

Cindy Meister, Christian Freßdorf, Thomas Gahler, Peter Jamieson

Microsoft Word-Programmierung – Das Handbuch

Cindy Meister, Christian Freßdorf, Thomas Gahler, Peter Jamieson: Microsoft Word-
Programmierung – Das Handbuch
Copyright © 2010 O'Reilly Verlag GmbH & Co. KG

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
13 12 11 10

ISBN 978-3-86645-458-3

© 2010 O'Reilly Verlag GmbH & Co. KG
Balthasarstr. 81, 50670 Köln
Alle Rechte vorbehalten

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

Cindy Meister, Christian Freßdorf, Thomas Gahler, Peter Jamieson

Microsoft Word- Programmierung – Das Handbuch

Microsoft[®]
Press

Übersicht

Vorwort	17
Teil A	
Grundlagen der Arbeit mit VBA	23
1 Word-Makros	25
2 VBA-Grundlagen	55
3 Windows-APIs in VBA nutzen	117
Teil B	
Das Objektmodell von Word	157
4 Überblick über die Arbeit mit Objekten	159
5 Grundlagen des Objektmodells	171
6 Professionelle Dokumente	241
7 Word und Datenstrukturen	341
8 Ereignisse in Word	453
Teil C	
Steuern und gesteuert werden	493
9 Grundlagen der Fernsteuerung	495
10 Word von anderen Umgebungen aus steuern	513
11 Andere Programme von Word aus steuern	573
12 Eingebettete Objekte	605
Teil D	
Optimierung der Benutzerschnittstelle	647
13 Anwendungsoptionen	649
14 Speicherort der Anpassungen	675
15 Mit Dialogfeldern arbeiten	683
16 Die Office Fluent UI	727
17 Tastaturbelegungen	779
18 Word-Aufgabenbereiche	793
19 Interne Word-Befehle übersteuern	811
20 Zugriff auf den Visual Basic-Editor (VBE)	819

Teil E**XML und Smart-Technologien** 867

21	Word und XML: Eine Einführung	869
22	Word Open XML-Dateiformat	915

Teil F**Anhang** 969

A	Namenskonventionen für Word-VBA	971
B	Allgemeines zum Thema Feldfunktionen	975
C	Startoptionen von Word	983
D	Bei Problemen – Word zurücksetzen	987
E	Nützliche Links ins Internet	991
F	Begleitdateien	993
	Verzeichnis zum Objektmodell	995
	Stichwortverzeichnis	1003
	Über die Autoren	1015

Inhaltsverzeichnis

Vorwort	17
Für wen wurde das Buch geschrieben?	18
Word-Versionen	19
Wie ist das Buch aufgebaut?	20
Stichwortverzeichnisse	21
Die Begleitdateien zum Buch	21

Teil A

Grundlagen der Arbeit mit VBA	23
--	-----------

1 Word-Makros	25
Aller Anfang ist schwer	26
Programmierhilfen	27
Den Makrorekorder einsetzen	27
Unterstützende Werkzeuge des VB-Editors	34
Die Objektmodell-Hilfe – eine verborgene Schatzkammer	38
Makros in die Benutzerschnittstelle einbinden und verwalten	41
Makro einem Symbol zuweisen	41
Makro einer Tastaturkombination zuweisen	42
Makros kopieren	44
Makrosicherheit	45
Sicherheitsstufe anpassen	46
Eigene Signatur mittels <i>selfcert.exe</i> erstellen	50
VBA-Projekte mit einer Signatur versehen	51
Zertifikat zur Signatur erstellen	52
Zertifikat einlesen	53
Zusammenfassung	54
2 VBA-Grundlagen	55
Variablen	56
Standarddatentypen	57
Gültigkeit bzw. Sichtbarkeit	59
Umwandlung von Datentypen	62
Weiterreichen von Variablen an Prozeduren	63
Variablen in Datenfeldern ablegen	66
Konstanten	69
Benutzerdefinierte Typen	71
Type-Anweisung	71
Enum-Anweisung	75

Nützliche VBA-Funktionen	79
Bedingungen	85
Schleifen	88
Compileranweisungen	91
Code im VB-Editor debuggen	94
Der Überwachungsbereich	97
Fehlerbehandlung	99
Dateisystem-Operationen	105
Alle Dateien eines Verzeichnisses auflisten	106
Verzeichnisname mit Backslash ergänzen	107
Prüfen, ob eine bestimmte Datei vorhanden ist	109
Prüfen, ob ein bestimmter Ordner vorhanden ist	110
Prüfen, ob eine Datei von jemanden im Zugriff ist	111
Datei löschen	112
Letztes Speicherdatum einer Datei ermitteln	113
Größe einer Datei ermitteln	115
Zusammenfassung	116
 3 Windows-APIs in VBA nutzen	 117
Aufbau der API-Funktionen	118
API-Funktionen unter Office-Versionen mit 64 Bit und 32 Bit verwenden	121
Pfade kombinieren und abschließen	123
Pfade abschließen	123
Pfade kombinieren	124
Dateinamen mit Dateierweiterung an Pfad anhängen	125
Datei über die Dateierweiterung ausführen	127
Benutzerformular transparent darstellen	131
Auf Registry-Einträge zugreifen	132
Registry-Einträge und -Werte ermitteln	132
Auf INI-Dateien zugreifen	139
Abschnitte auslesen und schreiben	139
Alle Abschnitte einer INI-Datei ermitteln	142
Name des angemeldeten Benutzers ermitteln	144
Verarbeitung für eine bestimmte Zeit unterbrechen	146
Wave-Datei abspielen	147
Tastenstatus abfragen	150
Zusammenfassung	156
 Teil B	
Das Objektmodell von Word	157
 4 Überblick über die Arbeit mit Objekten	 159
Arbeiten mit Objekten	160
Das gesuchte Objekt aufspüren	160
Objekte als Variablen	161

Auflistung von Objekten	165
Auflistung bearbeiten	166
Zusammenfassung	170
5 Grundlagen des Objektmodells	171
Das <i>System</i> -Objekt	173
Die Anwendung: das <i>Application</i> -Objekt	176
Die gegenwärtige Markierung: <i>Selection</i> und ähnliche Objekte	186
Der Kern der Sache: das <i>Document</i> -Objekt	188
Dokumente speichern	193
Dokumente drucken mit der Methode <i>PrintOut</i>	196
Dokumentvorlagen: das <i>Template</i> -Objekt	202
Mit Bereichen arbeiten: das <i>Range</i> -Objekt	207
Einen Bereich definieren	207
Einen Bereich bearbeiten	208
Wo befindet sich der Bereich?	214
Text aus einem Bereich lesen	216
Eine Formel berechnen	218
Die Nadel im Heuhaufen: <i>Find/Replace</i> einsetzen	220
Vor- und Nachteile des Makrorekordergebnisses	221
Anpassung des aufgezeichneten Codes	225
Bekannte Probleme vermeiden oder beheben	238
Zusammenfassung	240
6 Professionelle Dokumente	241
Abschnitte im Dokument: das <i>Section</i> -Objekt	242
Bereiche im Dokument: das <i>StoryRanges</i> -Objekt	244
Seite definieren: das <i>PageSetup</i> -Objekt	248
Seite gestalten: das <i>HeaderFooter</i> -Objekt	256
Lange Dokumente	262
Zentraldokumente	262
Dokumente verknüpfen	266
Bausteine: das <i>BuildingBlocks</i> -Objekt	266
Schnellbausteine	267
Die zentrale Bausteinvorlage <i>Built-In Building Blocks.dotx</i>	269
Die Dokumenteigenschaften	270
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
Anpassungen der Benutzerschnittstellen	284
Formatierungseinschränkungen	285
Word-interne Formatvorlagen	290
Formatvorlagen erstellen und modifizieren	293

Automatische Nummerierung mit Listen	297
Listeigenschaften ermitteln	298
Listvorlagen (<i>ListTemplates</i>)	300
Grafiken: die <i>InlineShape</i> - und <i>Shape</i> -Objekte	308
Grafiken einfügen	309
Verknüpfungen erstellen und verwalten	316
Layout-Optionen	321
Zeichnungsobjekte (<i>AutoFormen</i>)	335
Zusammenfassung	339
 7 Word und Datenstrukturen	341
Zielscheibe Textmarke: das <i>Bookmark</i> -Objekt	342
Inhalt mit Tabellen strukturiert darstellen	351
Tabellen erstellen	352
Tabellen formatieren	358
Informationen aus Tabellen holen	372
Feldfunktionen	380
Verknüpfungen, Tabellen und Berechnungen	387
Formulare: das <i>FormField</i> -Objekt	393
Die Alternative zu Formularfeldern: das <i>ContentControls</i> -Objekt	402
Die Grundlagen	403
Inhaltssteuerelemente und XML	410
Inhaltssteuerelemente im Objektmodell	415
Der Seriendruck: das <i>MailMerge</i> -Objekt	433
Datenquelle einbinden	446
Einige Beispieldatenverbindungen	449
Zusammenfassung	452
 8 Ereignisse in Word	453
AutoMakros als Pseudoereignisse	454
Ereignisse auf Dokumentenebene	456
Ereignisse auf Applikationsebene	459
Klassenmodule einrichten und initialisieren	461
Klassenmodule einrichten	461
Klassenmodule initialisieren	463
Übersicht über die verfügbaren Ereignisse	464
WindowActivate	464
WindowDeactivate	465
WindowSize	465
WindowBeforeDoubleClick	466
WindowBeforeRightClick	467
WindowSelectionChange	469
NewDocument-Ereignis	471
DocumentOpen	472
DocumentChange	472
DocumentBeforeClose	473
DocumentBeforeSave	474

DocumentBeforePrint	475
DocumentSync	477
Seriendruckereignisse	478
MailMergeBeforeMerge	478
MailMergeBeforeRecordMerge	478
MailMergeAfterRecordMerge	479
MailMergeAfterMerge	479
Beispiel: 1:n-Liste im Seriendruckresultat	479
ProtectedViewWindow-Ereignisse	484
ProtectedViewWindowActivate	486
ProtectedViewWindowOpen	487
ProtectedViewWindowBeforeEdit	488
ProtectedViewWindowDeactivate	489
ProtectedViewWindowBeforeClose	490
ProtectedViewWindowSize	491
Zusammenfassung	492

Teil C

Steuern und gesteuert werden 493

9 Grundlagen der Fernsteuerung	495
Verweise auf externe Bibliotheken im VB-Editor	496
Verweise in Visual Studio 2008	502
Early versus Late Binding	503
Vorteile von Early Binding	504
Vorteile von Late Binding	506
Ganz clever, wer beide Arten kombiniert	507
Late Binding in Visual Studio .NET	510
Zusammenfassung	511
10 Word von anderen Umgebungen aus steuern	513
Fernsteuern von Microsoft Word	514
Makros von außen anstoßen	514
Word effektiv fernsteuern	518
Versteckte Word-Instanz steuern	520
Programmzeilen zentral verwalten und gemeinsam nutzen (Add-Ins)	522
Dokumentvorlagen (.dotm) als Add-In	522
Add-In-Prozeduren in anderen Umgebungen nutzen	526
Fernsteuerung aus Office-fremden Umgebungen	527
Die Word-Anwendung über Visual Studio .NET fernsteuern	527
Eine laufende Instanz ansprechen sowie Word starten	528
COM-Anwendung-Ressourcen freigeben	531
VSTO COM-Add-Ins	533
Das Anlegen eines VSTO-Add-In-Projekts	535
VSTO-Add-In-Benutzerschnittstellen	537

VSTO-Elemente Dokumenten dynamisch zufügen	550
Das Beispiel ausprobieren	550
VSTO-Dokumentlösungen	551
Eine VSTO-Lösung anlegen	552
Der Code »hinter dem Dokument«	554
Das VSTO-Dokument vorbereiten	555
Die Benutzerschnittstellen einer VSTO-Lösung	556
Der Datenaustausch bei geschlossenem Dokument	564
VSTO-Lösung von einem Dokument abtrennen	568
VSTO 2005-Dokumentlösungen in Word 2007	569
Verteilung von VSTO-Lösungen	570
Das Beispiel ausprobieren	570
Zusammenfassung	571
11 Andere Programme von Word aus steuern	573
Microsoft Excel fernsteuern	574
Versteckte Excel-Instanz steuern	575
Berechnungen von Excel erledigen lassen	576
Microsoft PowerPoint fernsteuern	577
Versteckte PowerPoint-Instanz steuern	578
Text der Präsentation als Inhaltsstruktur in ein Dokument übernehmen	580
Microsoft Visio fernsteuern	582
Versteckte Visio-Instanz steuern	582
Microsoft Outlook fernsteuern	584
Versteckte Outlook-Instanz steuern	584
Kontakt als Empfängeradresse im Brief nutzen	586
Die aktuelle Datei über Outlook versenden	591
Microsoft Access fernsteuern	592
Auf Datenbanken zugreifen	594
Access-Datenbank ansprechen	594
Excel-Tabelle ansprechen	601
Zusammenfassung	604
12 Eingebettete Objekte	605
Excel-Tabellenobjekte	609
Office (Excel)-Diagramme	616
Microsoft Graph-Diagramme	623
WordArt	628
SmartArt	633
SmartArt-Grafiken auflisten	633
SmartArt-Grafiken einfügen	635
Auf vorhandene SmartArt-Grafik prüfen	637
Anpassung einer SmartArt-Grafik	637
Zusammenfassung	646

Teil D**Optimierung der Benutzerschnittstelle 647**

13 Anwendungsoptionen	649
Dokumentvorlage <i>Normal.dotm</i> konfigurieren	650
Vorlage »UrNormal.dotm« erstellen	650
Benutzereinstellungen abspeichern	660
<i>SaveSetting</i> -Anweisung	662
<i>GetSetting</i> -Funktion	663
<i>PrivateProfileString</i> -Eigenschaft	663
Informationen mit MSXML schreiben und lesen	667
Dokumenteinstellungen abspeichern	669
Dokumenteigenschaften bearbeiten	670
Dokumentvariablen bearbeiten	673
Zusammenfassung	674
 14 Speicherort der Anpassungen	 675
Der Speicherort einer Anpassung	676
Auswahl des Speicherorts	678
Den Speicherort programmtechnisch festlegen	680
Kontext für COM-Add-Ins	681
Hierarchie der Anpassungen	682
Zusammenfassung	682
 15 Mit Dialogfeldern arbeiten	 683
Benutzerdefinierte Dialogfelder	684
Ein UserForm in verschiedenen Projekten nutzen	685
Das <i>UserForm</i> -Objekt	686
Steuerelemente einbinden	695
Steuerelemente und ihre Besonderheiten	696
Anforderungen an ein Dialogfeld	700
Dialogfelder zur Laufzeit beeinflussen	702
Interne Dialogfelder	708
Dialogfelder <i>anzeigen</i> , <i>anzeigen und ausführen</i> oder <i>ausführen</i>	709
Dialogfelder konfigurieren, vorbelegen und auswerten	712
Übersicht über die in Word enthaltenen Dialogfelder	713
<i>FileDialog</i> -Objekt	715
Übersicht über die verschiedenen <i>FileDialog</i> -Typen	715
Dialogfelder <i>anzeigen</i> oder <i>anzeigen und ausführen</i>	717
Definieren von Dateiauswahlfilter	718
Anwendung des <i>FileDialog</i> -Typs <i>msoFileDialogOpen</i>	720
Anwendung des <i>FileDialog</i> -Typs <i>msoFileDialogSaveAs</i>	721
Anwendung des <i>FileDialog</i> -Typs <i>msoFileDialogFilePicker</i>	723
Anwendung des <i>FileDialog</i> -Typs <i>msoFileDialogFolderPicker</i>	724
Zusammenfassung	726

16 Die Office Fluent UI	727
Was ist das Ribbon?	728
Die neue Terminologie	730
Die Rolle von <i>CommandBars</i>	731
Einführung in die Menüband-Erweiterung	732
Werkzeuge	732
Basiswissen	733
Erweiterte Funktionalität	738
Befehle mit Tastenkombinationen verbinden	738
Eine Registerkarte an eine beliebige Stelle positionieren	738
Word-eigene Schaltflächen benutzen	740
Word-Befehle übersteuern	741
Word-eigene Registerkarten außer Kraft setzen	742
QuickInfo für Menüband-Steuerelemente	742
Grafiken in die Menüband-Erweiterung einbinden	743
Code für den dynamischen Ablauf	747
Menüband-Registerkarten teilen	751
Eine Registerkarte anwählen	752
Dynamische Größenanpassung von Gruppen	754
Die Menüband-Erweiterung bei Null anfangen	755
Die Steuerelemente	755
Schaltfläche	759
dialogBoxLauncher	760
splitButton	760
toggleButton	761
checkBox	761
box	762
buttonGroup	763
editBox	763
comboBox	765
labelControl	766
gallery	766
dropDown	770
dynamicMenu	770
Kontextmenüs definieren	771
Backstage	774
Zusammenfassung	777
17 Tastaturbelegungen	779
Tastenbelegungen im Objektmodell	781
Bestehende Tastenbelegungen ermitteln	783
Tastenbelegungen eines Befehls ermitteln	788
Tastenkombinationen verwalten	789
Tastenbelegungen entfernen	789
Tastenkombinationen definieren	790
Tastenbelegung mit COM-Add-In verbinden	791
Zusammenfassung	792

