428  
  DataSource.ActiveRecord 428  
  DataSource.Included 430  
  DataSource.RecordCount 429  
  DataSource.SetAllIncludedFlags-Methode 431  
  Destination 435  
  Execute-Methode 435

MailMerge-Objekt (*Fortsetzung*)

- FindRecord 40, 432
- MailMergeWizardStateChange-Ereignis 434
- OpenDataSource-Methode 441
- ShowSendToCustom-Methode 434
- ShowWizard-Methode 434

## Mso-Konstantwert

- MsoBarProtection 712
- MsoPresetTextEffect 612
- MsoPresetTextEffectShape 612

## MsoKonstantwert

- MsoLanguageID 180

**N**

## NewDocument-Objekt 793, 796

- Add-Methode 793
- Remove-Methode 794

## NormalTemplate 204, 628

## NormalTemplate-Objekt 192

**O**OperatingSystem *siehe* System-Objekt 175

## Options-Objekt 180

- DefaultFilePath 182
- PrintBackground 197
- WPDNavKeys 181
- WPHelp 181

**P**

## PageSetup-Objekt 244, 248

- DifferentFirstPageHeaderFooter 253
- FirstPageTray 254
- Gutter 251
- Header-, FooterDistance 251
- MarginBottom 250
- MarginLeft 250
- MarginRight 250
- MarginTop 250
- OddAndEvenPagesHeaderFooter 253
- Orientation 250
- OtherPagesTray 254
- PaperSize 249
- TextColumns 252

## Pages-Objekt 214

PathSeparator *siehe* Application-Objekt 192PrintBackground *siehe* Options-Objekt 197PrintOut *siehe* Dokument-Objekt 196**R**

## Range-Objekt 204, 207

- Calculate-Methode 219
- Collapse-Methode 210
- ConvertToTable-Methode 351
- Duplicate 217, 228
- Find *siehe* Find 222
- Font.Color 356
- Font.Name 356
- FormattedText 218, 348
- GoTo-Methode 214
- Information 215
- InRange 383
- InRange-Methode 216
- InsertAfter-Methode 211
- InsertBefore-Methode 211
- InsertBreak-Methode 211
- InsertParagraphAfter-Methode 211
- InsertParagraphBefore-Methode 211
- IsEqual 216
- MoveEnd-Methode 212
- MoveEndUntil-Methode 212
- MoveEndWhile-Methode 212
- Move-Methode 212
- MoveStart-Methode 212
- MoveStartUntil-Methode 212
- MoveStartWhile-Methode 212
- MoveUntil-Methode 212
- MoveWhile-Methode 212
- Paragraph.Border 357
- ParagraphFormat.FirstlineIndent 356
- Select-Methode 208
- Shading 357
- Text 210
- TextRetrievalMode 217, 234

Range-Objekt *siehe* InStory-Methode 216

## Reference-Objekt

- AddFromFile-Methode 850
- AddFromGUID-Methode 850
- Count 849
- IsBroken 854
- Remove-Methode 854

**S**Save *siehe* Document-Objekt 194SaveAs *siehe* Document-Objekt 194Saved *siehe* Document-Objekt 194ScreenUpdating *siehe* Application-Objekt 178

## Section-Objekt 242

- ProtectedForForms 396

SelectCurrent *siehe* Selection-Objekt 188

- Selection-Objekt 186
  - Find *siehe* Find 222
  - Move 188
  - SelectCurrent 188
  - Type 187
  - WdSelectionType 188
- Shape-Objekt 309
  - AddOLEObject 591
  - AddPicture-Method 310
  - AddTextEffect-Method 611
  - ConvertToInlineShape-Method 316
  - ConvertToShape-Method 316
  - Left 322
  - LinkFormat 316
  - LockAnchor 322
  - Nodes.Points 335
  - OLEFormat 591
  - OLEFormat.DoVerb-Method 591
  - RelativeHorizontalPosition 322
  - RelativeVerticalPosition 322
  - TextboxTightWrap 329
  - TextEffect 614
  - TextFrame 337
  - Top 322
  - WrapFormat 328
  - ZOrder-Method 331
  - ZOrderPosition 331
- SmartTag-Objekt 933, 935
  - CustomProperty 934
  - SmartTagAction 934
  - SmartTagRecognizer 934
  - SmartTagType 933, 934
- StoryRanges-Auflistung 234
  - NextStoryRange 245
- StoryRanges-Objekt 244
- Style-Objekt 283
  - Add-Method 292
  - BuiltIn 289
  - Delete-Method 287
  - Hidden 804
  - InUse 286
  - LinkStyle 297
  - Locked 285
  - NameLocal 289
  - Priority 804
  - Type 292
  - UnhideWhenUsed 804
  - Visibility 803
- System-Objekt 174
  - CountryRegion 174
  - Cursor 175
  - LanguageDesignation 175
  - OperatingSystem 175
  - PrivateProfileString 175, 644
  - ProfileString 175
  - Version 175

## T

- Table-Objekt 349
  - BottomPadding 360
  - Cell 351
  - Cell.Border 357
  - Cell.Split-Method 369
  - Cell.VerticalAlignment 360
  - Cells.Merge-Method 369
  - Column.Width 367
  - ConvertToText-Method 371
  - FitText 361
  - ID 370
  - LeftPadding 360
  - NestingLevel 375
  - RightPadding 360
  - Row.AllowBreakAcrossPages 359
  - Row.Cells 351
  - Row.HeadingFormat 359
  - Row.Height 367
  - Row.Index 351
  - Row.LeftIndent 359
  - Rows.Add-Method 351
  - Rows.Alignment 365
  - Rows.HorizontalPosition 365
  - Rows.RelativeHorizontalPosition 365
  - Rows.RelativeVerticalPosition 365
  - Rows.VerticalPosition 365
  - Rows.WrapAroundText 365
  - Shading 357
  - TopPadding 360
  - WordWrap 361
- Template-Objekt 203
  - AutoTextEntries *siehe* AutoTextEntry 204
  - BuildingBlockEntries 274
  - OpenAsDocument-Method 207
  - Type 206
- Templates-Auflistung 205
- ThisDocument-Objekt 623