18	Word-Aufgabenbereiche	793
	Allgemeines zum Aufgabenbereich	794
	Der programmtechnische Umgang mit Aufgabenbereichen (Task Panes)	795
	Einen bestimmten Aufgabenbereich einblenden	795
	Einen Aufgabenbereich positionieren	798
	Fehlermeldungen	800
	Der Aufgabenbereich <i>Formatvorlagen</i>	803
	Zusammenfassung	810
19	Interne Word-Befehle übersteuern	811
	Word-Befehl außer Kraft setzen	812
	Zusammenfassung	817
20	Zugriff auf den Visual Basic-Editor (VBE)	819
	Notwendige Verweise und Sicherheitseinstellungen	820
	Der Visual Basic-Editor	821
	Die VBA-Projekte	823
	Übersicht über alle VBA-Projekte	824
	Das aktive VBA-Projekt	825
	Zugriff auf die in einem Projekt enthaltenen Komponenten	827
	Auslesen des VBA-Codes von Komponenten	829
	Allgemeine Zugriffe auf die Codezeilen	830
	Zugriff auf den Deklarationsbereich	831
	Zugriff auf die Codezeilen einzelner Prozeduren	832
	Auflisten und Durchsuchen aller Projekte	835
	Ersetzen und Entfernen von VBA-Codezeilen	837
	Hinzufügen von Komponenten zu einem Projekt	840
	Eine vorhandene Komponente in ein Projekt importieren	840
	Eine neue Komponente einem Projekt hinzufügen	842
	Entfernen von Komponenten aus einem Projekt	850
	Anzeigen von dynamisch erzeugten UserForms	851
	Eigenschaften von Steuerelementen über das Designer-Objekt dynamisch ändern	854
	Verweise auf Bibliotheken und Dateien ermitteln und zur Laufzeit setzen	856
	Übersicht über die gesetzten Verweise	856
	Neuen Verweis setzen	857
	Ungültige Verweise korrigieren bzw. entfernen	860
	Zusammenfassung	864
	Teil E	
	XML und Smart-Technologien	867
21	Word und XML: Eine Einführung	869
	Was ist XML und wozu dient es?	870
	XML-Vokabular	872
	XML-Daten transformieren	874

Welche Aufgabe erfüllt XML in Word?	876
XML-Bestandteile	877
XML-Parser	878
XML-Elemente, -Tags, -Inhalt und -Attribute	878
Fehlende sowie mehrfach vorkommende Werte	879
Elemente oder Attribute?	880
XML-Dokumente und Deklarationen	880
XML-Dokumentinstanz, Markup und Inhalt	881
Wohlgeformte und gültige XML-Dokumente	882
XML-Schema-Definitionen	883
XML-Namensräume und Schemas	886
XML-Daten manipulieren	893
Das XML-Dokument-Objektmodell (DOM)	893
XSLT	894
XML in Word	898
WordProcessingML	898
Benutzerdefiniertes XML	901
Die Word-Schemabibliothek	903
Transformationen und Lösungen (Solutions)	907
Schemas und Transformationen im Word-Objektmodell	912
WordProcessingML außerhalb von Word generieren	912
Zusammenfassung	914
22 Word Open XML-Dateiformat	915
Die Word-Dateiformate	916
Erstellung eines Open XML-Dokuments	917
Das OPC und dessen Bestandteile	918
Das »flache« OPC-Format	919
Das vollumfängliche OPC-Format	923
Beispiel: Einen <i>customUI</i> -Teil in ein Dokument einbinden	926
Teile, Beziehungen und Inhaltstypen	934
Teile und ihre Typen	935
Mehr zum Thema Beziehungen	939
Quellen und Ziele	939
Implizite und explizite Beziehungen	940
Erlaubte Beziehungstypen und Typen-URIs	941
Beziehungen zu Teilen für Custom XML Parts (Daten)	943
Standards für die Benennungen von Teilen	943
Die programmtechnische Arbeit mit Open XML: eine Übersicht	944
Das XML-Dateiformat	944
Das Word-Objektmodell	944
Das Zusammenspiel der zwei Methoden	945
Zugang zum Open XML durch das Objektmodell	945
Werkzeuge für die direkte Bearbeitung eines Open XML-Pakets	946
Word-VBA und die Arbeit mit Open XML	947
Beispiel: Seriendruck-Datenquelleninformationen abfragen und entfernen	948
Dokumenteigenschaften, Custom XML Parts und Inhaltssteuerelemente	954
Beispiel: Custom XML Parts ersetzen ohne das Word-Objektmodell	959

Open XML-Dokumente & XSLT	965
Überlegungen zur Arbeit mit XSLT	965
Eine letzte Bemerkung	966
Beispiel	966
Zusammenfassung	968
 Teil F	
Anhang	969
 A Namenskonventionen für Word-VBA	971
Variablen	972
Benutzerdefiniertes Dialogfeld	972
Entwicklungsumgebung	973
Word-Objekte	973
 B Allgemeines zum Thema Feldfunktionen	975
 C Startoptionen von Word	983
Die Befehlszeilenargumente von Word	984
 D Bei Problemen – Word zurücksetzen	987
 E Nützliche Links ins Internet	991
 F Begleitdateien	993
Die Beispieldateien	994
Zusatz-Software: Die Dateien im Anhang	994
 Verzeichnis zum Objektmodell	995
 Stichwortverzeichnis	1003
 Über die Autoren	1015

Vorwort

In diesem Vorwort:

Für wen wurde das Buch geschrieben?	18
Word-Versionen	19
Wie ist das Buch aufgebaut?	20

Die Autoren freuen sich, Ihnen die dritte Auflage des Microsoft Press-Handbuchs zur Word-Programmierung vorzulegen. Wie im Vorwort der vorherigen Auflage erwähnt, konzentriert sich die Weiterentwicklung der Office-Anwendungen auf Bedürfnisse von Großfirmen. Office 2010 bietet beispielsweise die Kernanwendungen, Word, Excel und PowerPoint, als Web-Anwendungen an, die unter SharePoint oder als Teil von Office Live laufen.

Ein herber Rückschlag für dieses Konzept war der Gerichtsentscheid vom August 2009, der die von Microsoft eingesetzte Technologie für »Custom XML« verbietet (in den Medien wurde darüber ausführlich berichtet). Für den europäischen Markt bedeutet dies, dass ab Office 2010 die rosa XML-Elemente, (im Kapitel 21 vorgestellt) beim Öffnen eines Office 2007-Dokuments (*.docx* bzw. *.docm*) ohne Warnung entfernt werden. In Word 2003 erstellte *.doc*-Dokumente werden hingegen nicht geändert. Mehr zu diesem Thema steht im Blog vom Gray Knowlton geschrieben (http://blogs.technet.com/b/gray_knowlton/archive/2009/12/23/what-is-custom-xml-and-the-impact-of-the-4i-judgment-on-word.aspx sowie http://blogs.technet.com/b/gray_knowlton/archive/2010/01.aspx). Inhaltssteuerelemente (Kapitel 7) sind von der Entscheidung *nicht* betroffen und funktionieren weiterhin. Sie werden von Microsoft als Ersatz für »Custom XML« vorgeschlagen.

Nichtsdestotrotz gibt es Neues im Bereich der Word-Programmierung, die wir in dieser Auflage diskutieren:

- Eine neue Compiler-Konstante, um zu testen, ob der Code in Office 2010 läuft.
- Eine neue Methode zum *Application*-Objekt, die dem Entwickler erlaubt, mehrere Handlungen in einen Eintrag der »Rückgängig machen«-Liste zu bündeln
- Die neuen Objektmodelle für das Erstellen und Verwalten von SmartArt sowie Diagramme
- Neue Möglichkeiten für die Erweiterung und Anpassung des Menübandes (Multifunktionsleiste bzw. »Ribbon«)
- Die neue »Office-Backstage-Ansicht«, die ebenfalls über Menüband-XML anpassbar ist
- Neue Erkenntnisse zur Handhabung von Inhaltssteuerelementen
- Einführung in das Erstellen von Word-Dokumenten als XML-Dateien

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

Da die Word-Programmierschnittstelle weiterhin auf das in Word 97 eingeführte Objektmodell basiert, gelten die Kernaussagen für alle seither veröffentlichten Versionen. Diese Auflage behandelt aber primär die Ribbon-Versionen von Word 2007 sowie 2010. Alle Abbildungen wurden in Word 2010 unter Windows 7 erstellt. Im Allgemeinen hat das Entwicklerteam auf die Rückwärtskompatibilität geachtet und, falls die Funktionalität in einer älteren Version von Word vorhanden ist, sollen die Codebeispiele auch in Word 97, 2000, 2002 sowie 2003 laufen.



Alle Codebeispiele wurden in Word 2010 getestet, falls notwendig korrigiert, aber weitgehend für gut befunden und belassen. Um gezielt Word 2010-spezifische Informationen zu finden, halten Sie bitte nach dem nebenstehend dargestellten Symbol Ausschau.



Die C#-Beispiele wurden in das Visual Studio 2008-Format konvertiert und funktionieren, wie der VBA-Code, weiterhin. Die Referenzen zeigen auf die PIAs für Word 2010. Falls Sie mit 2007 oder früher arbeiten, werden Sie die Referenzen anpassen müssen (siehe Kapitel 9).

C#-Entwickler, die bereits Visual Studio 2010 mit .NET Framework 4.0 benutzen, benötigen diese Beispiele nicht unbedingt, da .NET 4.0 neu optionale und benannte Parameter unterstützt. Wir weisen jedoch darauf hin, dass solcher Code langsamer laufen könnte.

Die Zukunft von VBA

Die Rückwärtskompatibilität von VBA ist für die nächste Version von Word (»Office 15«) weiterhin gewährleistet. Somit wird das aus dem vorliegenden Buch Erlernte auch für diese Version seine Gültigkeit haben. Aufgrund 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 zufriedengestellt.


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 mit Office 2007 eingeführten OpenXML-Dateiformate ist es nun möglich, Dokumente zu erstellen, zu bearbeiten und zu 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 behält VSTO nach wie vor seinen Platz in dieser 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 Programmierumgebung – den Visual Basic-Editor – vor. Eine Zusammenfassung zu den VBA-Grundlagen, erste allgemeine Beispiele, eine Diskussion des Office Sicherheitskonzepts 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.
-  **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 2008 und VSTO 2010 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 des Menübands (in Word 2007 Multifunktionsleiste genannt) 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 zum Erstellen von Dokumenten im OpenXML-Dateiformat.

Stichwortverzeichnisse

Am Ende des Buchs 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 Begleitdateien 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).

Wenn Sie dieses Buch ohne Begleitmedium erworben haben (z.B. als E-Book), können Sie die für das Durcharbeiten notwendigen Dateien aus dem Internet herunterladen. Rufen Sie dazu die folgende Adresse auf und geben Sie – wie auf der Internetseite beschrieben – die Teilnummer der ISBN zu diesem Buch ein:

<http://www.microsoft-press.de/support.asp?cnt=support>

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 dritten Auflage Platz für die neuen Themen geschaffen werden konnte, mussten einige Seiten entfernt werden. In erster Linie betraf 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 Ordner *\Bonus* und *\Beilagen*.

Im Buch wird an verschiedenen Stellen auf Informationsquellen im Internet verwiesen. Die entsprechende Internetadresse (URL) ist jeweils direkt im Text mit angegeben. Damit diese zum Teil recht langen und kryptischen Zeichenfolgen nicht manuell abgetippt werden müssen, sind diese Adressen in Form von einzelnen *.url*-Dateien innerhalb der Begleitdateien vorhanden. Sie finden diese Dateien jeweils im zugehörigen Kapitelordner *\Beispiele\KapXX\Internet*.

Weitere Informationen zu den Begleitdateien zum Buch sind im Anhang zusammengefasst.

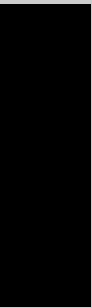
Das Autorenteam, im September 2010.

Teil A

Grundlagen der Arbeit mit VBA

In diesem Teil:

Kapitel 1	Word-Makros	25
Kapitel 2	VBA-Grundlagen	55
Kapitel 3	Windows-APIs in VBA nutzen	117



Kapitel 1

Word-Makros

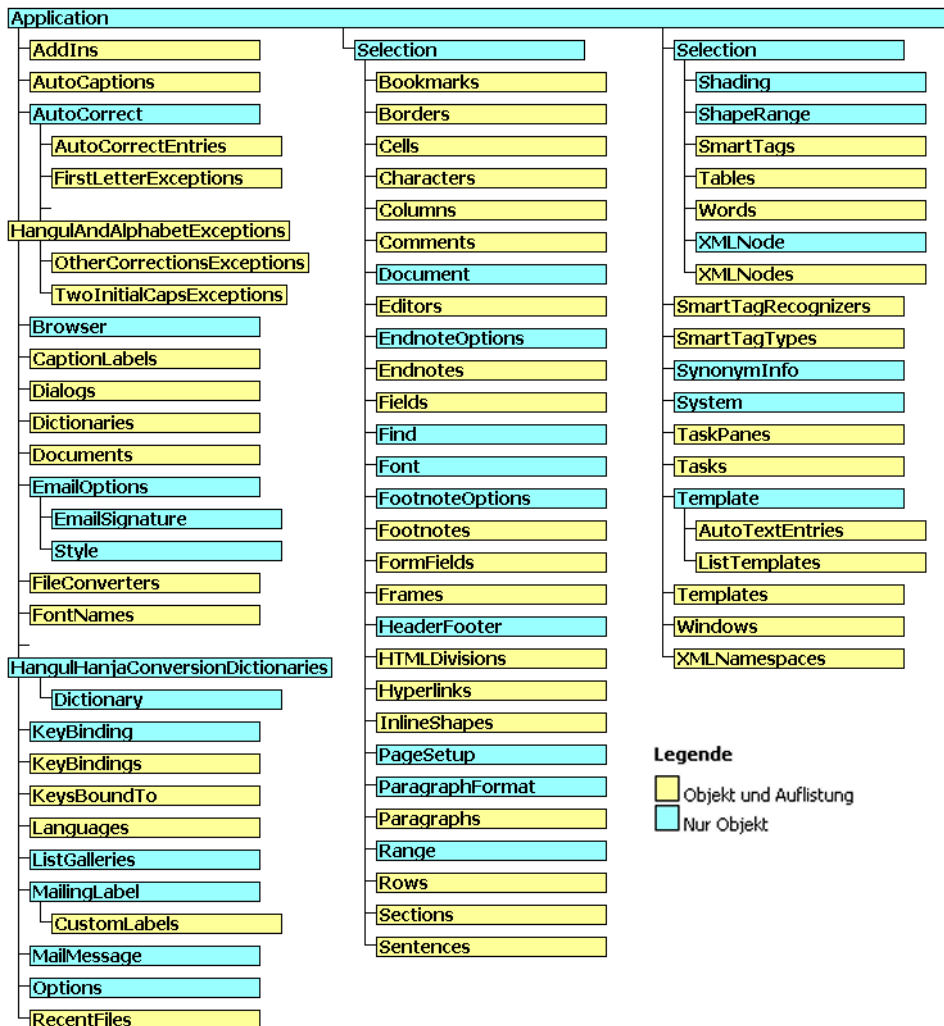
In diesem Kapitel:

Aller Anfang ist schwer	26
Programmierhilfen	27
Makros in die Benutzerschnittstelle einbinden und verwalten	41
Makrosicherheit	45
Zusammenfassung	54

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. Die Änderungen, die am Objektmodell von Word 2010 vorgenommen wurden, sind auf der Seite [http://msdn.microsoft.com/en-us/library/ee836186\(office.14\).aspx](http://msdn.microsoft.com/en-us/library/ee836186(office.14).aspx) zusammengefasst worden.

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



Wir sind uns bewusst, dass bei dieser Abbildung das Objektmodell einer sehr alten Programmversion dargestellt wird. Es ist jedoch die einzige uns bekannte Grafik, welche die Objekte auf einen Blick darstellt. Für die nachfolgenden Programmversionen von Word würde die Grafik entsprechend komplexer ausfallen, da das Objektmodell jeweils um die Neuerungen im Programm erweitert wurde. Eine Übersicht zum aktuellen Objektmodell finden sie hier <http://office.microsoft.com/client/helpcategory.aspx?CategoryID=CH806001069990&lcid=1033&NS=WIN-WORD%2EDEV&Version=14>.

Programmierhilfen

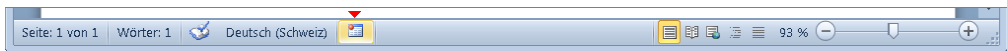
Wie kann also auf eine einfache Art der Einstiegspunkt zur Lösung des Problems gefunden werden? Die schnellste Hilfe erhält der Entwickler durch die Verwendung des Makrorekorders. Mit etwas Glück liefert dieser die zu einer Aufgabe benötigten Objekte, Eigenschaften und Methoden. Die zugehörigen Details werden in einem zweiten Schritt in der Hilfe nachgeschlagen (siehe dazu den Abschnitt »Die Objektmodell-Hilfe – eine verborgene Schatzkammer« ab Seite 38). Danach wird der bereinigte Code in das effektive Makro oder in die Prozedur eingebaut.

Den Makrorekorder einsetzen

Der Makrorekorder kann im Menüband via *Entwicklertools/Code/Makro aufzeichnen* oder mittels Doppelklick auf das entsprechende Symbol in der Statusleiste (Abbildung 1.2) gestartet werden.

Damit die Registerkarte *Entwicklertools* im Menüband sichtbar ist, muss dieses via *Datei/Optionen/Menüband anpassen* eingeblendet werden.