## U

- UserForm-Objekt 667
  - Activate 673
  - Auflistung 674
  - Caption 669, 682
  - Deactivate 673
  - Font 682
  - Height 669, 682
  - Hide 670
  - Initialize 672
  - layout 673
  - Left 669, 682
  - Load 670
  - Move 671

UserForm-Objekt (*Fortsetzung*)

Name 668  
 QueryClose 674  
 Repaint 671  
 Show 670  
 StartUpPosition 669  
 Tag 669  
 Terminate 672  
 Top 669, 682  
 Unload 670  
 Width 669, 682

## V

VB-Anweisung

#Const 492  
 #If...Then 495  
 Debug.Print 99  
 Do-Schleife 95, 113, 228  
 Enum 82, 122, 510  
 For Each...Next 168  
 For...Next 96, 168  
 GetObjekt 109  
 GetSetting 643  
 If...Then...Else 92  
 Is 216  
 Kill 118  
 On Error Goto 106  
 On Error Resume Next 109  
 Open 118  
 Option Base 75  
 Redim 75  
 Redim Preserve 75  
 Resume 107  
 SaveSetting 643  
 Select Case 84, 94, 108  
 Set 164  
 Type 79

VBComponent-Objekt

Add-Methode 836  
 CodeModule 823, 838  
   InsertLines-Methode 841  
 CodeModule.AddFromFile-Methode 838  
 CodeModule.AddFromString-Methode 838  
 CodeModule.CountOfDeclarationLiens 825  
 CodeModule.CountOfLines 824, 839  
 CodeModule.CreateEventProc-Methode 840  
 CodeModule.DeleteLines 833  
 CodeModule.InsertLines-Methode 839  
 CodeModule.Lines 824  
 CodeModule.ProcBodyLine 827  
 CodeModule.ProcCountLines 827  
 CodeModule.ProcOfLine 826  
 CodeModule.ProcStartLine 827

CodeModule.ReplaceLine 831  
 Count 822  
 Designer *siehe* Designer  
 Name 835  
 Remove-Methode 845  
 Type 822

VB-Datentyp

Boolean 65  
 Integer 66  
 Long 66  
 Object 67, 163  
 Single 66  
 String 65  
 Variant 66

VBE

AddFromString-Methode 838  
 Application-Objekt 820  
 Document-Objekt 820  
 MacroContainer-Eigenschaft 820  
 Property Get 828  
 Property Let 828  
 Property Set 828  
 Property-Prozeduren 828  
 References-Auflistung 849  
 Show-Eigenschaft 846  
 Template-Objekt 820  
 UserForm-Objekt 846  
 UserForms-Auflistung 846

VBE-Objekt 816

ActiveVBProject 819  
 FileName 819  
 Name 818  
 Protection 818  
 VBProject.Import-Methode 835  
 VBProject.References *siehe* Reference-Objekt  
 VBProject.VBComponents *siehe* VBComponent-Objekt  
 VBProjects 818

VB-Ereignis

Auto-Makros 448

VB-Function

OnTime 833

VB-Funktion

CallByName 847  
 CBool 70  
 CInt 70  
 CLng 70  
 CreateObject 489  
 CSng 70  
 CStr 70  
 CVar 70  
 Date 91  
 DateDiff 510  
 Dir 112, 127  
 Environ 145

VB-Funktion (*Fortsetzung*)

FileDateTime 119  
 FileLen 121  
 Format 91, 92  
 GetAttr 115, 117  
 GetPoint 215  
 InputBox 86  
 Instr 90  
 InstrRev 90  
 IsDate 93  
 IsNumeric 92  
 LBound 76  
 Left 90  
 Mid 90  
 MsgBox 87  
 Replace 90, 506  
 Right 90, 114  
 Timer 148  
 UBound 76  
 UCase 93

## VB-Konstantwert

vbAlias 116  
 vbArchive 116  
 vbCR 90  
 vbDirectory 116, 117  
 vbHidden 116  
 vbNormal 116  
 vbObjectError 110  
 vbReadOnly 116  
 vbSystem 116  
 vbVolume 116

## VB-Objekt

Err 110

## VB-Schlüsselwort

ByRef 72, 73  
 ByVal 71, 72  
 Nothing 69, 166  
 Optional 120  
 Private 67, 79, 83  
 Public 68  
 Step 170  
 WithEvents 455

Version *siehe* System-Objekt 175

## View-Objekt

ShowTextBoundaries 250

Visible *siehe* Application-Objekt 178

**W**

wdDontBreakWrappedTable 366

## Wd-Konstantwert

WdBreakType 212  
 WdBuildingBlockTypes 268, 275  
 WdBuiltinStyle 289  
 WdCollapseDirection 210  
 WdCursor 175  
 WdDefaultFilePath 182  
 WdDocPartInsertOptions 276  
 WdDocumentType 191  
 WdFieldKind 379  
 WdHeaderFooterIndex 258  
 WdKey 763  
 WdKeyCategory 763  
 WdMailMergeActiveRecord 429  
 WdMailMergeDestination 435  
 WdMailMergeMainDocType 428  
 WdMergeSubType 442  
 WdPaperTray 255  
 WdPrintOutItem 200  
 WdPrintOutPages 201  
 WdPrintOutRange 198  
 WdRowHeightRule 367  
 WdSelectionType 188  
 WdStatistic 191  
 WdStoryType 246  
 WdTablePosition 366  
 WdTaskPane 784  
 WdTemplateType 206  
 WdUnits 212

wdLine *siehe* Selection-Objekt, Move 188

wdWithinTable 215

WithEvents 455

## WordBasic 183

CreateCommonFieldBlockFromSel 185  
 DisableAutoMacros 180, 184, 450  
 FileNameInfo 185  
 FilePrintSetup 184  
 MailMergePropagateLabel 439  
 SelectSimilarFormatting 184  
 SortArray 184

**X**

XMLNamespace-Objekt 906

XSLTransform-Objekt 906





# Stichwortverzeichnis

## .NET

- aufgezeichneten Code bearbeiten 33
- Interop Assemblies 487
- Konvertierung von Hilfe-Beispielen 64
- Konvertierung von VBA ByVal / ByRef 72
- Konvertierung von VBA User Defined Types 80
- Konvertierung von VBA Variant 67
- Konvertierung von VBA-Ganzzahlen 66
- Konvertierung von VBA-Zeichenketten 65
- Late Binding 495
- Option Strict 495
- optionale Argumente 189
- Umwandlung von VBA-Datentypen 71
- Verweis zur COM-Anwendung hinzufügen 44
- Verweise auf COM-Objekte 487