Abbildg. 1.2 Startsymbol für den Makrorekorder in der Statusleiste



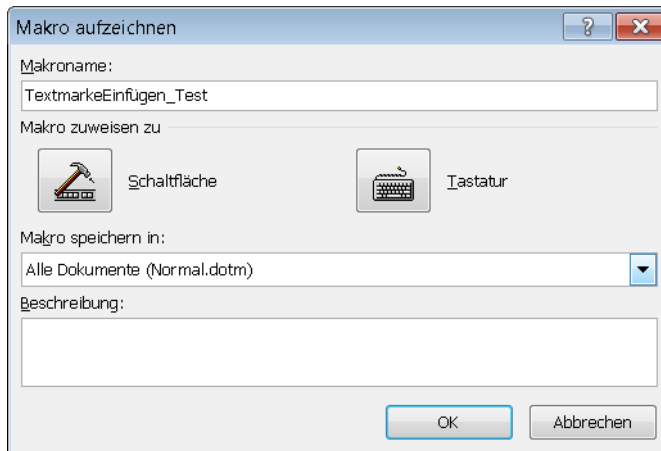
In dem nun eingeblendeten Dialogfeld *Makro aufzeichnen* (Abbildung 1.3) vergeben Sie für das Makro einen aussagekräftigen Namen (vorgeschlagen wird »Makron«, wobei *n* eine fortlaufende Nummer darstellt). Zusätzlich sollten Sie eine hilfreiche Beschreibung eintragen. Zudem besteht die Möglichkeit, das neue Makro einer Schaltfläche in der Symbolleiste für den Schnellzugriff (Word 2007 und 2010), im Menüband (Word 2010) 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. 1.3 Neben dem Makronamen ist es wichtig, den Speicherort für das Makro festzulegen



Nach Bestätigung des Dialogfelds *Makro aufzeichnen* (falls Sie weder *Symbolleiste* noch *Tastatur* angeklickt haben), kehren Sie ins Dokument zurück. In der Statusleiste wird jetzt das Symbol *Aufzeichnung beenden* eingeblendet.

Von nun an werden fast alle in Word ausgeführten Interaktionen in eine Prozedur im Codemodul *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 Dialogfelds.) 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 die Registerkarte *Ansicht* oder die Windows-Taskleiste erfolgt. Nicht erkannt werden Wechsel, die mit **Alt** + **Tab** 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 in der Registerkarte *Entwicklertools* auf die Schaltfläche *Aufzeichnung anhalten* klicken. Klicken Sie nochmals auf die Schaltfläche, um mit der Aufzeichnung fortzufahren.



HINWEIS

Bei der Einführung neuer Funktionen innerhalb von Word muss zusätzlich die Funktionalität des Makrorekorders angepasst werden. Je nach Zeitpunkt einer Änderung und vorhandenen Ressourcen wird die neue Funktionalität im Makrorekorder nicht aufgenommen. Als Folge werden verschiedene in Word 2007 bzw. in Word 2010 enthaltene Funktionalitäten nicht aufgezeichnet. Im Objektmodell sind diese Neuerungen jedoch vorhanden und können in der Hilfe nachgeschlagen werden (mehr dazu im Abschnitt »Die Objektmodell-Hilfe – eine verborgene Schatzkammer« ab Seite 38).

Eine Ausnahme besteht jedoch, in der ersten Ausgabe von Word 2007. Diese Version enthielt kein Objektmodell für die neuen Grafik- und Diagrammschnittstellen. Diese Objekte konnten lediglich manuell vom Benutzer oder mittels der Open XML-Technologie erstellt werden. Inzwischen wurde in diesem Bereich das Objektmodell in einem Service Pack für Word 2007 ergänzt und ist in Word 2010 vollständig vorhanden.

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

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 for Applications«). 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 1.4). Damit das aufgezeichnete Makro wunschgemäß arbeitet, muss der Anwender die Einfügemarke vor dem Ausführen unbedingt in der ersten Zelle einer Tabelle positionieren. Wird die Einfügemarke außerhalb einer Tabelle platziert, tritt ein Laufzeitfehler auf.

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

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

Spalte-1x	Spalte-2x	Spalte-3x	Spalte-4x	Spalte-5x
x	x	x	x	x
x	x	x	x	x

Listing 1.1 Die mit dem Makrorekorder aufgezeichnete Prozedur

```
Sub TabelleVorbereiten()
'
' TabelleVorbereiten Makro
' Makro aufgezeichnet am 02.07.2010 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

Die Dokumentation zum Word-Objektmodell ist VBA-orientiert, weil VBA die Office-Programmiersprache ist. Zudem spiegelt das Objektmodell eher die Benutzerschnittstelle und Arbeitsweise von Word wider und kann daher etwas »fremd« vorkommen. Im Abschnitt »Die Objektmodell-Hilfe – eine verborgene Schatzkammer« ab Seite 38 werden wir etwas näher darauf eingehen. Aber genau weil dem so ist, kann auch Ihnen der Makrorekorder behilflich sein.

Sehen Sie sich das Listing 1.2 an. 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. Diese Markierung wird wieder aufgehoben, sodass 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
    Dim i As Integer

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



In C# entspricht der Code für das Beispiel der Darstellung in Listing 1.3. Der Code ist Teil eines Windows Form-Projekts und wird durch die oberste Schaltfläche in Abbildung 1.5 ausgeführt. Die Prozedur enthält eine minimale Fehlerbehandlung, 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. 1.5 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()
{
    wd.Application wdApp = null;
    try
    {
        //GetActiveObject verursacht einen Fehler, falls die Anwendung noch nicht läuft.
    }
}
```

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

```

        wdApp = (wd.Application)wdMarshal.GetActiveObject("Word.Application");
    }
    catch (System.Runtime.InteropServices.COMException cex)
    {
        string exception = cex.ToString();
        string notRunning = "HRESULT: 0x800401E3 (MK_E_UNAVAILABLE)";
        if (exception.Contains(notRunning))
        {
            Type t = Type.GetTypeFromProgID("Word.Application");
            wdApp = (wd.Application)Activator.CreateInstance(t);
        }
        else
        {
            MessageBox.Show(exception);
            return;
        }
    }
    try
    {
        if (wdApp == null)
        {
            MessageBox.Show("Word läuft nicht.");
            return;
        }
        if (!wdApp.Visible) wdApp.Visible = true;
        if (wdApp.Documents.Count == 0)
        {
            MessageBox.Show("Kein geöffnetes Dokument gefunden.");
            return;
        }
        object objDirectionStart = wd.WdCollapseDirection.wdCollapseStart;
        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 objDirectionStart);
            sel = null;
            tbl = null;
            row = null;
        }

        //Das Word-Fenster anzeigen
        MessageBox.Show("Fertig!");
        wdApp.Activate();
    }
    catch (System.Runtime.InteropServices.COMException ex)
    {
        MessageBox.Show(ex.Message);
    }
}

```

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

```
finally
{
    wdMarschal.ReleaseComObject(wdApp);
    wdApp = null;
}
}
```

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
```

Die Codezeilen dieses Makros lassen erkennen, dass es sich bei einer AutoForm um ein Shape-Objekt handelt. Ferner verfügt das Shape-Objekt über die Methode `IncrementLeft` (in kleinen Schritten nach links verschieben). Mit diesen Informationen können in der Hilfe (siehe Kapitel 2) die jeweiligen Objekte erforscht werden.

TIPP

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

Wie wird jetzt aus einem wie in Listing 1.1 aufzeichneten Makro eine strukturierte, übersichtliche Prozedur, wie dies in Listing 1.2 der Fall ist? Dazu verhilft ein Programm, das auf Objekten basiert und auf deren Eigenschaften und Methoden zurückgreift. Dies ist ein Hauptziel dieses Handbuchs, dem vor allem die ersten beiden Teile gewidmet sind. Im ersten Teil stellen wir in 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.

CD-ROM

Die Beispieldatei *Bsp01_01.docm* 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 steht die gleiche Programmiersprache – VBA (Visual Basic for Applications) – 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]** oder im Menüband via *Entwicklertools/Code/Visual Basic* auf.

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 Codebibliotheken zu verwalten. Dieser Aspekt wird in Kapitel 20 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 *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.dotm* aufgezeichnet und bearbeitet, wird die *Normal.dotm* und nicht das aktuelle Dokument gespeichert. Speichern Sie umgekehrt das aktuelle Dokument, wird der Makrocode in der *Normal.dotm* oder einem anderen zum Dokument gehörenden Projekt nicht gespeichert.

Für große Projekte oder zwecks Codeaustausch ist es oft ratsam, Sicherheitskopien zu erstellen. Dies geht über den Menübefehl *Datei/Datei exportieren* des VB-Editors. Im Gegensatz zu den im Abschnitt »Makros kopieren« ab Seite 44 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 Codemodule 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 nutzen

Im Abschnitt »Den Makrorekorder einsetzen« ab Seite 27 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 1.6 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. 1.6 VBA im Codefenster des VB-Editors

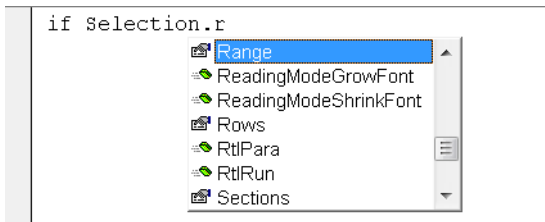
```
Sub TabelleVorbereiten2()
    Dim tbl As Word.Table
    Dim row As Word.Row
    Dim i As Integer

    If Selection.Range.Information(wdWithinTable) Then
        Set tbl = Selection.Tables(1)
        Set row = tbl.Rows(1)
        row.Range.Bold = True
        For i = 1 To row.Cells.Count
            row.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 unterstützt Sie mit der »IntelliSense«-Funktion.

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 1.7 dargestellt.

Abbildg. 1.7 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

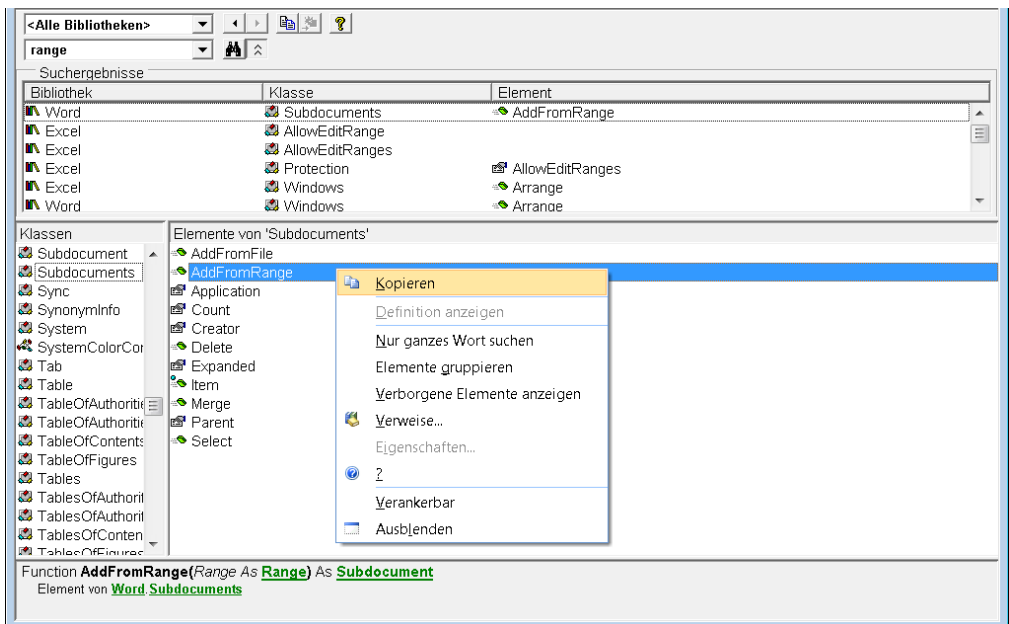
Für alle IntelliSense-Listen gilt: Drücken Sie **[Esc]**, um die Liste ungenutzt zu schließen.

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 **[F2]**.


Abbildg. 1.8 Der Objektkatalog bietet einen Zugriff auf die Objekthierarchie aller geladenen Bibliotheken



Wie in Abbildung 1.8 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.

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).

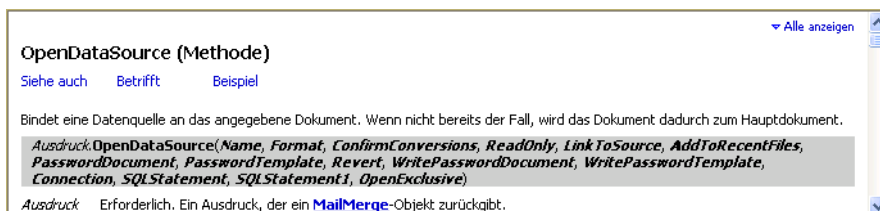
Die Suchergebnisse erscheinen im zweiten Teil des Objektkatalogs. Alle Elemente, in denen die gesuchte Zeichenkette enthalten ist, werden rechts in der dritten Spalte aufgelistet. Das Objekt (Klasse), dem sie angehören, befindet sich in der Spalte links daneben, und die zugehörige Anwendung (Bibliothek) 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 1.8, die Prozedursyntax. Die unterstrichenen Begriffe dienen jeweils als Hyperlink, worüber weitere Informationen in der Liste *Elemente* angezeigt werden.

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

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 1.9 mit der Abbildung 1.10. 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. 1.9 Die Methodenunterschrift für die *OpenDataSource*-Methode laut Hilfedokumentation

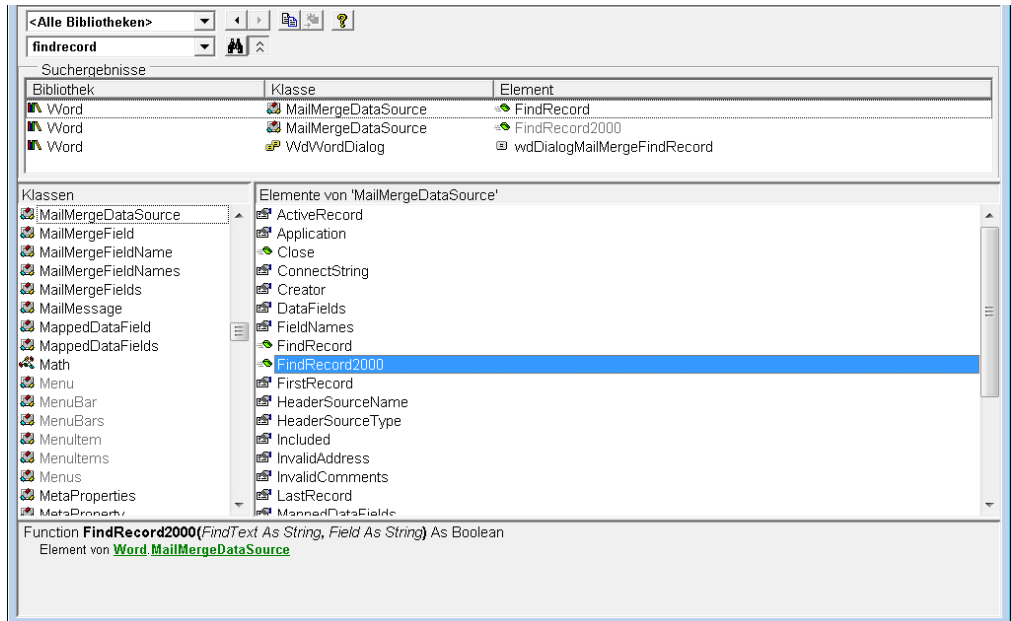


Abbildg. 1.10 Die Methodenunterschrift für die *OpenDataSource*-Methode laut Objektkatalog, einschließlich *SubType*



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 1.11 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.

Abbildg. 1.11 In Word 2002 gibt es die Methoden *FindRecord* sowie *FindRecord2000*. Letztere ist normalerweise verborgen.



Die Objektmodell-Hilfe – eine verborgene 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-Hilfeschchnittstellen sogar auf unterschiedlichen Technologien auf.

In diesem Abschnitt werden wir eine Übersicht der Objektmodell-Hilfeschchnittstellen in Word 2000 bis 2010 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 for Applications-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 hüpft 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 ausgewählten Thema angezeigt. Informationen können in jeder der drei Registerkarten gesucht werden. Bei Bedarf kann der Teil mit den Registerkarten ein- und ausgeblendet werden.

- **Word 2002** Das Hilfenfenster 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* aktivierten 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.

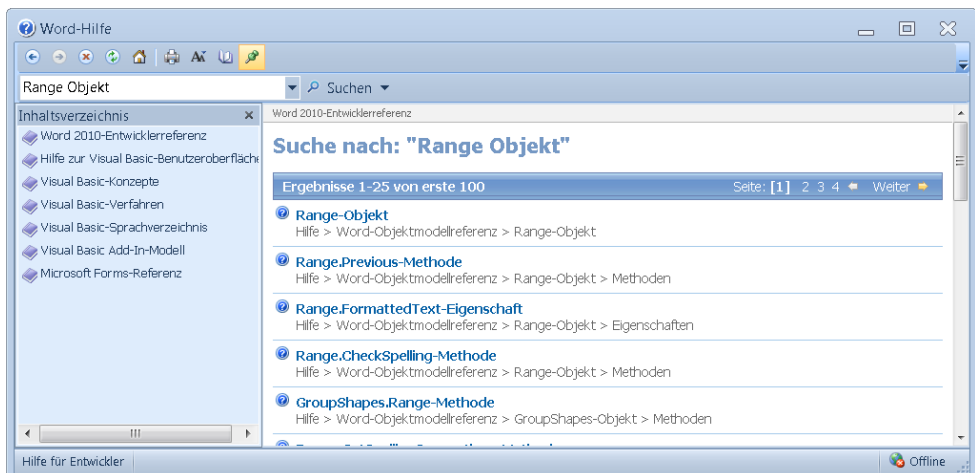
Die Liste der Programmierreferenzen ist nicht erweiterbar, egal ob ein aktiver Verweis zu einer Anwendungsbibliothek vorliegt.

Bei Auswahl eines Themas wird ein separates Hilfenfenster 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 und 2010** Der Aufgabenbereich ist wieder verschwunden. Es wird ein unabhängiges Fenster eingeblendet, wie in Abbildung 1.12 ersichtlich. Der gesamte Hilfebereich 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.