\\* Mergeformat 316, 380

»Delete Block« Aufforderung 181

## A

### Absatz

- Einzug der ersten Zeile 356
- mit Rahmen versehen 357

### Abschnitt 242

- Eigenschaften 242
- Kopf- und Fußzeile 244

Abschnittsumbruch einfügen 211

Access *siehe* Fernsteuern

### ActiveX-Steuerelement 615

- Datengültigkeit prüfen 620
- Entwurfs-Modus 616
- Größe regeln 617
- im Code ansprechen 623
- Nachteile 616
- navigieren 618

### Add-In (Vorlage)

- darauf verweisen 509
- Funktion aufrufen 509
- Prozedur aufrufen 509
- Prozeduren aus anderer Anwendung aufrufen 512
- UserForm aufrufen 510
- Vorlage (\*.dot) 508

### Add-In (VSTO)

- anlegen 545
- Custom Task Pane 554
- GetCustomUI 549
- Registry-Eintrag 558
- Ribbon-Erweiterung zufügen 546

ADO 582, 586, 601

### Aktualisierung

- von Feldfunktionen 378

Anforderungen an ein Dialogfeld 682

### API 126

- Deklaration und Aufbau 126
- GetKeyState 151
- GetPrivateProfileSection 141
- GetPrivateProfileSectionNames 143
- GetUserNameEx 145
- PathAddBackslash 128
- PathAddExtension 130
- PathAppend 130
- PathCombine 129
- RegCloseKey 135
- RegEnumKeyEx 135
- RegOpenKey 135
- ShellExecute 131
- Sleep 147
- sndPlaySound 148

Application-Objekt *siehe* auch VBE

- Language 290
- Run-Methode 832

Application-Objekt *siehe* VBE

Array *siehe* Datenfeld

### Aufgabenbereich

- Clipart 785
- Dokument schützen 785, 790
- Dokumentaktionen 785, 789
- Dokumentaktualisierung 785, 789
- Dokumentverwaltung 785
- einfache Suchoptionen 786, 789
- erste Schritte 783, 785, 789, 796
- Faxdienst 790
- Faxdienste 785
- Formatinspektor 785
- Formatvorlage übernehmen 786, 791
- Formatvorlagen und Formatierungen 785, 790
- Formatvorlagen und Formatierungen,  
Word 2003 797
- Formatvorlagen, Word 2007 799
- freigegebener Arbeitsbereich 787, 790
- Hilfe 786
- neues Dokument 782, 785, 789, 793
- recherchieren 786–787, 790
- selbstdefiniert 944
- Seriendruck 786, 789

Aufgabenbereich (*Fortsetzung*)

- Signaturen 787, 791
- Startaufgabenbereich 782
- XML-Dokument 787
- XML-Struktur 791, 900
- Zwischenablage 785, 789
- Aufgabenbereich *siehe* Task Pane 780
- XML-Struktur 894

Auflistung

- bearbeiten 168
- ein Element ansprechen 167
- Element entfernen 169
- Index eines Objekts ermitteln 167

Autoform

- Standardwerte festlegen 635

AutoKorrektur

- definieren 633

Auto-Makros 180, 448

- AutoClose 449
- AutoExec 449
- AutoExit 449
- AutoNew 449, 572, 583
- AutoOpen 449
- und Makro-Sicherheit 450
- unterbinden 450
- Vergleich mit Ereignissen 451

AutoText

- entfernen 631

AutoTextEintrag

- einer Kategorie zuweisen 103

## B

Bausteine 267

- Building Blocks.dotx 268
- Building Blocks.dotx, laden erzwingen 270
- BuildingBlockTypes 268
- Dokumenteigenschaften, *siehe* Dokument, Eigenschaften
- Eigenschaften bearbeiten 273
- Felder 267, 272
- im Dokument einfügen 277
- Katalog 268, 275
- Katalog, benutzerdefiniert 278
- Makrorekorder 274
- mit Text und Grafik erstellen 274
- Organizer 267, 270, 279–280
- Schnellbaustein erstellen 273
- Schnellbaustein-Katalog 267
- Schnellbaustein-Menü
  - Dokumenteigenschaften 271
  - Felder 272
  - Organizer 280

Schnellbaustein-Katalog 272

- Schnellbaustein-Katalog, Auswahl speichern im 272

wdTypeQuickParts-Konstante 275

Befehlszeilenargumente 1026

Benutzerdefinierte Dialogfelder

- alle geladenen Objekte bearbeiten 674
- als Argument an Prozedur übergeben 675
- Anforderungen des Anwenders 682
- anzeigen 670
- benennen 668
- Daten in Xml-Datei 689
- effizient erstellen 684
- Eigenschaften in Dateien auslagern 687
- Eigenschaften in Ini-Datei 687
- Eigenschaften in Txt-Datei 688
- Eigenschaften zwischenspeichern 684
- entladen 670
- Ereignisse 672
- erstellen 667
- Fenstertitel eintragen 669
- flackern am Bildschirm verhindern 672
- laden 670
- neu zeichnen 671
- positionieren 669
- schließen verhindern 674
- Startposition festlegen 669
- Steuerelement

Anzeige 677

Befehlsschaltfläche 677

Bezeichnungsfeld 676

Bildlaufleiste 677

Drehfeld 677

Kombinationsfeld 676

Kontrollkästchen 676

Listenfeld 676, 680

Multiseiten 677

Optionsfeld 676, 679

Rahmen 677

Register 677

Textfeld 676

Umschaltfeld 677

Steuerelemente einbinden 676

verbergen 670

verschieben 671

wann tritt ein Ereignis ein 673

zur Laufzeit beeinflussen 684

Benutzernamen ermitteln (API) 145

Benutzerschnittstelle anpassen

Möglichkeiten 656

Speicherort 656

Speicherort (COM-Add-In) 661

Zielgruppe 658

## Bereich

- auf einen Punkt verkleinern 210
- erweitern 212
- festlegen 208
- formatieren 209
- formatierter Text 218
- Formel berechnen 219
- in Tabelle umwandeln 351
- lokalisieren 215
- schattieren 357
- Text hinzufügen 211
- Text zuweisen 210
- und Feldcodes 217
- und verborgener Text 217
- vergleichen 216
- verkleinern 212
- verschieben 212

## Bildschirm

- Flattern vom 178
- unsichtbar machen 178

## Bitweiser Vergleich 154

## BOM 864

## Browserobjekt 239

Building Blocks, *siehe* Bausteine 268

## Building Blocks.dotx

- Neu erstellen 271

Byte Ordering Mark *siehe* BOM**C**

## Codezeilen umbrechen, lange 90

## Codierung 864

## COM-Add-In

- entfernen 777
- installieren 776
- mit Tastenkombination verbinden 773

## Compiler-Anweisung 495

## Content Controls Toolkit 409

**D**

## Datei über Dateieindung ausführen (API) 131

## Dateisystem

- alle Dateien auflisten 112
- Datei löschen 118
- Dateidatum ermitteln 119
- Dateigröße ermitteln 121
- ist Datei gesperrt 117
- ist Datei vorhanden 115
- ist Verzeichnis vorhanden 116
- Ordnername mit Backslash ergänzen 113
- Platzhalter 113

Datenbanken *siehe* Fernsteuern

## Datenfeld

- dimensionieren 75
- Größe ermitteln 76
- sortieren 77

## Datentyp

- Aufzählung 82
- benutzerdefinierter Typ 79
- umwandeln 70

## Debuggen 98

- Debug.Print 99

## Deklarationsbereich 825

## Dialogfelder

- anzeigbare Dialogfelder 696
- anzeigen 691
- Argumente der internen Dialogfelder 693
- Argumente für integrierte Dialogfelder 693
- ausführen 691
- AutoText 270
- AutoText (Word 97–2003) vs. Organizer (Word 2007) 270
- benannte Konstanten 690
- direkt ausführen 692
- Eigenschaft 713
- Einstellungen aktualisieren 692
- Feldfunktionen 272
- Index 695
- integrierte 690
- interne 690
- Konstanten 695
- nur ausführbare Dialogfelder 696
- Organizer für Bausteine 280
- Rückgabewerte 691
- Übersicht der Wordbasic-Anweisungen 693

## digitales Zertifikat 997

Document Object Model *siehe* XML, DOM

## Dokument

- als Textdatei speichern 194
- Ansicht festlegen 631
- ausdrucken 196
- AutoWiederherstellen-Info 1030
- Eigenschaft bearbeiten 650
- Eigenschaften 267, 271
- Eigenschaften, Bausteine 267
- Eigenschaften, Word2007-Fenster einblenden 272
- erstellen 189
- in Vorlage umwandeln 191
- mit einer Vorlage verbinden 192
- mit Kennwort speichern 194
- öffnen 189
- schützen 395
- speichern 194
- Variable bearbeiten 653

## Dokumentbeschädigung 37, 243

## Dokumenteigenschaft bearbeiten 652

## Dokumenteigenschaft eintragen 630

Dokumenteigenschaft *siehe* Dokument  
 Dokumenteinstellungen  
     abspeichern 649  
 Dokumentvariable *siehe* Dokument  
 Dokumentvorlage 203  
 DOM 415  
 Druckerschacht 254  
 Dsofile.dll 652

## E

Early Binding *siehe auch* Verweis 595, 623  
 Eigenschaftenprozeduren 828  
 Eingabeaufforderung  
     InputBox 86  
 Elemente einer Auflistung wahlweise löschen 97  
 Ereignisse 448  
     auf Applikationsebene 453  
         anlegen 455  
         deklarieren 455  
     auf Dokumentebene 450  
     DocumentBeforeClose 467  
     DocumentBeforePrint 469  
     DocumentBeforeSave 468  
     DocumentBeforeSync 471  
     DocumentChange 466, 551  
     DocumentOpen 466  
     Gültigkeitsbereich 453  
     Klassenmodul einrichten 455  
     Klassenmodul initialisieren 457  
     MailMergeDataSourceLoad 551  
     MailMergeWizardSendToCustom 434  
     MailMergeWizardStateChange 434  
     NewDocument 465  
     Reihenfolge der Auslösung 451  
     Speicherort 453  
     WindowActivate 458  
     WindowBeforeDoubleClick 460  
     WindowBeforeRightClick 461  
     WindowDeactivate 459  
     WindowSelectionChange 463  
     WindowSize 459  
 Excel  
     Tabelle in Word verknüpfen 385  
 Excel *siehe auch* Fernsteuern  
     Diagramm im Word-Dokument 600  
     Tabellenobjekt im Word-Dokument 593  
     Tabellenobjekt, Größe festlegen 594  
     Tabellenobjekt, schließen 597  
 Extensible Markup Language *siehe* XML  
 Extensible Stylesheet Language Transformation  
 Externe Daten einfügen 386  
 Externe Daten verknüpfen 385

## F

Facets *siehe* XML, Facetten  
 Fehlerbehandlung 105, 194  
 Fehlermeldungen  
     für Task Panes 783  
 Feldfunktion  
     \* MergeFormat-Schalter 317, 380  
     aktualisieren 378  
     als Baustein 272  
     Ausdruck 386  
     Bausteine 267  
     CreateDate 257, 583  
     Database 580  
     DocProperty 257  
     DocVariable 257  
     Feldcode bearbeiten 381  
     Feldcode in Text umwandeln 390  
     FileName 257  
     If 257  
     in einem Textfeld 381  
     IncludePicture 316  
     IncludeText 992  
     ist Markierung in einer 383  
     Link 385  
     MacroButton 587  
     Macrobutton 384, 425  
     NumPages 257  
     Page 257  
     PrintDate 257  
     Ref 412  
     SaveDate 257  
     Section 257  
     SectionPages 257  
     StyleRef 257  
 Feldfunktionen  
     Hilfe 378  
     verschachteln 387  
 Fernsteuern  
     Access 578  
         ADO Verknüpfung 582  
         Bericht ausdrucken 579  
         Datenbank lesen 580  
         Formular anzeigen 578  
         Tabelle in Word verknüpfen 580  
     Datenbanken 580  
     Excel 560, 586  
         ADO-Verbindung 587  
         Arbeitsmappe drucken 561  
         Tabelle als Datenbank ansprechen 586  
     Excel für komplexe Berechnungen verwenden 562  
     Excel-Daten an Word übermitteln 504  
     Outlook 570  
         Dokument versenden 576  
         Kalendereintrag drucken 570