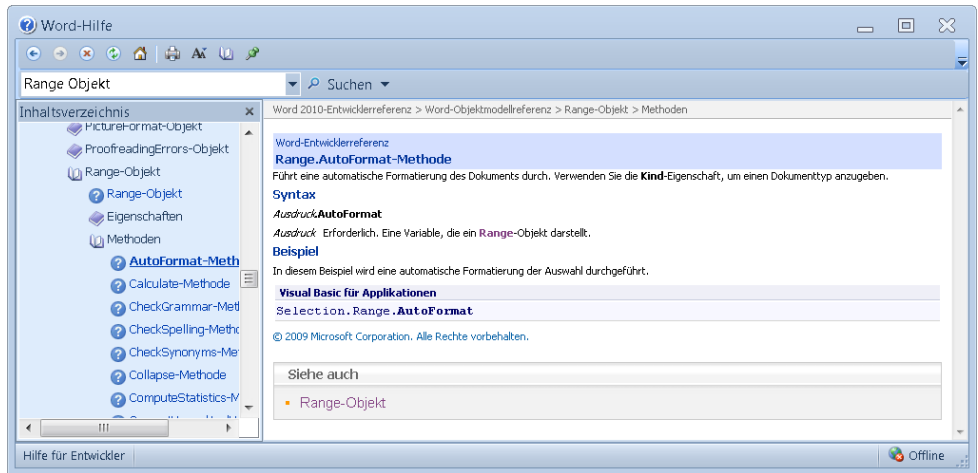
Abbildg. 1.12

Das Word 2010-Hilfenfenster



Das Hilfefenster können Sie mit der in Abbildung 1.12 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 1.13 veranschaulicht dies anhand der Auflistung aller Methoden und Eigenschaften des Range-Objekts. Per Klick auf einen Eintrag wird die Hilfe zum Thema eingeblendet.

Abbildg. 1.13 Auflistung aller Methoden und Eigenschaften eines Objekts



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 Hilfdateien direkt öffnen oder sie über die Anwendungsoberfläche (VB-Editor in Word) einsehen.

Die gesamte Entwicklerreferenz für Word 2010 steht in englischer Sprache auf MSDN zur Verfügung ([http://msdn.microsoft.com/en-us/library/ee861527\(office.14\).aspx](http://msdn.microsoft.com/en-us/library/ee861527(office.14).aspx)).

Jene für Word 2007 steht ebenfalls in englischer Sprache auf MSDN zur Verfügung (<http://msdn2.microsoft.com/en-us/library/bb244391.aspx>).

Die Entwicklerreferenz für Word 2003 steht in englischer Sprache auf MSDN zum Herunterladen bereit (<http://www.microsoft.com/downloads/details.aspx?familyid=179bee82-e6e6-4b78-aff9-9a541167541f&displaylang=en>) oder als Onlineversion auf [http://msdn2.microsoft.com/en-us/library/aa272078\(office.11\).aspx](http://msdn2.microsoft.com/en-us/library/aa272078(office.11).aspx).

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 Codefenster 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.

Makros in die Benutzerschnittstelle einbinden und verwalten

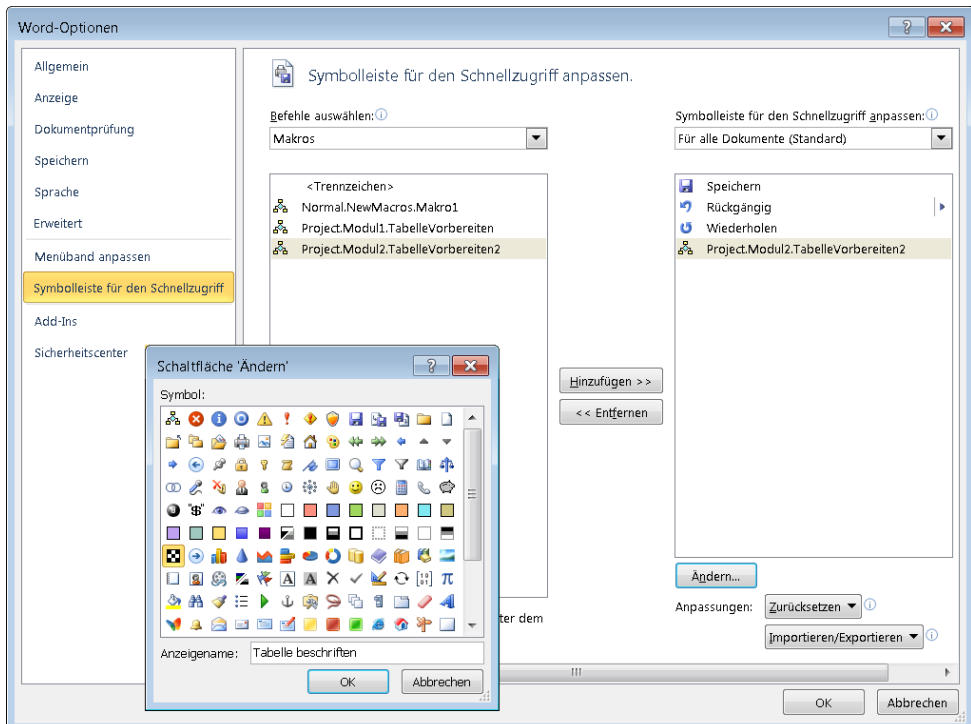
Wie im Abschnitt »Den Makrorekorder einsetzen« ab Seite 27 beschrieben, können Makros in die Symbolleiste für den Schnellzugriff und ab Word 2010 im Menüband aufgenommen werden. Bei Bedarf kann zusätzlich eine Tastenkombination zugewiesen werden. Was ist aber, wenn Sie diesen Schritt erst später vollziehen wollen? Dies kann jederzeit über die Benutzerschnittstelle nachgeholt werden.

Makro einem Symbol zuweisen

Um in Word ein Makro einer Symbolschaltfläche in der Symbolleiste für den Schnellzugriff zuzuweisen, führen Sie die folgenden Schritte durch (die Abbildung 1.14 veranschaulicht diesen Vorgang):

1. Klicken Sie zunächst auf die *Office*-Schaltfläche (Word 2007) bzw. auf die Registerkarte *Datei* (Word 2010) und anschließend auf die Schaltfläche *Word-Optionen*.

Abbildg. 1.14 Makros der Symbolleiste für den Schnellzugriff zufügen und die Symbolschaltfläche anpassen



2. Öffnen Sie die Kategorie *Symbolleiste für den Schnellzugriff*.
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*.

Um ein Makro einem Befehl im Menüband zuzuweisen (ab Word 2010) gehen Sie analog vor. Wählen Sie in den *Word-Optionen* anstelle des Eintrags *Symbolschaltfläche in der Symbolleiste* den Eintrag *Menüband anpassen*.

Bei den manuellen Anpassungen am Standard-Menüband können nicht alle Änderungen uneingeschränkt vorgenommen werden. So können die internen Registerkarten zwar ausgeblendet, aber nicht gelöscht werden. Die internen Gruppen innerhalb einer Registerkarte können nicht bearbeitet, jedoch ausgeblendet und durch gleichnamige ersetzt werden. Es können eigene Registerkarten angelegt oder die bestehenden durch eigene Gruppen und die zugehörigen Befehle ergänzt werden. Wobei hier interne Befehle von Word wie auch Makros zugewiesen werden können.

Die Änderungen am Menüband und an der Symbolleiste für den Schnellzugriff werden gemeinsam in einer Datei verwaltet. In Word 2007 hieß diese Datei *Word.qat*, ab Word 2010 ist der Name *Word.officeUI*. Die Datei wird vom Programm im Ordner *C:\Users\[Benutzername]\AppData\Local\Microsoft\Office* gespeichert.

Ab Word 2010 besteht zusätzlich die Möglichkeit, diese Anpassungen in eine Datei zu exportieren (*Word-Anpassungen.exportedUI*) und auf einem anderen System zu importieren. Doch aufgepasst, wird eine solche Datei importiert, werden gleichzeitig alle bestehenden Änderungen überschrieben und gehen somit verloren.

Bei der *officeUI*- wie auch bei der *exportedUI*-Datei handelt es sich um reine XML-Dateien. Somit lassen sich diese Daten sehr einfach mit XML-Werkzeugen oder einem Texteditor bearbeiten.

HINWEIS

In Word 2007 kann vom Benutzer nur die Symbolleiste für den Schnellzugriff 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 das Menüband mittels Programmierung ergänzt wird, erfahren Sie in Kapitel 16.

Makro einer Tastaturkombination zuweisen

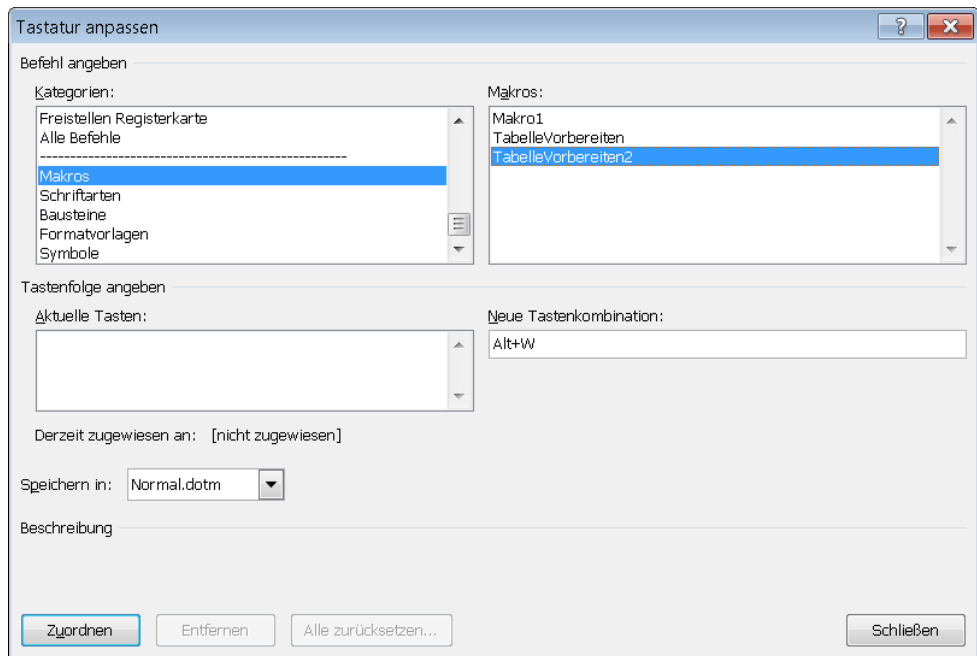
Um ein Makro einer Tastenkombination zuzuweisen, gehen Sie wie folgt vor:

1. Klicken Sie zunächst auf die *Office*-Schaltfläche (Word 2007) bzw. auf die Registerkarte *Datei* (Word 2010) und anschließend auf die Schaltfläche *Word-Optionen*.
2. Öffnen Sie die Kategorie *Symbolleiste für den Schnellzugriff* bzw. *Menüband anpassen*.
3. Wählen Sie die Schaltfläche *Anpassen*.
4. Legen Sie im geöffneten Dialogfeld *Tastatur anpassen* (Abbildung 1.15) den Speicherort fest (siehe dazu Kapitel 14).
5. Wählen Sie die Kategorie *Makros* aus.

6. Markieren Sie in der Liste *Makros* den Eintrag, dem eine Tastenkombination zugewiesen werden soll.
7. Klicken Sie in das Feld *Neue Tastenkombination*.
8. 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 Rück-Taste und geben ein anderes Kürzel ein.

Abbildg. 1.15

Die Tastenkombination Alt + W ist noch nicht belegt und darf dem Makro *TabelleVorbereiten2* zugewiesen werden



HINWEIS

Wie Tastenkombinationen durch das Objektmodell definiert werden, wird in Kapitel 17 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 unten stehende Reihenfolge für Word festgelegt:

- Makros werden durch *ProjektName.ModulName.MakroName* identifiziert, wie dies die Liste *Befehle* in Abbildung 1.14 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 einer Tastenkombination oder in älteren Word-Versionen zusätzlich einer Symbolleiste zugewiesen, geht diese Verbindung verloren und muss neu hergestellt werden.
- Für Tastenkombinationen wie auch Symbolleisten 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.dotm*

WICHTIG Ein Makro erscheint nicht in der Liste in der Benutzerschnittstelle

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

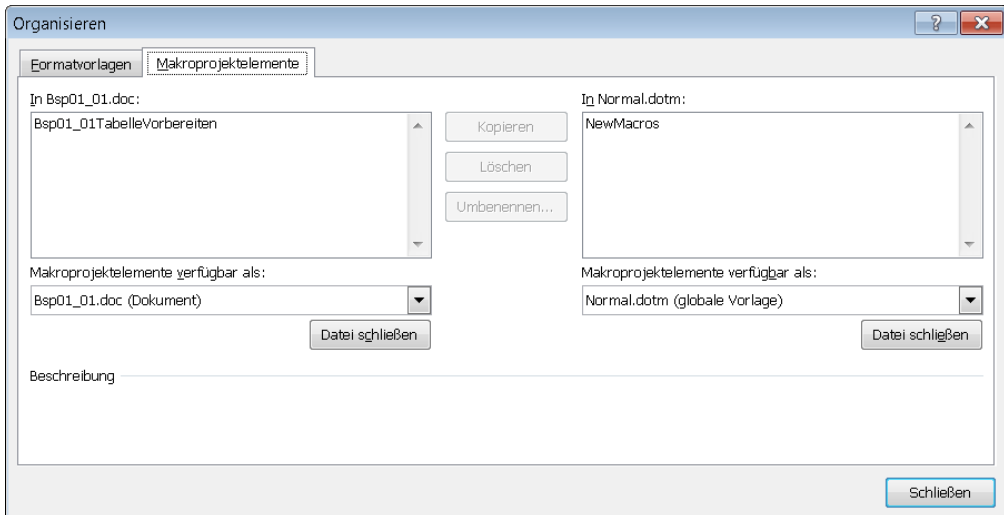
- Prozeduren, die als *Private* bezeichnet sind (mehr zu *Private* lesen Sie in Kapitel 2)
- Prozeduren, für die Argumente definiert sind
- Prozeduren, die einen Wert zurückgeben (Funktionen)
- Prozeduren, die sich in einem Modul befinden, das die Anweisung *Option Private Module* enthält
- Prozeduren,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

Makros kopieren

Oft werden Makros zuerst in der *Normal.dotm* erstellt, weil dies der standardmäßige Speicherort ist. Erst später wird realisiert, dass das Makro besser in einem einzelnen Dokument oder in einer Dokumentvorlage untergebracht wäre. Der Code kann dann problemlos im VB-Editor kopiert, gelöscht und eingefügt oder mittels einer Sicherheitskopie, wie im Abschnitt »Code speichern« ab Seite 34 beschrieben, ausgetauscht werden.

Was aber, wenn Sie Makros mit anderen Anwendern teilen wollen und diese Personen sich nicht mit dem VB-Editor auseinandersetzen möchten? Dazu bietet Word ein tolles Werkzeug an, mit welchem einzelne Codemodule (oder auch Formatvorlagen, und in Word 2003 und früher Symbolleisten und AutoTexte) zwischen Word-Dateien kopiert werden können: das Dialogfeld *Organisieren* (siehe Abbildung 1.16).

Abbildg. 1.16 Makros mit dem Dialogfeld *Organisieren* problemlos verwalten und zwischen Word-Dateien kopieren



Um das Dialogfeld *Organisieren* aufzurufen, muss im Menüband der Befehl *Entwicklertools/Vorlagen/Dokumentvorlage* und anschließend die Schaltfläche *Organisieren* angewählt werden.

HINWEIS Falls mit einer Programmversion vor Word 2007 gearbeitet wird und die Makros mit einer Symbolleiste verbunden sind, empfiehlt es sich, zuerst die Makros und anschließend die Symbolleiste in die andere Datei zu kopieren.

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

HINWEIS Die Funktionalität *Organisieren* ist auch im Word-Objektmodell vorhanden. Schlagen Sie in der VBA-Hilfe die Begriffe *OrganizerCopy*, *OrganizerDelete* und *OrganizerRename* nach.

Die programmatische Erstellung von Makros wird in Kapitel 20 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.

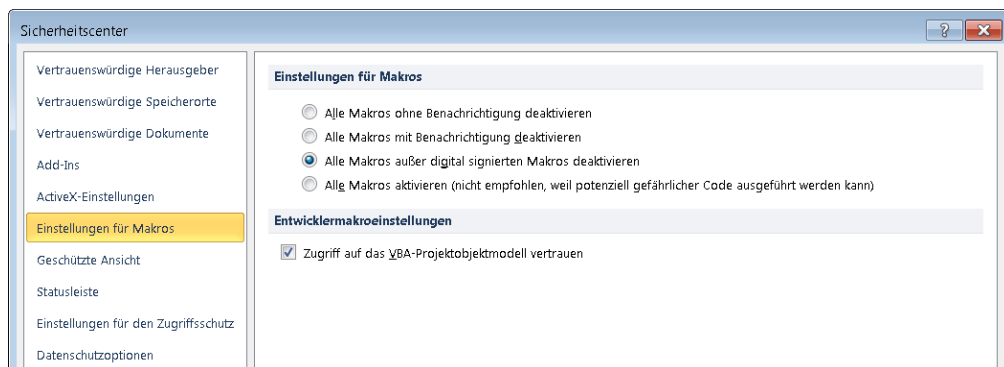
In Word 97 hat Microsoft als Gegenmaßnahme eine mehrstufig aufgebaute sogenannte *Makrosicherheit* integriert. Jetzt konnte der Anwender festlegen, ob die Makros innerhalb von Dokumenten überhaupt aktiviert und ausgeführt werden oder nicht.

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 festzulegen, wählen Sie *Datei/Optionen* und dort die Kategorie *Sicherheitscenter* aus. Diese Seite enthält Links zu weiteren Informationen über die Sicherheitsmaßnahmen sowie die Schaltfläche *Einstellungen für Makros*. Klicken Sie darauf, um das Dialogfeld in Abbildung 1.17 einzublenden.