Fernsteuern (*Fortsetzung*)

- Kontakte als Empfängeradresse verwenden 572
- MailItem-Objekt 576
- Sicherheitswarnung 576
- PowerPoint 563
  - Präsentation drucken 563
  - Struktur der Präsentation übernehmen 566
- Visio 568
  - Zeichnung drucken 568
- Word 506
  - Word Dokument drucken 506
- Fernsteuern (.NET)
  - Late Binding 514
  - Word fernsteuern damit 513
  - Word-Anwendung freigeben 517
  - Word-Instanz starten 514
- Fett formatieren 209
- FileDialog-Dialogfeld 696
  - anzeigen und ausführen 698
  - Datei speichern 702
  - Dateien öffnen 701
  - Dateien wählen 704
  - Filter definieren 699
  - Filter löschen und erstellen 700
  - Mehrfachauswahl 701
  - Ordner wählen 704
  - Rückgabewerte 698
- Formatierung 282
- Formatvorlage
  - anwenden 293
  - Auswahl in der Benutzeroberfläche 283
  - definieren 293, 632
  - Einschränkungen 284
  - Kategorie ändern 98
  - neue erstellen 292
  - Priorität 801
  - Sichtbarkeit 802
  - Sortierreihenfolge 801
  - verknüpft 296, 799
  - Vorteile 282
  - Word-interne 289
- Formel im Bereich berechnen 219
- Formular 391
  - ActiveX-Steuerelement 615
  - Enter-Taste unterbinden 773
  - Feldresultate lesen und schreiben 393
  - Formularfeld Ereignisse 397
  - Gültigkeitsprüfung 397
  - Inhaltssteuerelemente 400
  - Optionenfelder 397
  - schützen 395
  - Steuerelement einbinden 486
  - Zugriff auf unsichere ActiveX-Controls (Steuerelemente) 487

**G**

- GetSetting 643
- Gitternetz *siehe* Zeichnungselemente
- Grafiken 309
  - Anker 321
  - AutoFormen
    - mit VBA einfügen und bearbeiten 335
  - Beschriften 319
  - einfügen 310
  - in Positionsrahmen 319
  - mit dem Text verschieben 322
  - positionieren 320
  - Reihenfolge 331
  - Textflussformatierung 327, 329
  - Verknüpfung ändern 316
  - Verknüpfung auflösen 318
  - Verknüpfung erstellen 315
- GUID erstellen 968
- GUID für Office 850

**H**

- Hilfe 41
  - \*.chm-Dateien 44
  - Objektkatalog 38
  - Word 2007 42
- HTML 864

**I**

- Information *siehe* Range-Objekt 215
- Inhaltssteuerelemente 399
  - benutzerdefinierter XML-Teil 409
  - Content Controls Toolkit 409
  - Daten verbinden 408
  - Eigenschaften 401
  - Ereignisse 420
  - gruppieren 404
  - gruppieren (programmtechnisch) 419
  - Inhalt eingeben (programmtechnisch) 416
  - miteinander verbinden 412
  - Platzhaltertext 401
  - Platzhaltertext formatieren 401
  - programmtechnisch identifizieren 417
  - schützen 404–405
  - strukturiertes Bearbeiten 404
  - XML 407
- INI-Dateien 140
  - als Zwischenspeicher 501
  - Werte lesen und schreiben 177
- Interoperabilität
  - Formel-Editor 610
  - MS Graph 606
  - WordArt 610

## K

Kompatibilitäts-Optionen 639  
 Konstante 77  
 Kontextmenü  
     ansprechen 715  
     erstellen 719  
 Konvertierfunktionen  
     Zentimeter nach Points 324  
 Kopf- und Fußzeile 253, 256  
 Kursiv formatieren 209

## L

Lange Dokumente 262  
 Late Binding 593, 623  
     .NET 514  
     *siehe auch* Verweis

## M

MacroContainer 820  
 Makro  
     der QAT zuweisen 47  
     einer Symbolleiste zuweisen 48  
     einer Tastenkombination zuweisen 49  
     erscheint nicht in der Liste der  
         Benutzerschnittstelle 51  
     kopieren 51  
     löschen 51  
     organisieren 51  
     Reihenfolge bei Namensgleichheit 808  
     speichern 36  
     Speicherort 29, 656  
     synchron abarbeiten 500  
     verschieben 51  
     Zielgruppe 658  
     zur Laufzeit nachladen 503  
 Makroausführung unterbinden 179  
 Makrorekorder 29  
     anhalten 30  
     Arbeitsweise 31  
     Ergebnis betrachten 31  
     starten 29  
 Makrosicherheit  
     Projekt signieren 58  
     Sicherheitsstufe anpassen 52  
     Sicherheitswarnung Outlook 576  
     Signatur erstellen 57  
     umgehen 179  
     Zertifikat einlesen 60  
     Zertifikat erstellen 59  
 Mathematische Funktionen 562  
 Menüleiste *siehe* Symbolleiste

Menüs immer vollständig anzeigen 713  
 Mitteilungen an den Benutzer  
     MsgBox 87  
 MS Graph im Word-Dokument 606  
 MsForms *siehe* Benutzerdefinierte Dialogfelder  
 MSXML-Parser *siehe* XML, Parser

## N

Namenskonflikt 449  
 Namenskonventionen 1002  
 Namespaces *siehe* XML, Namensräume  
 Neue Zeile in einer Zeichenkette 90  
 Nodes *siehe* Knotenpunkte  
 Normal.dot 659  
     defekt 1030  
     konfigurieren 628

## O

Objekte  
     als Variablen verwenden 163  
     eingebettete 590  
     freigeben 166  
     im Objektmodell aufspüren 162  
     Standardeigenschaft 166  
 Objektkatalog 38  
 Office  
     DOM 415  
 OLE 590  
     Dateigröße 600  
     In-place-Aktivierung 590, 600  
     Objekt aktivieren 591  
     Objekt deaktivieren 592  
     Objekt einfügen 590  
     OLE-Client 590  
     OLE-Server 590  
 Option Explicit-Anweisung 837  
 Optionen  
     Registerkarte Bearbeiten 636  
     Registerkarte Kompatibilität 639  
     Registerkarte Rechtschreibung und Grammatik 640  
     Registerkarte Sicherheit 638  
     Registerkarte Speichern 637  
 Optionen (Options Objekt) 180  
 Outlook *siehe* Fernsteuern

## P

Pfade  
     Backslash anhängen (API) 128  
     Datei an Pfad anhängen (API) 130  
     Kombinieren zweier Pfade (API) 127

Pfade (*Fortsetzung*)  
 um Dateierweiterung ergänzen (API) 130  
 zwei Pfade kombinieren (API) 129  
 PI (processing instruction) *siehe* Verarbeitungsanweisung  
 Platzhalter *siehe* Dateisystem  
 Positionsrahmen  
 Anker 321  
 einfügen 319  
 mit dem Text verschieben 322  
 positionieren 320  
 PowerPoint *siehe* Fernsteuern  
 PrivateProfileString *siehe* System-Objekt  
 Programmbibliothek *siehe* Add-In

## Q

QAT *siehe* Ribbon, QAT  
 Quick Access Toolbar *siehe* Ribbon, QAT

## R

Registry  
 Data Schlüssel 780  
 Editor starten 176  
 Sicherung 135  
 Task Pane Einträge 782  
 Werte lesen und schreiben 175  
 Zugriff auf die Registry (API) 135  
 Ribbon 724, 739  
 Anwendungs-Schaltflächen 738  
 bei Null anfangen 741  
 CommandBar-Methoden für 726  
 Control 726  
 Control Group 726  
 Custom UI Editor 728, 742  
 dynamisch aktualisieren 750  
 erweitern 727  
 Erweiterungen teilen 733  
 Fehler beim Laden anzeigen 730  
 Grafikdateien ins Dokument einbinden 742  
 Group 726  
 in Dokument einbinden 730  
 Intellisense bei der XML-Eingabe 553  
 InvalidateControl-Methode 751  
 Invalidate-Methode 749  
 QAT 726, 741  
 Ressourcen im Internet 760  
 Tab 726  
 Tab anwählen 740  
 Terminologie 725  
 Word-Befehle übersteuern 739

Ribbon-Attribut  
 boxStyle 748  
 columns 755  
 enabled 740  
 getEnabled 740  
 getItemCount 755  
 getItemHeight 755  
 getItemId 755  
 getItemImage 755  
 getItemLabel 755  
 getItemScreenTip 755  
 getItemSupertip 755  
 getItemWidth 755  
 getPressed 749  
 getSelectedItemId 755  
 getSelectedItemIndex 755  
 getText 752  
 getVisible 749  
 idMso 738  
 idQ 733  
 image 744  
 imageMso 744  
 insertBeforeMso 740  
 itemWidth 755  
 keytip 741  
 label 744  
 loadImage 552, 758  
 maxLength 752  
 onAction 745  
 onChange 752  
 onLoad 750  
 rows 755  
 screenTip 744  
 size 744  
 sizeString 752  
 startFromScratch 741  
 supertip 745  
 tag 744, 746

Ribbon-Steuerelement  
 box 748  
 button 744  
 buttonGroup 748  
 checkBox 750  
 comboBox 753  
 dialogBoxLauncher 745  
 dropDown 755, 759  
 dynamicMenu 759  
 editBox 752  
 gallery 755  
 item 753  
 labelControl 754  
 splitButton 745  
 toggleButton 749

## S

- SaveSetting 643
- Schaltfläche für Office
  - Befehle außer Kraft setzen 740
- Schemabibliothek 869, 896
  - Schema laden 897
- Schnellbausteine, *siehe* Bausteine 267
- Schriftfarbe 356
- Schritt-für-Schritt
  - AutoKorrektur-Eintrag erstellen 634
  - AutoText entfernen 631
  - benutzerdefinierte Symbolleisten in die QAT 708
  - benutzerdefiniertes Dialogfeld erstellen 684
  - digitale Signatur erstellen 57
  - digitale Signatur zuweisen 58
  - Early und Late Binding 493
  - eine Transformation mit einem Schema verbinden 905
  - Inhaltssteuerelemente gruppieren und schützen 405
  - interne Word-Befehle auflisten 810
  - Makro einer Symbolleiste zuweisen 48
  - Makro einer Symbolschaltfläche zuweisen (QAT) 47
  - Makro einer Tastenkombination zuweisen 49
  - nachträglich ein geöffnetes Dokument transformieren 902
  - Normal.dot automatisch anlegen 629
  - Ribbon-Erweiterung einem VSTO-Projekt zufügen 546
  - Schema in die Schemabibliothek laden 897
  - Tabellenformatvorlage erstellen 361
  - Textdrucker installieren 201
  - Userform erstellen 684
  - XML-Datei beim Speichern transformieren 901
  - XML-Datei erstellen 864
  - Zertifikat einlesen 60
  - Zertifikat exportieren 59
- Seitenlayout 248
  - Bundsteg 251
  - festlegen 630
  - Ränder 250
- Seitenumbruch einfügen 211
- Seitenzahlen 260
- SelfCert.exe 57, 998
- Seriendruck 427
  - Datenquelle einbinden 441
  - Datensatz suchen 432
  - Datensätze ausschließen 429
  - Datensätze navigieren 428
  - Datenverbindungen
    - DDE, Access 445
    - DDE, Excel 446
    - Dokumente 443
    - Konvertierfilter, Textdateien 444
    - ODBC, Access 445
    - ODBC, Excel 446
    - OLE DB, Access 445
    - OLE DB, Excel 446
    - Text 444
  - Etiketten 438
  - Hauptdokument 428
  - Hauptdokument öffnen 433
  - Seriendruck-Assistent 434
  - Umschläge 436
  - zusammenführen 435
- Sicherheit *siehe* Makrosicherheit
- Signatur 56
- Silbentrennung
  - konfigurieren 633
- Smart Document 944
  - ControlCaptionFromID 979
  - ControlCount 978
  - ControlID 978
  - ControlIndex 978
  - ControlNameFromID 979
  - ControlTypeFromID 980
  - DLL erstellen 970
  - DLL-Aktionen 976
  - DLL-Lösung 970
  - DLL-Manifest 995
  - Ereignisse des Interface 982
  - Erweiterungspaket anfügen 949
  - Erweiterungspaket hinzufügen 948
  - InvokeControl 987
  - ISmartDocument-Interface 976
  - Lösung entfernen 953
  - Manifest 966
  - Manifest digital signieren 997
  - mit Erweiterungspaket verbinden 958
  - MOSTL 946
  - MOSTL aktualisieren 949
  - MOSTL installieren 947
  - MOSTL und Unicode 952, 961
  - MOSTL und XSLT 952, 959
  - MOSTL, Dokumentaktionen 950
  - MOSTL-Aktionen 962
  - MOSTL-Komponente über Erweiterungspaket installieren 953
  - MOSTL-Lösung-Datei 961
  - OnListOrComboSelectionChange 986
  - OnPaneUpdateComplete 983
  - PopulateHelpContent 983
  - PopulateListOrComboContent 984
  - PopulateRadioGroup 986
  - SDK 946
  - SmartDocInitialize 982
  - SmartDocXmlCaption 977
  - SmartDocXMLTypeCount 976
  - SmartDocXMLTypeName 977
  - speichern 957