Abbildg. 1.17 Das Word-Sicherheitscenter



Das Dialogfeld umfasst die in Tabelle 1.1 aufgeführten Kategorien. Diese ermöglichen eine verfeinerte Einstellung der Zugriffsmöglichkeiten für die Fernsteuerung von Word gegenüber früheren Versionen. Für Einzelheiten schlagen Sie in der Word-Hilfe den Ausdruck »Vertrauensstellungscenter« nach.

Tabelle 1.1 Die Kategorien des Dialogfelds *Sicherheitscenter*

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 seit Word 2007 können Dateien in beliebigen Ordnern auf dem Rechner vertraut werden. Das bedeutet, Makrocode in diesen Dateien ist aktiviert; Makros in anderen Speicherorten werden gesperrt. Automatisch vertraut werden die Speicherorte <i>[Laufwerk]\Programme\Microsoft Office\Templates</i> sowie <i>[Laufwerk]\Programme\Microsoft Office\Office14\Startup</i> .
Vertrauenswürdige Dokumente	Neu in Word 2010 können Dateien als vertrauenswürdig eingestuft werden, dies kann innerhalb eines Netzwerks sinnvoll sein

Tabelle 1.1 Die Kategorien des Dialogfelds *Sicherheitscenter* (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.
Geschützte Ansicht	Hier wird festgelegt, wie potentiell gefährliche Dokumente geöffnet werden sollen. Sind die entsprechenden Optionen aktiviert, so werden die Dateien ohne zusätzlichen Sicherheitshinweis in einem eingeschränkten Modus geöffnet.
Statusleiste	Hier geht es nicht um die Word-Statusleiste, sondern um einen Balken, der zwischen dem Menüband und dem Dokument mit Sicherheitsmeldungen eingeblendet wird. In dieser Kategorie wird ihre Aktivierung geregelt.
Einstellungen für den Zugriffsschutz	Legt für die unterschiedlichen Dateiformate der verschiedenen Word-Versionen fest, ob diese nur in der geschützten Ansicht geöffnet werden können. Gleichzeitig kann das Speichern in diesen Dateiformaten verweigert werden.
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 *Sicherheitsstufe* entspricht in etwa der Kategorie *Einstellungen für Makros* im Sicherheitscenter von Word. Die Sicherheitsstufe *Sehr hoch* wurde erst in Word 2003 eingeführt.

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

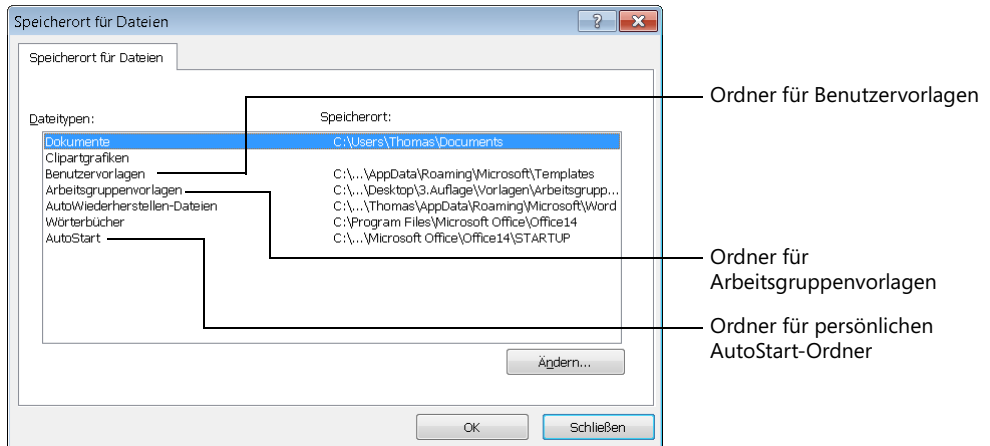
- *C:\Programme\Microsoft Office\Office11\Startup* als allgemeiner AutoStart-Ordner
- *C:\Dokumente und Einstellungen\[Benutzername]\Anwendungsdaten\Microsoft\Word\Startup* als persönlicher AutoStart-Ordner, sofern in den Optionen kein anderer Ordner eingetragen ist
- *C:\Dokumente und Einstellungen\[Benutzername]\Anwendungsdaten\Microsoft\Vorlagen* 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 1.18 ersichtlich, können die einzelnen Speicherorte bei Bedarf individuell festgelegt werden. Werden keine Einträge gesetzt, sind die oben aufgezählten Ordner als Standardwerte gültig. Um die Speicherorte anzupassen, wählen Sie *Datei/Optionen/Erweitert/Allgemein/Dateispeicherorte* und tragen im Dialogfeld *Speicherort für Dateien* 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. 1.18 Legen Sie die gewünschten Verzeichnisse für die Add-Ins und Vorlagen fest



Wir Autoren empfehlen die standardmäßigen Einstellungen für die Makrosicherheit. Für Word 2007 und 2010 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, sodass 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*.

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

Das Zertifikat wird von einer anerkannten Zertifizierungsstelle ausgestellt^a. 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^b) geliefert. Um die Integrität dieses Zertifikats zu garantieren, wird es mit dem privaten Schlüssel der Zertifizierungsstelle digital unterschrieben.^c

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

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

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

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

Ein signiertes Makro kann nur zusammen mit dem zugehörigen Zertifikat aktiviert werden, sofern die Sicherheitseinstellungen, wie im Abschnitt »Sicherheitsstufe anpassen« ab Seite 46 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 ausgewechselt).

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

b 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.

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

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

Tabelle 1.2 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.

In der Anwendungshilfe ist dieses Thema ebenfalls ausführlich beschrieben. Suchen Sie den Begriff »Digitales Signieren eines Makroprojekts« und wählen Sie das gleichnamige Thema aus.

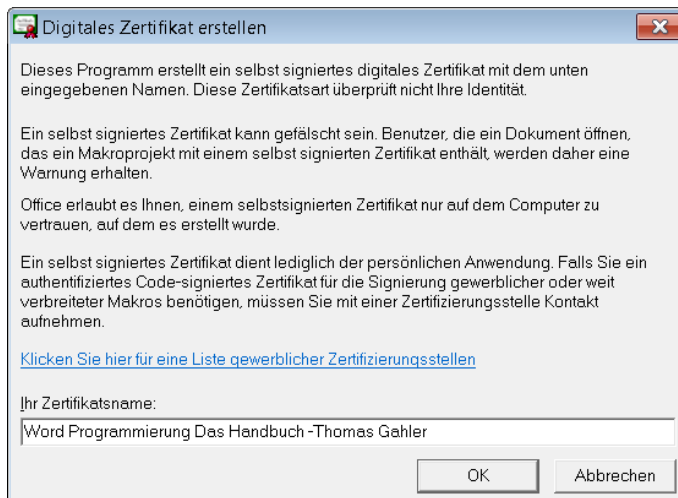
Zum Lieferumfang von Microsoft Office gehört das kleine Programm *selfcert.exe*. Die Datei wird, sofern die entsprechende Option ausgewählt wurde, während der Programminstallation im Verzeichnis *C:\Programme\Microsoft Office\Office14* angelegt. Mit diesem Hilfsprogramm lassen sich auf der eigenen Arbeitsstation 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\Office14* das Programm *selfcert.exe*.
2. Legen Sie für Ihre Signatur einen aussagekräftigen Namen fest und bestätigen Sie diese Angaben mit OK.

Abbildg. 1.19

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 (privater Schlüssel) 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 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

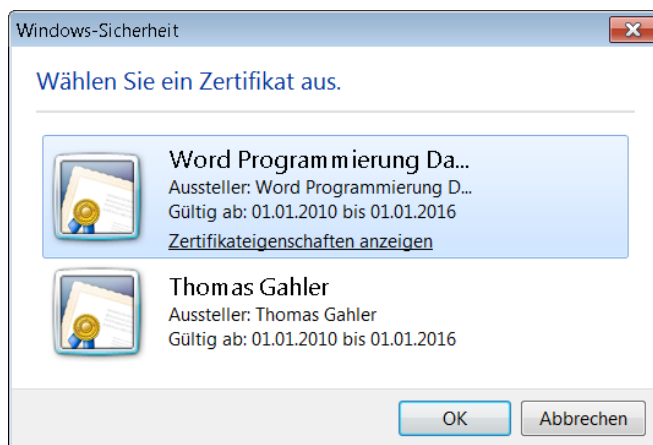
Damit die Makros innerhalb eines Add-Ins, einer Dokumentvorlage oder eines Dokuments im Zusammenspiel mit den empfohlenen Sicherheitseinstellungen aktiviert werden können, müssen die einzelnen Projekte mit einer digitalen Signatur ausgestattet werden. Diese Signatur stellt sicher, dass die Makros von 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 *Wählen Sie ein Zertifikat aus* das gewünschte Zertifikat in der Liste der vorhandenen Zertifikate und bestätigen Sie die Auswahl mit *OK*.
4. Bestätigen Sie anschließend die erfolgte Zuweisung ebenfalls mit *OK*.

Abbildg. 1.20

Weisen Sie das gewünschte Zertifikat dem Projekt zu



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

Zertifikat zur Signatur erstellen



Wurden Makros mit der gemäß dem Abschnitt »Eigene Signatur mittels *selfcert.exe* erstellen« ab Seite 50 erzeugten Signatur signiert, kann in einem zweiten Schritt das zugehörige Zertifikat erzeugt werden. Dieses Zertifikat kann vorgängig weitergereicht und installiert werden. Es stellt den sogenannten öffentlichen Schlüssel dar. Wird das Zertifikat vorab installiert, lassen sich die Dokumente, die später ausgetauscht werden, umgehend nutzen. Ansonsten kann das Zertifikat beim ersten Kontakt direkt aus der signierten Datei eingelesen werden.

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 *Wählen Sie ein Zertifikat aus* klicken Sie auf den Link *Zertifikateigenschaften 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. 1.21

Bestimmen Sie den Dateinamen der Zertifikatdatei

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« ab Seite 51 beschrieben, signiert und weitergereicht, kann 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 Zertifikatdatei 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.

Wurde das Zertifikat nicht vorab auf der Arbeitsstation eingelesen, kann dies beim ersten Kontakt mit einem signierten Dokument aus der besagten Quelle erfolgen. Beim Öffnen des Dokuments wird eine entsprechende Warnung ausgegeben (Abbildung 1.22). Um das Zertifikat direkt einzulesen, gehen Sie wie folgt vor:

1. Öffnen Sie das Dokument und klicken Sie im Dialogfeld *Sicherheitshinweis für Microsoft Word* auf den Link *Signaturdetails anzeigen*.
2. Wählen Sie auf der Registerkarte *Allgemein* die Schaltfläche *Zertifikat anzeigen* an.
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. 1.22

Ein signiertes Dokument wird geöffnet, dessen Zertifikat noch nicht installiert ist



Die Aufnahme des Zertifikats in den Zertifikatspeicher 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.

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 27 ff.)
- Nützliche Werkzeuge im VB-Editor wurden ab Seite 34 vorgestellt, unter anderen die Hilfe und das Objektkatalog
- Weiterhin wurde beschrieben, wie der Benutzer seine Makros in der Word-Umgebung über eine Schaltfläche oder Tastaturkombination zugänglich macht (Seite 41 ff.)
- Ein weiterer Abschnitt beschäftigte sich mit der integrierten Makrosicherheit von Word (Seite 45 ff.) und wie ein VBA-Projekt digital signiert werden kann (Seite 50 ff.)

Kapitel 2

VBA-Grundlagen

In diesem Kapitel:

Variablen	56
Konstanten	69
Benutzerdefinierte Typen	71
Nützliche VBA-Funktionen	79
Bedingungen	85
Schleifen	88
Compilieranweisungen	91
Code im VB-Editor debuggen	94
Fehlerbehandlung	99
Dateisystem-Operationen	105
Zusammenfassung	116

In diesem Kapitel versuchen wir Ihnen die Grundlagen zu Visual Basic for Applications (VBA) näherzubringen. 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« ab Seite 56 behandelt
- Der Nutzen von Konstanten, die Deklaration derselben und deren Gültigkeit innerhalb des Projekts werden ebenfalls in diesem Kapitel im Abschnitt »Konstanten« ab Seite 69 dargelegt
- Die Deklaration und der Umgang mit eigenen Datentypen, das Erstellen und Anwenden von eigenen Aufzählungen, sogenannten Enumerationen, werden im Abschnitt »Benutzerdefinierte Typen« ab Seite 71 erörtert
- Die Programmverzweigungen werden im Abschnitt »Bedingungen« ab Seite 85 und die Programmwiederholungen im Abschnitt »Schleifen« ab Seite 88 näher betrachtet
- Im Abschnitt »Code im VB-Editor debuggen« ab Seite 94 wird gezeigt, wie das Verhalten der erstellten Programmsequenzen analysiert wird. Und im Abschnitt »Fehlerbehandlung« ab Seite 99 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« ab Seite 105 behandelt.



HINWEIS

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 Code from VBA to Visual Basic .NET« 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://msdn.microsoft.com/en-us/library/skw8dhdd\(vs.71\).aspx](http://msdn.microsoft.com/en-us/library/skw8dhdd(vs.71).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
```

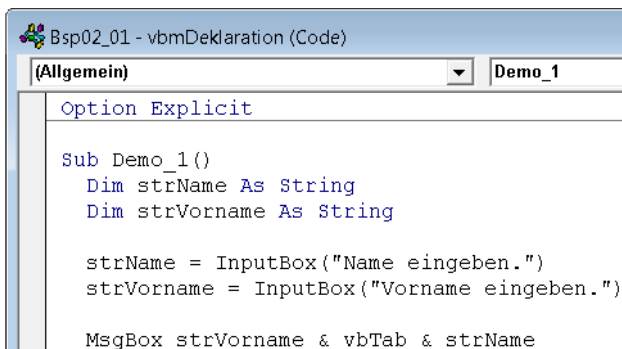
Standarddatentypen

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 VBA-Hilfe 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

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

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

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

```
Dim bStatus As Boolean
```


HINWEIS

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

Integer

Variablen vom Datentyp Integer enthalten nur ganze Zahlen, und zwar im Bereich von –32.768 bis 32.767. Deshalb eignet sich dieser Datentyp in erster Linie für Aufzählungswerte¹. 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

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

Single

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

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

Variant

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

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

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

**HINWEIS**

Die .NET-Umgebung unterstützt den Datentyp `Variant` nicht. Stattdessen muss für Variablen undefinierten Inhalts der Datentyp `Object` (bzw. `object` in C#) 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, erfolgt die Zuweisung an das entsprechende Objekt erst zur Laufzeit. Eine Bindung des Objekts bereits während des Kompilierungsvorgangs kann erreicht werden, indem die Variable mit dem Namen einer bestimmten Klasse deklariert wird. Der Standardwert ist `Nothing`.

Mehr zum Thema »Bindung zur Laufzeit bzw. zur Kompilierungszeit« erfahren Sie in Kapitel 9.

```
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 mithilfe 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:

- 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.

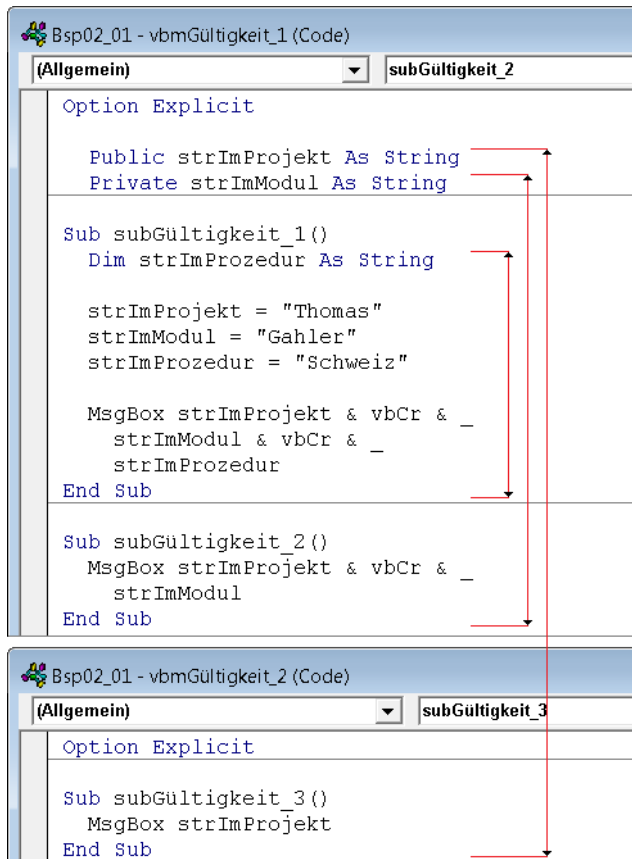
Dim

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« ab Seite 63.
- 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-
dauern 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.

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

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

```
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 VBA-Hilfe zum Thema »Typ-Umwandlungsfunktionen«.

Tabelle 2.1 Die wichtigsten *Typ*-Umwandlungsfunktionen im Überblick

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

Mithilfe 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**

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

Das gilt jedoch nicht für C# (mit Ausnahme der neuen Funktionalität im .NET Framework 4.0, das allerdings die Ausführung verlangsamen könnte). Falls Sie VBA-Code nach C# portieren, müssen Sie alle Datentypen explizit mit den entsprechenden .NET-Funktionen umwandeln.

Verknüpfen von Zeichenketten

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

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

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

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

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

PROFITIPP

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

- Jede Umwandlung eines Datentyps erfolgt durch die zugehörige Umwandlungsfunktion
- 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 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 an die Prozedur übergeben. Dies bedeutet, dass die Adresse der Variablen an die Prozedur übergeben wird. Innerhalb der Prozedur steht die Variable zur Bearbeitung zur Verfügung. Der Ursprungswert der eigentlichen Variablen kann verändert werden.

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



HINWEIS

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

Argument als Wert übergeben – ByVal

Das folgende Beispiel einer Dreiecksflächenberechnung¹ 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
```

¹ 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 Prozeduraufruf zur Berechnung der Dreiecksfläche so einfach wie möglich zu halten, könnten 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.

Mithilfe 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) & _
        vbCr & "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 nahelegen, 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

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. Von .NET-Framework wird die Anweisung `Option Base` nicht unterstützt.

Zudem wird dies in .NET Framework nicht unterstützt; dort ist die Untergrenze eines Datenfelds zwingend 0 (Null).



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

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

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

```
Dim strAdresse(5) As String
```

oder die Zuweisung der Größe mithilfe 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 strAdresseZeile As String
    Dim strAdresse() As String
    Dim intZähler As Integer
    Dim doc As Word.Document

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

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

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.

CD-ROM Die aufgeführten Codesequenzen finden Sie in der Beispieldatei *Bsp02_01.docm* 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 Werts 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.
- Mithilfe 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 regelmäßiger 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.

CD-ROM Die aufgeführten Codesequenzen finden Sie in der Beispieldatei *Bsp02_02.docm* 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

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
```

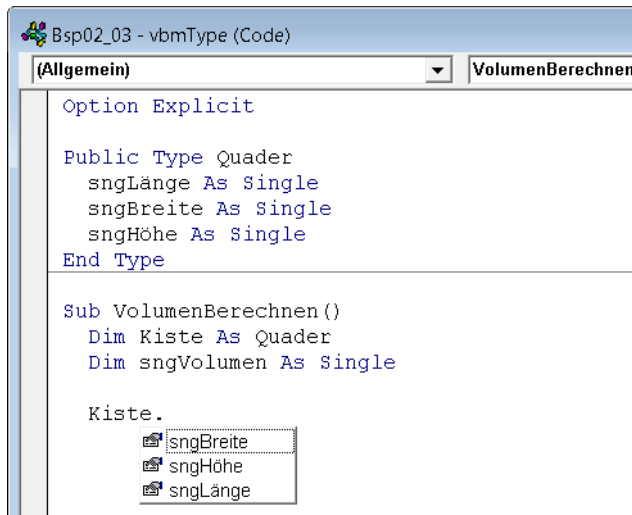
Benutzerdefinierte Typen können grundsätzlich nur auf Modulebene deklariert werden. Es besteht jedoch die Möglichkeit, die Gültigkeit für den neuen Typ auf das entsprechende Modul zu beschränken, und zwar durch Verwendung des Schlüsselworts `Private` innerhalb der Deklarationszeile.

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

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 näherbringen. 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

Überwachungsausdrücke		
Ausdruck	Wert	Typ
sngMitarbeiter		String(0 to 2, 0 to 2)
sngMitarbeiter(0)		String(0 to 2)
sngMitarbeiter(0,0)	"Gahler"	String
sngMitarbeiter(0,1)	"Müller"	String
sngMitarbeiter(0,2)	"Sohm"	String
sngMitarbeiter(1)		String(0 to 2)
sngMitarbeiter(1,0)	"Thomas"	String
sngMitarbeiter(1,1)	"Karin"	String
sngMitarbeiter(1,2)	"Cordula"	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 vorhin 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
```

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

```

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

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

    intZähler = intZähler + 1
Loop
End Sub

```

In Listing 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

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.

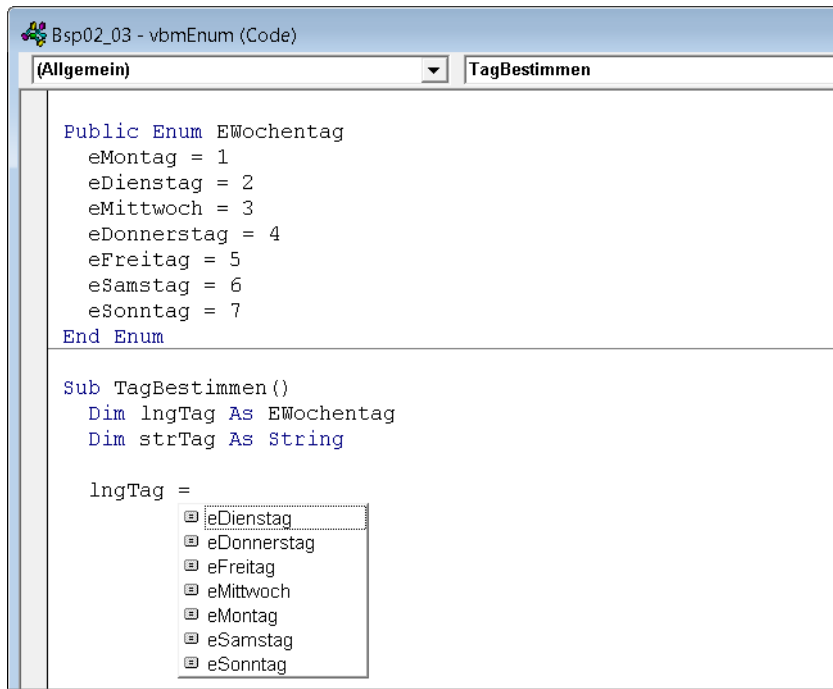
Aufzählungsvariablen können grundsätzlich nur auf Modulebene deklariert werden. Es besteht jedoch die Möglichkeit, die Gültigkeit der Aufzählung auf das entsprechende Modul zu beschränken. Dies erfolgt durch die Verwendung des Schlüsselworts `Private` innerhalb der Deklarationszeile.

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

    MsgBox CStr(lngHimmelsrichtung) & 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.")
```

Listing 2.12 Auswerten der eingegeben Himmelsrichtung, ein gelungenes Codebeispiel (Fortsetzung)

```
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
```

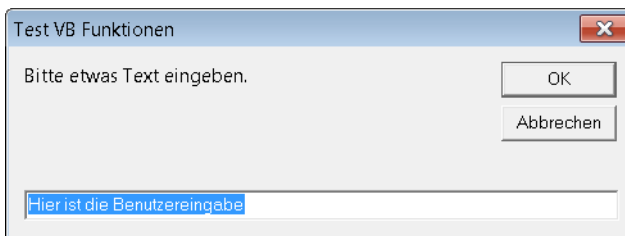
CD-ROM Die aufgeführten Codesequenzen finden Sie in der Beispieldatei *Bsp02_03.docm* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap02*.

Nützliche VBA-Funktionen

Mit dem Namen »Visual Basic for Applications« 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 15 beschrieben. Handelt es sich jedoch nur um sehr kurze oder einfache Eingaben, ist es weniger aufwendig, 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.

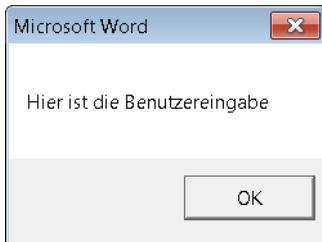
MsgBox In Listing 2.14 ist ein einfaches Beispiel dargestellt. In der Prozedur *InputBoxAnzeigen* werden zwei Variablen als Zeichenketten (String) deklariert. Der Aufforderungstext wird *strAufforderung* zugewiesen, den Wert für *strAntwort* gibt die InputBox in Abbildung 2.6 zurück. Der Text, den der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


Die Funktion *MsgBox* hat folgende Syntax:

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

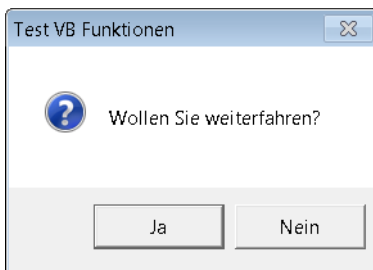
Auch hier ist *prompt* das einzige erforderliche Argument. Das Argument *buttons* ist aber sehr interessant und hilfreich. Damit lassen sich nicht nur verschiedene Schaltflächen anzeigen, um das Feedback des Benutzers auszuwerten. Sie können ihm auch den Wichtigkeitsgrad der Meldung mitteilen. In Tabelle 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

Tabelle 2.2 Werte für das Argument *buttons* der Funktion *MsgBox* (Fortsetzung)

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

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

```

Listing 2.15 Die *MsgBox* wird mit Schaltflächen, Symbolen und einem Titel ausgestattet (Fortsetzung)

```
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 voran stehende 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« ab Seite 85 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 Zeichenkettenfunktionen *Left*, *Right* und *Mid*

```

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

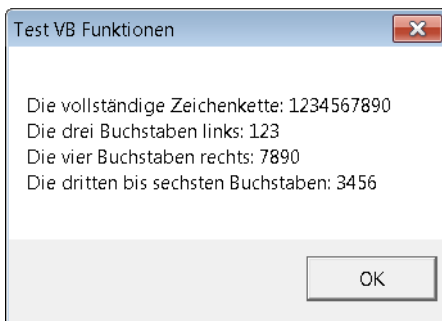
    strAufforderung = "Bitte etwas Text eingeben."
    strAntwort = InputBox (strAufforderung)
    MsgBox "Die vollständige Zeichenkette: " & strAntwort & vbCr & _
        "Die drei Buchstaben links: " & Left(strAntwort, 3) & _
        vbCr & "Die vier Buchstaben rechts: " & Right(strAntwort, 4) & _
        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
InstrRev
Replace

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

```

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

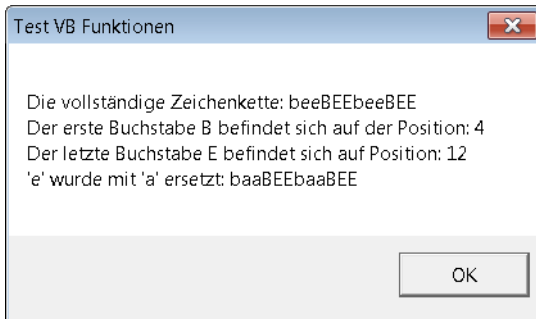
```

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 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.

Format

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-jjjj* 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
```

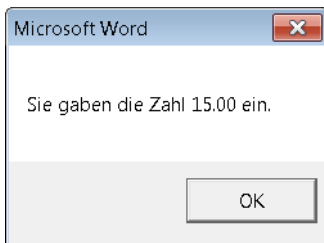
Bedingungen

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 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, sodass 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 " & _
```

Listing 2.19 Die Bedingung, ob die Benutzereingabe numerisch ist, wird geprüft (Fortsetzung)

```

        Format(strAntwort, "0.00") & " ein."
    End If
End Sub

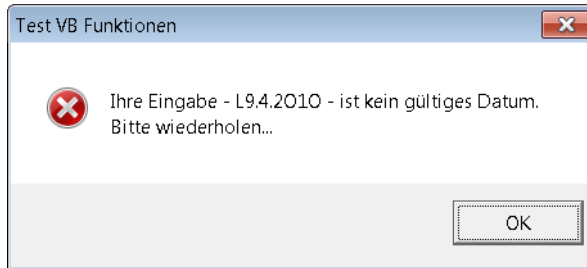
```

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



IsDate 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-2010 oder 2010/01/01. Wichtig zu wissen ist, dass die Funktion ein gültiges Datum aktiv forciert. 8.24.2010 wird ebenso akzeptiert wie 24.8.2010; IsDate versucht aus allen möglichen Kombinationen des angegebenen Werts ein gültiges Datum zu machen.

UCase 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." _

```

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. (Fortsetzung)

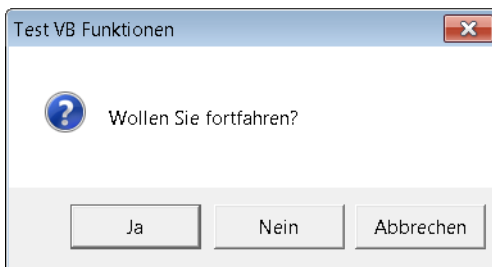
```
& vbCr & "Bitte wiederholen...", vbOKOnly + vbCritical, strMSGTITEL
End If
End Sub
```

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
```

Listing 2.21 Die Rückgabewerte der *MsgBox* mit *Select Case* auswerten (Fortsetzung)

```

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

```

Schleifen

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

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] + [Untr]` 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)
    Loop
    MsgBox "Programmende"
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 Zählervariable nach der Next-Anweisung schreiben. Dies hat auf den Codeablauf 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 durchlaufen

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

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

Standardmäßig wird Zähler bei jeder Durchführung der Schleife automatisch um den Faktor Eins erhöht. Mit der Anweisung Step kann ein anderer, auch negativer, Faktor bestimmt werden. Ein Beispiel dafür zeigt der Code in Listing 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 2 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
```

Listing 2.24 Dieses Beispiel löscht alle Absätze mit der Formatvorlage »Standard« (Fortsetzung)

```
para.Range.Delete
End If
Next lZaehler
End Sub
```

TIPP

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

CD-ROM

Die Beispieldatei für die Abschnitte »Nützliche VBA-Funktionen« (Seite 79), »Bedingungen« (Seite 85) und »Schleifen« (Seite 88) finden Sie unter dem Namen *Bsp02_04.docm* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap02*.

Compileranweisungen

Mittels einer Compileranweisung können VBA-Programme erstellt werden, die auf unterschiedlichen Systemumgebungen lauffähig sind.

Mit der »normalen« If...Then-Bedingung lassen sich Programmverzweigungen zur Laufzeit erzeugen. Mit einer Compileranweisung werden einzelne Programmzeilen oder ganze Programmbereiche von der Kompilierung ausgeschlossen.

Steht am Anfang einer Programmzeile eine Raute (#), 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.

Damit die Compileranweisung ausgewertet werden kann, müssen entsprechende Compilerkonstanten definiert werden. In der Tabelle 2.4 sind die von der Entwicklungsumgebung zur Verfügung gestellten Compilerkonstanten zusammengefasst. Bei Bedarf kann der Entwickler mittels einer Compileranweisung eigene Konstanten definieren. Diese speziellen Konstanten steuern innerhalb des Programms eine #If...Then-Anweisung.

```
#Const EARLYBINDING = False
```

Compilerkonstanten stehen nur einer Compileranweisung zur Verfügung und können nicht innerhalb einer »normalen« Programmzeile verwendet werden. Das Gleiche gilt umgekehrt natürlich auch. Eine »normale« Konstante kann innerhalb einer Compileranweisung nicht ausgewertet werden. Im Weiteren können Compilerkonstanten nur auf Modulebene deklariert werden und stehen nur in diesem Modul zur Verfügung.

Tabelle 2.4 Zusammenstellung der definierten Compilerkonstanten

Compilerkonstante	Bedeutung
Win16	Ist True , wenn es sich um eine 16-Bit-Entwicklungsumgebung handelt
Win32	Ist True , wenn es sich um eine 32-Bit-Entwicklungsumgebung handelt. Ist ebenfalls True , wenn es sich um eine 64-Bit-Entwicklungsumgebung handelt, dem diese ist zur 32-Bit-Entwicklungsumgebung kompatibel.

Tabelle 2.4 Zusammenstellung der definierten Compilerkonstanten (Fortsetzung)

Compilerkonstante	Bedeutung
Win64	Ist True , wenn es sich um eine 64-Bit-Entwicklungsumgebung handelt
VBA6	Ist in einer Entwicklungsumgebung mit Visual Basic for Application Version 6.0 auf True gesetzt. Diese Entwicklungsumgebung ist Teil des Lieferumfangs von Office 2000 bis einschließlich Office 2007.
VBA7	Ist in einer Entwicklungsumgebung mit Visual Basic for Application Version 7.0 auf True gesetzt. Diese Entwicklungsumgebung ist im Lieferumfang von Office 2010 enthalten, der ersten Office-Version, die das 64-Bit-Betriebssystem unterstützt. Damit kann Code zwischen 32- und 64-Bit-Anweisungen verzweigen, ohne Kompilierungsfehler auszulösen.
Mac	Auf einem Macintosh ist der Wert auf True gesetzt

Compileranweisung werden dann genutzt, wenn das gleiche Programm auf verschiedenen Plattformen lauffähig sein muss. So muss vor dem Kompilieren des Programms lediglich die entsprechende Konstante angepasst werden. Mit dieser Technik kann sichergestellt werden, dass beispielsweise das gleiche Projekt auf einem Windows-System sowie auf einem Macintosh lauffähig ist.

```
#If Mac Then
    strDateiname = Dir("MeinPfad", MacID("TEXT"))
#Else
    strDateiname = Dir("MeinPfad\*.txt")
#End If
MsgBox strDateiname
```

In unserem kleinen Beispiel würde dies bedeuten, dass die Zeilen zwischen dem `#If Mac Then` und dem `#Else` nur auf einem Macintosh-Rechner ausgeführt würden, die Zeilen zwischen dem `#Else` und dem `#End If` auf Windows-Systemen. Die nachfolgenden Programmzeilen würden auf beiden Plattformen ausgeführt werden. Auf diesem Wege könnte sichergestellt werden, dass auf beiden Systemen der Name der ersten Textdatei im entsprechenden Ordner ermittelt würde und anschließend in der Messagebox ausgegeben würde.

Wir Autoren empfehlen grundsätzlich, dass mit der ältesten Word-Version das VBA-Projekt entwickelt wird. So ist sichergestellt, dass dieses auf allen nachfolgenden Programm-Versionen ebenfalls funktionsfähig ist. Wird mit einer aktuellen Programm-Version entwickelt, besteht die Gefahr, dass neue Funktionen oder Objekte in das Programm einfließen, die in älteren Versionen von Word noch nicht enthalten waren. Als Beispiel könnte die `Split`-Funktion aufgeführt werden, welche in Word 97 nicht enthalten war, diese könnte nachgebaut werden und mit der `VBA6`- und `VBA7`-Konstante aktiviert werden.