Smart Document (*Fortsetzung*)  
 Speicherort der Komponenten 954  
 Steuerelemente ansprechen 988  
 Type Name 961  
 XMLTypeID 977

Smarttag 910  
 Aktionen 912  
 Aktionen festlegen (MOSTL) 922  
 Ausnahmeliste 913  
 automatische Aktualisierung 916, 919  
 COM DLL 914, 923  
 COM DLL installieren 930  
 eigenes entwickeln 914  
 Erkennungsausdrucke festlegen (MOSTL) 921  
 Grundtypen 910  
 in der Benutzeroberfläche 910  
 in VB6 entwickeln 923  
 ISmartTagAction-Interface 928  
 ISmartTagRecognizer-Interface 927  
 MOSTL 914–915  
 MOSTL lokalisieren 917  
 MOSTL Speicherorte 917  
 Office Smart Tag SDK 914  
 Optionen 913  
 Property Bag 912  
 Recognizer 912  
 Untermenüs 915  
 Word-Objektmodell 933

Speichern  
 AutoWiederherstellen-Info 1030

Sprache der Word-Umgebung 180

Startoptionen 1026

StartUp-Ordner 182, 203, 509, 817

Statistische Angaben eines Dokuments 191

Steuerelemente 842

Suchen und Ersetzen 221  
 im ganzen Dokument 234  
 in einer Schleife ausführen 226  
 Unterschiede zur Benutzerschnittstelle 223

Symbolleiste 708  
 adaptive Menüs 713  
 alle Steuerelemente anzeigen 713  
 Anpassungen unterbinden 712  
 bestimmte Steuerelemente immer anzeigen 713  
 erstellen 719  
 in Word 2007 726  
 Kontextmenü ansprechen 715  
 Kontextmenü erstellen 719  
 positionieren 709  
 schützen 712  
 Steuerelement-Toolbox 616  
 Symbolschaltfläche 713  
 verfügbaren auflisten 709

Symbolleiste für den Schnellzugriff *siehe* Ribbon, QAT

Symbolschaltfläche 713  
 ausführen 713  
 Bearbeitungsfeld 717  
 betätigte abfangen 716  
 Dropdownliste 717  
 identifizieren 713  
 mit Makro verbinden 715  
 mit Word-Befehl belegen 715  
 Popup-Menüs 714  
 Sichtbarkeit 713

## T

Tabelle  
 aus Access einbinden 580  
 Automatisierung von 349  
 Daten lesen 370  
 Einzug 359  
 Excel-Tabellenobjekt 593  
 formatieren 356  
 Höhe 368  
 im Dokument finden 370  
 Kopfzeile wiederholen 359  
 letztes Zeichen einer Tabellenzelle 357  
 mit Textfluss auf einer Seite behalten 366  
 positionieren 365  
 schattieren 357  
 senkrechte Ausrichtung vom Text in einer Zelle 360  
 Spalte formatieren 358  
 Spaltenbreite 367  
 Textanpassen 361  
 verschachtelte 374  
 Zeile formatieren 358  
 Zeilenhöhe 367  
 Zellen teilen 369  
 Zellen verbinden 369  
 Zellenbegrenzung 360  
 Zellenumbruch 361  
 Zellenwechsel 359

Tabellenformatvorlage 361  
 diagonale Rahmenlinien 363  
 Formatierung von Eckzellen 363  
 Kopfzeile wiederholen 364  
 Standard für neue Tabellen 364  
 Streifen 363  
 Tabelleneigenschaften 364

Task Pane 780  
 abfangen oder außer Kraft setzen 789  
 den Inhalt aktualisieren 788  
 eigene erstellen 780  
 einen bestimmten einblenden 788  
 Fehlermeldungen 783  
 Objektmodell 781  
 Registry-Einträge 782

Task Pane (*Fortsetzung*)

- Startaufgabenbereich kontrollieren 782
- Typen 784
- und der Makrorekorder 780
- welcher ist eingeblendet 787
- Work Pane 780

Tastenkombination

- alle auflisten 767
- Alt+F11 31, 36
- Alt+F9 317, 380, 385
- aussperren 772
- entfernen 771
- F1 44
- F2 38
- F8 99
- F9 102
- für bestimmten Befehl ermitteln 770
- programmtechnisch anlegen 772
- programmtechnisch ermitteln 765
- programmtechnische Definition 763
- Strg+Umschalt+F7 267
- Strg+Bild ab 239
- Strg+Bild auf 239
- Strg+F9 103, 379
- Strg+G 100
- Strg+S 36
- Strg+Umschalt+F9 103
- Umsch+Alt+- 780
- Umschalt+F9 101

Tastenstatus ermitteln (API) 151

Tastenwerte

- ASCII-Werte 152
- virtuelle Tastenkonstanten 152

Textmarke 340

- Benennung von 341
- Daten schreiben 344
- einfügen 341
- Inhalt lesen 346
- verborgene 340
- vordefinierte 346