```
#If VBA6 Or VBA7 Then
#Else
    Public Function Split(ByVal Expression As String, _
        ByVal Delimiter As String) As Variant
        'Hier die Funktionalität von Split() nachbilden
    End Function
#End If
```


Die nachgebaute Split-Funktion muss im #Else-Zweig der Compileranweisung stehen und steht somit innerhalb von Word 97 zur Verfügung. Bei allen anderen Word-Versionen wird automatisch die integrierte Funktion aufgerufen. Für diese nachgebaute Funktionen wird am besten ein eigenständiges Modul zur Verfügung gestellt.

Doch mit der ältesten Word-Version entwickeln und so auf die vielen Neuigkeiten der aktuellen Programm-Versionen verzichten, das ist nicht Erfolg versprechend. Als Beispiel kann das neue UndoRecord-Objekt erwähnt werden, welches in Office 2010 in den Objektkatalog aufgenommen wurde. Viele Entwickler haben auf diese Möglichkeit gewartet und sollten diese jetzt nicht nutzen, nur weil die Anwender ältere Word-Versionen im Einsatz haben. Dieses Dilemma lässt sich mit einer Compileranweisung ebenfalls umgehen. Alle Programmzeilen, welche mit diesem neuen Objekt arbeiten, müssen somit in eine #If...Then-Anweisung eingeschlossen werden.

```
#If VBA7 Then
    Dim undo As Word.UndoRecord
#End If
```

PROFITIPP

Eine weitere Möglichkeit, um Programmsequenzen innerhalb bestimmter Word-Versionen zu nutzen, besteht darin, dass die jeweiligen Programmzeilen in einem eigenständigen Modul stehen. Die Zeilen für die eine Version stehen in einem Modul, jene für die anderen Versionen getrennt in einem anderen Modul. Die zentrale Logik für das Programm steht in einem dritten Modul. Diese Vorgehensweise ist vor allem dann interessant, wenn keine interne Compileranweisung zur Verfügung steht, welche die Unterscheidung zwischen den einzelnen Word-Versionen ermöglicht (beispielsweise zwischen Word 2000 und 2002).

Der Compiler von Word prüft jeweils ein ganzes Modul, sobald ein erstes Mal eine Prozedur bzw. eine Funktion aufgerufen wird. Somit kann ein solches Modul Objekte enthalten, die in der im Einsatz stehenden Programmversion nicht enthalten sind.

Als Beispiel kann wiederum das UndoRecord-Objekt erwähnt. Dieses kann somit in einem Modul verwendet, in einem anderen Modul nicht verwendet werden.

```
If Application.Version = "14.0" Then
    Modul2010.TabelleErstellen    'inkl. Nutzung von UndoRecord-Objekt
Else
    Modul2007.TabelleErstellen    'ohne UndoRecord
End If
```

Diese Methode beinhaltet jedoch einen kleinen Nachteil. Wird im VB-Editor der Menüpunkt *Debuggen/Kompilieren von Projekt* angewählt, kann das Projekt in diesem Fall nur innerhalb von Word 2010 kompiliert werden. Bei allen anderen Programmversionen würde der Vorgang mit der Fehlermeldung »Benutzerdefinierter Typ nicht definiert« abgebrochen werden.

Im Kapitel 9 sind verschiedene Beispiele aufgeführt, welche eine #If...Then-Bedingung nutzen, damit das Programm unterschiedlich kompiliert wird.

Code im VB-Editor debuggen

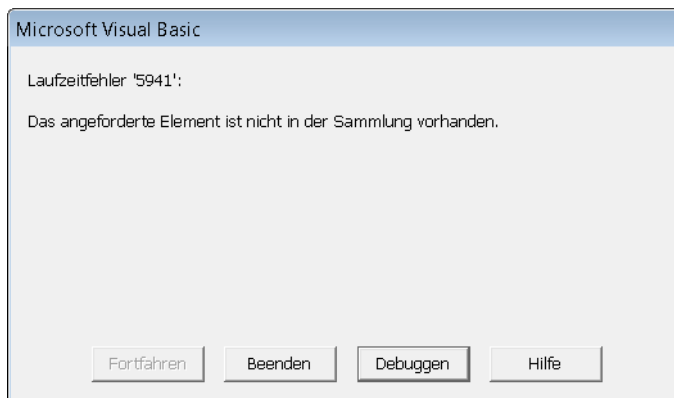
Vorausgesetzt, Sie haben Option `Explicit` eingeschaltet **und** verfügen über viel Erfahrung sowie eine übergroße Portion Glück, wird Ihr Code bei jedem ersten Mal ohne Fehler laufen. Ist Ihnen jedoch so viel Glück zuteil, wären Sie gut beraten, besser in einem Spielcasino anstatt vorm Bildschirm zu 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 Codeaufbaus 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 zu 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 zeigt eine Fehlermeldung, welche an die Diskussion über die `For...Next`-Anweisung im Abschnitt »Schleifen« ab Seite 88 anschließt. In den folgenden Zeilen soll aufgezeigt werden, wie die Ursache zu dieser Fehlermeldung ermittelt werden kann.

Abbildg. 2.15 Eine *For Each*-Schleife bricht mitten in der Verarbeitung ab



Folgendes Szenario ist angedacht. Ein Dokument enthält viele Textmarken, die gelöscht werden sollen. Es wurde entschieden, 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. Etwas konsterniert stellt er fest, dass eine so einfache Aufgabe bereits zu einem Laufzeitfehler führen kann.

Listing 2.25 Löschen aller Textmarken innerhalb eines Dokuments

```
Sub TextmarkenLöschen()  
    Dim doc As Word.Document  
    Dim i As Integer
```

Listing 2.25 Löschen aller Textmarken innerhalb eines Dokuments (Fortsetzung)

```

Set doc = Documents.Add(Template:=ThisDocument.FullName)

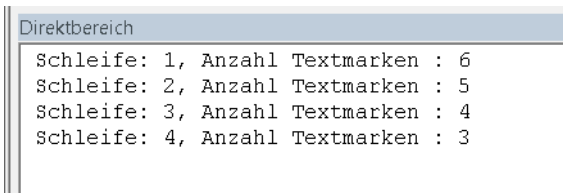
For i = 1 To doc.Bookmarks.Count
    doc.Bookmarks(i).Delete
Next i
End Sub

```

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 die Debug.Print-Anweisungen benutzen. Statt den Ablauf ständig zu unterbrechen, schreibt Debug.Print die Meldungen in den Direktbereich, wie in Abbildung 2.16 ersichtlich.

```
Debug.Print "Schleife: " & CStr(i) & ", Anzahl Textmarken : " & doc.Bookmarks.Count
```

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.

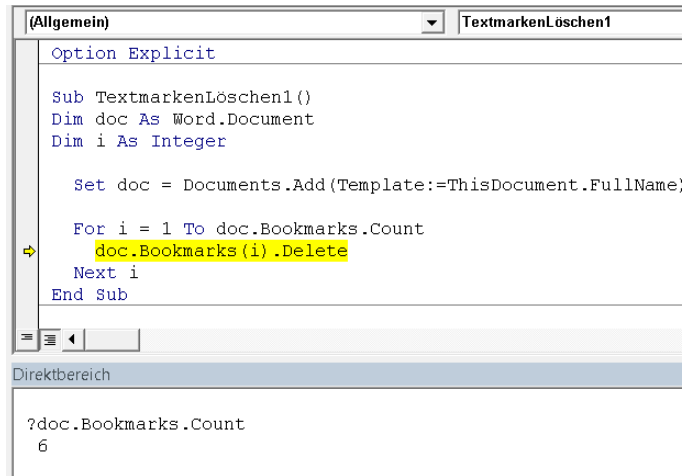
Der Befehl für die schrittweise Ausführung einer Prozedur befindet sich im Menü *Debuggen/Einzel-schritt*. Es wäre nun recht mühsam, für jede Codezeile das Menü zu öffnen und den Eintrag auszuwählen. Deshalb sollten Sie sich die Taste **F8** einprägen. Beim ersten Tastendruck wird die erste Codezeile (Sub TextmarkenLöschen2) 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 ?doc.Bookmarks.Count). 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 Codefenster weisen auf die nächste Anweisung hin.

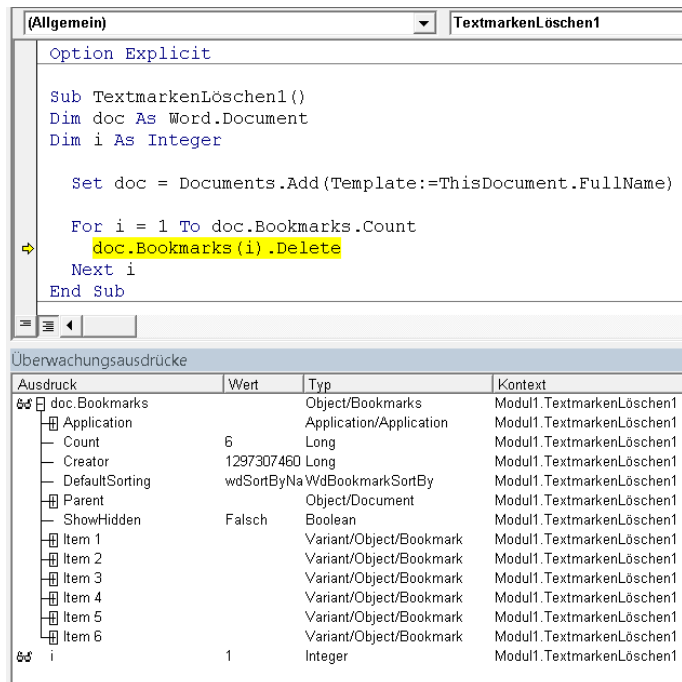


TIPP

Während der schrittweisen Ausführung dürfen Sie den Variablen auch andere Werte im Direktfenster zuweisen. Beispiel: 1Ende = 7 (aus Listing 2.24) und anschließend die -Taste drücken.


Abbildg. 2.18

Im Überwachungsbereich können mehrere Werte gleichzeitig bei jedem Schritt geprüft werden



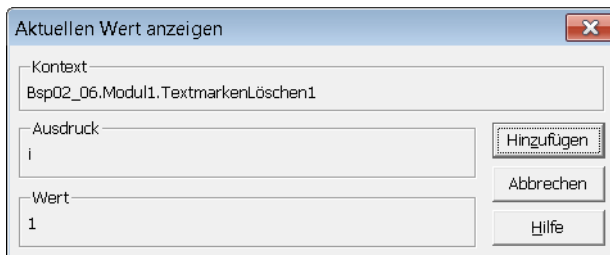
Der Überwachungsbereich

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 Codefenster und betätigen  + **F9**. Das Dialogfeld *Aktuellen Wert anzeigen* (siehe Abbildung 2.19) wird eingeblendet; klicken Sie auf *Hinzufügen*. (Ein Ausdruck darf vor sowie während der Codeausführung 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, hier bemerkt der Entwickler, dass der Schleifenzähler *i* beim vierten Durchgang einen größeren Wert annimmt als die Anzahl der im Dokument verbleibenden Textmarken. Daraus schließt er, dass zu Beginn des ersten Durchgangs bereits festgelegt wird, wie oft die Schleife durchlaufen wird.

Als Nächstes passt er die Verarbeitung innerhalb der *For...Next*-Schleife an. In Listing 2.26 wird grundsätzlich die erste Textmarke entfernt, denn nach dem Entfernen der ersten Textmarke wird die zweite Textmarke automatisch zur ersten Textmarke. Und so wird die Programmsequenz ohne Probleme verarbeitet.

Listing 2.26

Als Alternative in der *For...Next*-Schleife jeweils das erste Element löschen

```
Sub TextmarkenLöschen1()
    Dim doc As Word.Document
    Dim i As Integer

    Set doc = Documents.Add(Template:=ThisDocument.FullName)

    For i = 1 To doc.Bookmarks.Count
        doc.Bookmarks(1).Delete
    Next i
End Sub
```

Eine zweite Möglichkeit besteht darin, die Schleife rückwärts zu bearbeiten. Dieser Lösungsansatz wird in Listing 2.27 genutzt und das Programm arbeitet ebenfalls fehlerfrei.

Listing 2.27 Eine weitere Möglichkeit: Die *For...Next*-Schleife kann rückwärts durchlaufen werden

```
Sub TextmarkenLöschenStepMinus1()
    Dim doc As Word.Document
    Dim i As Integer

    Set doc = Documents.Add(Template:=ThisDocument.FullName)

    For i = doc.Bookmarks.Count To 1 Step -1
        doc.Bookmarks(i).Delete
    Next i
End Sub
```

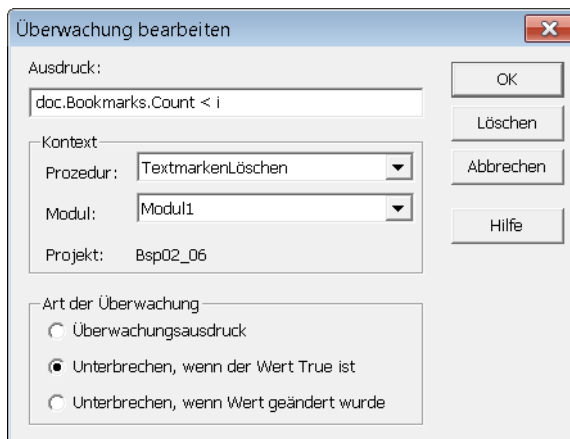
Bei einem umfänglicheren Programm spart es Zeit, wenn der Code bis zur Schleife ausgeführt und an diesem Punkt unterbrochen wird und somit auf der *For*-Zeile anhält. Um einen Haltepunkt zu setzen, klicken Sie in die Codezeile und betätigen dann die Taste **F9** (die Codezeile wird dunkelrot hervorgehoben). Drücken Sie **F5**, um die Ausführung zu starten; wenn sie beim Haltepunkt anhält, drücken Sie **F8** so lange, bis die Problemstelle gefunden wird.

Wären in unserem Dokument wirklich viele Textmarken vorhanden, wäre dem Entwickler nicht gedient, wenn die Ausführung zwar unterbrochen, die Schleife jedoch x-mal problemlos im Einzelschritt durchlaufen werden müsste, bis der Fehler auftritt. Er möchte den Code nur anhalten, wenn der Schleifenzähler einen größeren Wert hat als die Anzahl der verbleibenden Textmarken. Im Programmcode wird der gewünschte Ausdruck (`doc.Bookmarks.Count`) 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*.

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 `doc.Bookmarks(i).Delete` an. Jetzt kann die Umgebung im Debugger und das Dokument selber analysiert werden.

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.

CD-ROM Die Beispieldatei *Bsp02_05.docm* mit den Listings zu diesem Abschnitt finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap02*.

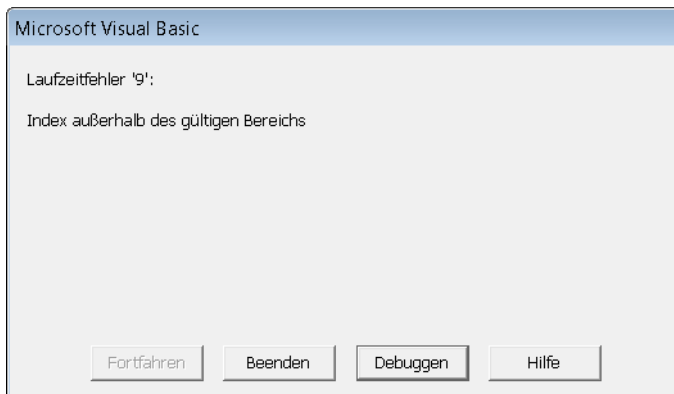
Fehlerbehandlung

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 bereit zu stehen.

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 Buchs 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 Problembeseitigung 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 2.28 sehen Sie ein vereinfachtes Beispiel. Einem Array (Datenfeld) werden vier Elemente zugewiesen. Danach wird der Benutzer aufgefordert, Anfangs- und Endzahlen einzugeben. Die Schleife wird, auf diesen Angaben basierend, ausgeführt und die Elemente des Arrays werden in den Direktbereich ausgegeben.

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

Listing 2.28 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
```


Listing 2.28 Die Grundrisse der Fehlerbehandlung: *On Error GoTo*-Anweisung, mit der Zeilenmarke *Fehlerbehandlung* (Fortsetzung)

```
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 2.29.

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 2.29 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:
    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 FehlerBehandlung2" & vbCrLf _
        & Err.Description & vbCrLf & "Fehlernummer: " & Err.Number, _
        vbCritical + vbOKOnly
    Resume Start
End Sub
```

Die Meldung der *MsgBox* ist noch verbesserungswürdig. Zudem müssen wir zwischen verschiedenen Fehlern unterscheiden können. Unser Beispiel enthält zwei verschiedene Arten von Fehlern: *Index außerhalb des gültigen Bereichs* (mit der Fehlernummer 9) sowie *Typen unverträglich* (mit der Fehler-

nummer 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 2.30 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 2.30

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:
    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 FehlerBehandlung3" & 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 9, 10 und 11 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 2.31 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 2.31 *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." _
            & vbCr & Err.Description & vbCr & "Fehlernummer: " & Err.Number
    End If
    On Error GoTo 0
    xlApp.Visible = True
End Sub
```

Err-
Objekt

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

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

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

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

Detaillierte Informationen enthält die VBA-Hilfe zum Thema.

Das Prinzip wird veranschaulicht in Listing 2.32. 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 Anwendungsfehlnummern vermieden.

Listing 2.32 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")

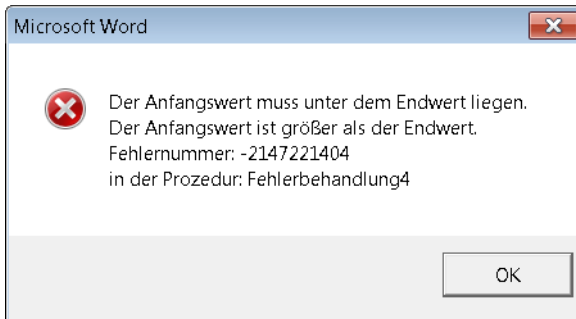
Start:
    lAnfang = CLng(InputBox("Bitte die Anfangszahl eingeben"))
    lEnde = CLng(InputBox("Bitte die Endezahl eingeben"))
    If lAnfang > lEnde Then
        Err.Raise lEINGABEFEHLER, "Fehlerbehandlung4", _
            "Der Anfangswert ist größer als der Endwert."
    End If

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

EndPunkt:
    Exit Sub

Fehlerbehandlung:
    Select Case Err.Number
        Case 9, 13
            MsgBox "Sie müssen eine Zahl zwischen 0 und 3 eingeben.", vbInformation + vbOKOnly
            Resume Start
        Case lEINGABEFEHLER
            MsgBox "Der Anfangswert muss unter dem Endwert liegen." _
                & vbCr & Err.Description & vbCr & "Fehlernummer: " & Err.Number _
                & vbCr & "in der Prozedur: " & Err.Source, vbCritical + vbOKOnly
            Resume Start
        Case Else
            MsgBox "Fehler in der Prozedur FehlerBehandlung4" & 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).

CD-ROM

Die Beispieldatei *Bsp02_06.docm* 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, wird eine leere Zeichenkette ("") zurückgeliefert. Die Dir-Funktion unterstützt eine Suche mittels Platzhalterzeichen.

Im Listing 2.33 wird dieser Umstand genutzt, um eine Liste aller vorhandenen Dokumente im aktuellen Verzeichnis auszugeben.

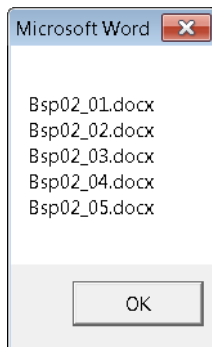
Listing 2.33 Auflisten aller Dateinamen im aktuellen Verzeichnis

```
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 Ordnernamen, die andere den Dateinamen), muss sichergestellt sein, dass das Resultat dieser Verknüpfung ein gültiger Dateiname ist:

```
strDateiname = strPfad & strDateiname
```

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

In Listing 2.34 wird mittels einer eigenen Funktion diesem Umstand Rechnung getragen. Die Funktion hängt bei Bedarf den fehlenden Backslash an den Ordnernamen an.