ThisDocument-Modul 451

ThisDocument-Objekt 834

TrueType-Schrift

- Eigenschaften 638
- einbetten 637

## U

Umgebungsvariablen

- Environ-Funktion 145

Umwandlungsfunktionen *siehe* Datentyp

Uniform Resource Identifier *siehe* URI

URI 879

Userform *siehe* Benutzerdefinierte Dialogfelder

UTF-8 *siehe* Codierung

## V

Variable 64

- Auflistung *siehe* Document-Objekt
- deklarieren 65, 69
- Gültigkeit 67
- Sichtbarkeit 67
- Werte nachprüfen 101

VBA-Code 814

VBA-Projekte 817

VBE

- Makro ausführen 832
- Verweise neu setzen 854

VB-Editor 36, 814

- Code speichern 36
- Code-Fenster 815
- Debuggen 98
- Eigenschaftenfenster 677, 816
- Haltepunkt festlegen 102
- Haltepunkte, alle entfernen 103
- Hilfe zum 36
- IntelliSense 37
- lange Codezeilen umbrechen 90
- Microsoft Visual Basic for Applications
- Extensibility 5.3 815
- nächste Anweisung festlegen 103
- Option 'Zugriff auf Visual Basic-Projekt vertrauen' 814
- programmtechnisch steuern 816
- Projekt-Explorer 815
- Sicherheitskopien erstellen 36, 668
- Überwachungsbereich 101
- UserForm-Fenster 815
- Werkzeugsammlung 677

VB-Editorr

- Sicherheitsstufe anpassen 814

Verarbeitung unterbrechen (API) 147

Verknüpfungen 385

- relative Pfadangaben 317

Verweis 482

- auf Add-In setzen 509
- Early Binding 489, 492, 560
- Late Binding 489, 492, 560
- manuell einfügen 483
- unterschiedliche Versionen 491

Visio *siehe* Fernsteuern

VSTO 519

- Aufgabenbereich Dokumentaktionen 527, 530
- Aufgabenbereich mit Benutzersteuerelement 530
- Benutzerschnittstelle bereitstellen 527
- Code hinter dem Dokument 523
- COM-Add-Ins *Siehe* Add-In (VSTO)
- Datencache 535
- Daten-Zwischenspeicher 535
- Dokument mit anderer Vorlage verbinden 539

VSTO (*Fortsetzung*)

- Dokumentlösung und Word 2007 540
- Dokumentlösungen 519
- InnerObject 532
- Lösung anlegen 520
- Lösung vom Dokument trennen 539
- Nachschlagswerke 543
- Nachteile 519
- RemoveCustomization 539
- ServerDocument-Funktionalität 535, 539
- Symbolleiste anlegen 528
- Textmarken-Steuerelement 524
- Textmarken-Steuerelement-Ereignisse 524
  - Deselect-Ereignis 526
  - Select-Ereignis 526
- ThisApplication-Objekt 532
- Vorteile 519

**W**

- Wave-Dateien abspielen (API) 148
- wdDialogs-Konstante *siehe* Dialogfelder-Konstanten
- Wd-Konstantwert
  - WdFieldType 1007
- Windows-Registrierung
  - Data-Key löschen 1030
  - Werte einlesen 643
  - Werte speichern 643
- Winword.exe
  - Startoptionen 1026
- Word
  - siehe auch* Fernsteuern
  - Startoptionen 1026
- Word XML toolbox 899
- WordArt 610
- WordBasic 183
- Word-Befehle
  - übersteuern 739, 808
  - unabhängig von Programmsprache 811
- Word-Meldungen unterbinden 179
- WordML *siehe* WordProcessingML
- WordProcessingML 868, 890
- WordProcessingML XSLT Inference Tool 906
- Work Pane *siehe* Task Pane 780

**X**

- XHTML 864, 868
- XML 862
  - Attribut 871
    - aus Access exportieren 958
  - benutzerdefiniertes Vokabular in Word 894
  - CDATA 873
  - Datei erstellen 864
  - Deklarationen 872
  - Dokument beim Öffnen transformieren 901, 937
  - Dokument beim Speichern transformieren 901
  - Dokumentinstanz 873
  - DOM 648, 870, 884, 938
  - Element 870
  - Facetten 876
  - gültiges Dokument 874
  - Inhaltssteuerelemente und 407
  - Knotenpunkte 885–886
  - Markup 873
  - Namensräume 422, 870, 878
  - Namensraumpräfix 423, 882–883, 896
  - Namensraumpräfix programmtechnisch
    - ermitteln 424
  - Nutzen in Word 868
  - Parser 870, 907
  - Schema 875, 878
  - Schema in Schemabibliothek laden 897
  - Schema in Word einem XML-Dokument
    - zuweisen 898
  - Schema programmtechnisch der Schemabibliothek
    - hinzufügen 907
  - Schemaverletzungen einblenden 957
  - Solutions in Word 900
  - Tag 865, 870
  - Transformation mit dem DOM 907
  - Transform Inference Tool 959
  - Transformation automatisch erstellen 906
  - Transformation in Word 900
  - Transformation mit dem DOM 937
  - Transformation mit Schema verbinden 904
  - transformieren 866
  - über IncludeText einfügen 992
  - Verarbeitungsanweisung 873, 892
  - Vokabular 864
  - wohlgeformtes Dokument 874
  - Wurzelelement 873, 881
  - XPath 870
- XML-Erweiterungspaket
  - Sicherheitsmeldung 949
- XMLSign.exe 998
- XSD *siehe* XML, Schema
- XSLT 866, 870, 885, 938
  - Werte aus Attributen lesen 994

**Z**

- Zeichencodierung *siehe* Codierung
- Zeichenkette
  - verknüpfen 71
- Zeichenkettenbearbeitung
  - Format 91
  - Instr 90
  - InstrRev 90
  - Left 90
  - Mid 90
  - Replace 90
  - Right 90
- Zeichnungselemente optimieren 634
- Zeilenweise verschieben 188
- Zeitungsspalten 252
- Zentral-/Filialdokumente 263
- Zertifikat 56
- Zoomfaktor
  - festlegen 631
- Zugriff auf die Registry (API) 135
- Zwischenablage
  - Inhalte einfügen 313
  - Office 785