Listing 2.34 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\Office14"
    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

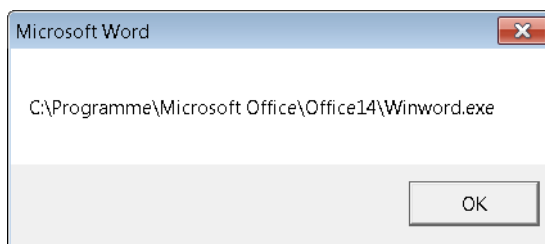
    fktPfadInklBackslash = strPfad
End Function
```

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« ab Seite 106 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 2.33 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 2.35 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 2.35

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

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

Wie aus Tabelle 2.5 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 2.5 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« ab Seite 106 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« ab Seite 109 erläutert, kann die Verwendung der `Dir`-Funktion zu Problemen führen. Um diese zu umgehen, wird der Programmcode aus Listing 2.35 verwendet und leicht angepasst.

In Listing 2.36 wird ebenfalls ohne die Verwendung der `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 2.36 Anhand der Dateiattribute prüfen, ob ein Ordner überhaupt vorhanden ist

```
Sub PrüfenObOrdnerVorhandenIst()
    MsgBox fktExistiertOrdner("C:\Programme")
    MsgBox fktExistiertOrdner("C:\")
End Sub
```

Listing 2.36 Anhand der Dateiattribute prüfen, ob ein Ordner überhaupt vorhanden ist (Fortsetzung)

```

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 2.5 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 2.37 genutzt, um die entsprechende Prüfung durchzuführen.

Listing 2.37 Prüfen, ob ein exklusiver Zugriff auf eine Datei möglich ist

```

Sub PrüfenObDateiImZugriffIst()
    MsgBox fktIstDateiBereitsGeöffnet("C:\BOOTLOG.TXT")
    MsgBox fktIstDateiBereitsGeöffnet(ThisDocument.FullName)
End Sub

```

Listing 2.37 Prüfen, ob ein exklusiver Zugriff auf eine Datei möglich ist (Fortsetzung)

```
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 2.38 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 2.35 und Listing 2.37 bewerkstelligt.

Listing 2.38 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

Function fktDateiLöschen( _
    ByVal strDateiname As String) _
    As Boolean

    Const intATTR_NODELETE = vbReadOnly + vbHidden
    Dim bFlag As Boolean

    'Ist die Datei vorhanden
```

Listing 2.38 Datei auf dem Datenträger löschen und mögliche Fehler bearbeiten (Fortsetzung)

```

    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 2.36 integriert und detailliert beschrieben.

Letztes Speicherdatum einer Datei ermitteln

FileDate-
Time

Um das letzte Speicherdatum einer Datei bzw. eines Ordners auf dem Datenträger zu ermitteln, steht die `FileDateTime`-Funktion zur Verfügung. Auch in diesem Fall ist es sinnvoll, zuerst das Umfeld dahingehend zu überprüfen, ob der Versuch, das Datum des Verzeichniseintrags zu ermitteln, Erfolg haben könnte.

TIPP Wir empfehlen grundsätzlich, eine Sammlung von einzelnen Funktionen und Prozeduren mit genau definiertem Funktionsumfang (beispielsweise eine Datei löschen, das Datum einer Datei ermitteln usw.) anzulegen. So wird vermieden, dass Sie sich bei jedem Aufruf des Original-VBA-Befehls auch noch um die möglichen Fehlerfälle kümmern müssen. Dies ist nicht mehr nötig, da dies zentral innerhalb der einzelnen Funktion aus der Sammlung bereits erfolgt ist.

In Listing 2.39 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 2.39 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

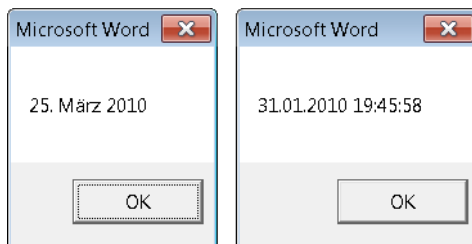
```

HINWEIS

Das aktuelle Programmbeispiel baut auf die beiden bereits vorgestellten Funktionen aus Listing 2.35 und Listing 2.36 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 2.40 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 2.40 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 2.35 auf. Damit das Beispiel ausgeführt werden kann, muss diese Funktion im Zugriff sein.

Das gewünschte Ausgabeformat der Funktion *fktDateigrösseErmitteln* kann als zusätzlicher Parameter übergeben werden. Wird kein Wert eingetragen, wird der definierte Standardwert übernommen. Dies wird durch die Angabe des Schlüsselworts `Optional` und der zusätzlichen Zuweisung eines Werts innerhalb der Deklarationszeile erreicht:

```
Optional ByVal strFormat As String = "###,###,###,##0"
```

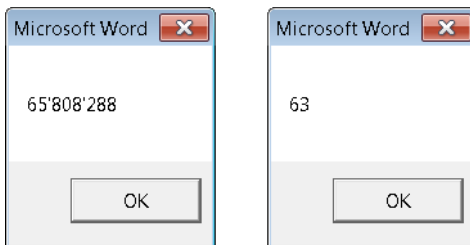
Als weitere Besonderheit der Funktion *fmtDateiGrößeErmitteln* kann die gewünschte Maßeinheit für den Rückgabewert übergeben werden. Dazu wurde auf Modulebene eine entsprechende Aufzählung deklariert. Die zugewiesenen Werte entsprechen gleichzeitig den Umrechnungsfaktoren:

```
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



CD-ROM

Die aufgeführten Codesequenzen finden Sie in der Beispieldatei *Bsp02_07.docm* 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 Standarddatentypen, die Sichtbarkeit derselben und die Übergabe an Prozeduren vorgestellt (Seite 56 ff.)
- In einem zweiten Abschnitt wurden die Konstanten (Seite 69 ff.) und die benutzerdefinierten Typen (Seite 71 ff.) erläutert
- Ein weiterer Abschnitt widmete sich nützlichen VBA-Funktionen (Seite 79 ff.), Programmbedingungen und Schleifen (Seite 85 ff.)
- Ein besonderer Augenmerk wurde auf die bedingte Kompilierung mittels Compileranweisung gelegt (Seite 91 ff.)
- Ebenso wurde aufgezeigt, wie der Programmcode mit dem Debugger untersucht (Seite 94 ff.) und eventuelle Programmfehler mittels einer Fehlerbehandlung aufgefangen werden können (Seite 99 ff.)
- Praktische Beispiele zum Dateisystem (Seite 105 ff.) runden das Erlernte in diesem Kapitel ab

Kapitel 3

Windows-APIs in VBA nutzen

In diesem Kapitel:

Aufbau der API-Funktionen	118
API-Funktionen unter Office-Versionen mit 64 Bit und 32 Bit verwenden	121
Pfade kombinieren und abschließen	123
Datei über die Dateieindung ausführen	127
Benutzerformular transparent darstellen	131
Auf Registry-Einträge zugreifen	132
Auf INI-Dateien zugreifen	139
Name des angemeldeten Benutzers ermitteln	144
Verarbeitung für eine bestimmte Zeit unterbrechen	146
Wave-Datei abspielen	147
Tastenstatus abfragen	150
Zusammenfassung	156

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.

CD-ROM

In der Datei *Bsp03_1.docm* auf der CD-ROM zum Buch 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 »Auf INI-Dateien« ab Seite 139.

Aufbau der API-Funktionen

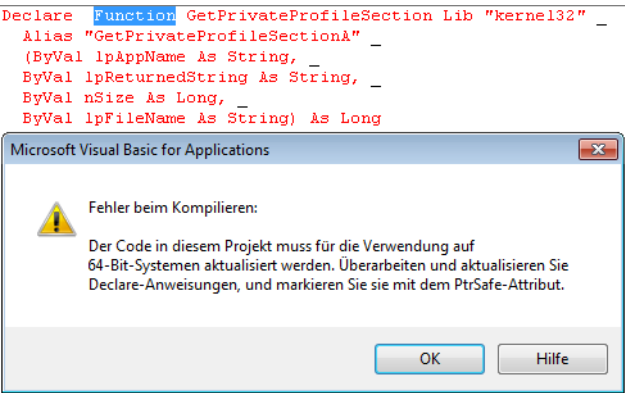
Jede API-Funktion besitzt den prinzipiell gleichen Aufbau, der sich hauptsächlich aus dem Namen der Funktion und der Bibliothek, in der sich die Funktion befindet, zusammensetzt. Zusätzlich können einzelne notwendige Argumente oder auch ganze Argumentlisten für den Aufruf angegeben werden.



Mit Office 2010 hat Microsoft erstmals neben den bisherigen 32-Bit-Versionen eine 64-Bit-Version von Office auf den Markt gebracht, die ausschließlich auf der 64-Bit-Version von Windows 7 und Windows Server 2008 R2 installiert werden kann. Diese Unterteilung in 32-Bit- und 64-Bit-Versionen hat einschneidende Auswirkungen auf den Einsatz von APIs unter Office: Erstmals laufen keine API-Funktionen mehr direkt unter Office 2010 64 Bit, sondern müssen angepasst werden.

Wenn Sie versuchen, den bisher funktionierenden Code unter Word 64 Bit auszuführen, erhalten Sie einen Compilerfehler, der darauf hinweist, dass der Code für die Verwendung auf 64-Bit-System angepasst werden muss. Die Meldung liefert auch gleich die Lösung für die Anpassung: Verwenden Sie in der Declare-Anweisung das `PtrSafe`-Attribut.

Abbildg. 3.1 Fehlermeldung beim Versuch, eine API-Funktion unter Word 64 Bit auszuführen



Alle Declare-Anweisungen müssen nun das Schlüsselwort `PtrSafe` enthalten, damit sie unter den 64-Bit-Versionen von Microsoft Office ausgeführt werden können: Das Schlüsselwort `PtrSafe` gibt dabei an, dass eine Declare-Anweisung unter den 64-Bit-Versionen von Microsoft Office sicher ausgeführt werden kann.

Die grundlegende Syntax einer API-Funktion sieht nun wie folgt aus (die einzelnen Teile werden in Tabelle 3.1 erläutert):

```
[Public|Private] Declare [PtrSafe] 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 Public .
Declare	Notwendiges Schlüsselwort, mit dem der Zugriff auf eine API- oder DLL-Funktion definiert wird
PtrSafe	Notwendiges Attribut, wenn die API-Funktion auf einem 64-Bit-System sicher ausgeführt werden soll
Sub Function	Legt den Typ der API-Funktion fest. Wenn die API-Funktion einen Rückgabewert liefert, sollte sie als Function deklariert werden; andernfalls kann sie als Sub deklariert werden.
Name	Der Zugriffsname der Funktion
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

Tabelle 3.1 Aufbau der API-Funktion (Fortsetzung)

Parameter	Beschreibung
"AliasName"	Wird das Schlüsselwort Alias 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 Long zurück.

WICHTIG

Die Ergänzung der **Declare**-Anweisung mit dem **PtrSafe**-Attribut garantiert aber nicht die Funktionsfähigkeit einer API-Funktion. Um das etwas besser zu verstehen, muss man sich mit dem Unterschied zwischen 32-Bit- und 64-Bit-System auseinandersetzen. Da dies aber an dieser Stelle zu weit führen würde, soll nur ein Hauptmerkmal angesprochen werden: die unterschiedliche Speicheradressierung.

Während ein 32-Bit-System einen Speicherbereich von maximal 4 GB (2³² Bit) adressieren und nutzen kann, umfasst bei 64-Bit-Systemen der Adressraum 2⁶⁴ Bit, also theoretische 16 EiB (Exbibyte). Unter Windows 7 (Ultimate) stehen rund 192 GB zur Verfügung.

Dieser vergrößerte Adressraum hat nun direkte Auswirkungen auf alle API-Funktionen, die entweder über einen Handle oder Pointer auf Speicherbereiche zugreifen. Dadurch ändert sich z.B. der Datentyp **Long** (ein 4-Byte-, also 32-Bit-Datentyp) auf den Datentyp **LongLong** (64-Bit-8-Byte-Datentypen).

Weitere Informationen finden Sie auch in der VBA-Hilfe unter dem Stichpunkt »64-Bit-Visual Basic für Applikationen – Überblick« und »PtrSafe«.

Microsoft stellt weitere Informationen zur Verwendung von 32-Bit-API-Funktionen unter 64-Bit-Systemen unter folgendem Link zur Verfügung:

<http://www.microsoft.com/downloads/details.aspx?displaylang=en&FamilyID=035b72a5-eef9-4baf-8dbc-63fbd2dd982b>

Nach der Installation des Downloads finden Sie die Dateien mit weiteren Informationen im Ordner *C:\Office 2010 Developer Resources\Documents\Office2010Win32API_PtrSafe*

Dazu gehört auch die Datei *Win32API_PtrSafe.txt* mit »Declare statements for Visual Basic for Applications and Microsoft Office 2010«, also mit Deklarationsangaben API-Funktionen und Type-Deklarationen für Office 2010 64 Bit.

Weitere Informationen zur Kompatibilität zwischen den 32-Bit- und 64-Bit-Versionen von Office 2010 erhalten Sie im Artikel »Compatibility Between the 32-bit and 64-bit Versions of Office 2010«, die Sie unter folgendem Link bei Microsoft finden:

[http://msdn.microsoft.com/en-us/library/ee691831\(office.14\).aspx](http://msdn.microsoft.com/en-us/library/ee691831(office.14).aspx)

API-Funktionen unter Office-Versionen mit 64 Bit und 32 Bit verwenden

Da API-Funktionen unter 64-Bit-Office-Versionen nur mit Anpassungen und dann unter 32-Bit-Office-Versionen nicht mehr ausgeführt werden können, wird mit Office 2010 eine neue VBA-Version ausgeliefert: VBA7. In dieser VBA-Version wurden die Compilerkonstanten erweitert und bringen nun eigene Compilerkonstanten für die 64-Bit- und 32-Bit-Versionen von Office mit.

Die neuen Compilerkonstanten sind in Tabelle 3.2 aufgeführt.

Tabelle 3.2 Neue Compilerkonstanten in Word 2010

Konstante	Beschreibung
Win64	Gibt an, dass die Entwicklungsumgebung 64-Bit-kompatibel ist bzw. nicht kompatibel ist
Vba7	Gibt an, dass die Entwicklungsumgebung mit Visual Basic for Applications, Version 7.0, kompatibel ist bzw. nicht kompatibel ist

Mithilfe dieser zusätzlichen Compilerkonstanten lassen sich nun Compileranweisungen erstellen, die bewirken, dass API-Funktionen für beide Office-Versionen in einem gemeinsamen Projekt enthalten sein können. Dazu werden unter der jeweiligen Version nur die Bedingungen kompiliert, für die die angegebenen Bedingungen erfüllt sind.

Listing 3.1 Compileranweisung zur bedingten Kompilierung für 64-Bit- und 32-Bit-Office-Versionen

```
#If Win64 Then
    Private Declare PtrSafe Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Long)
#Else
    Private Declare Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Long)
#End If
```

WICHTIG

Damit die Compileranweisung unter 64-Bit-Office-Versionen ohne Fehler kompiliert wird, muss die Reihenfolge der Compilerkonstanten beachtet werden: Zuerst muss die Compilerkonstante Win64 abgefragt werden und erst dann die Compilerkonstante Win32. Denn die Compilerkonstante Win32 liefert sowohl unter 32-Bit- als auch unter 64-Bit-Office-Versionen True zurück, da die 32-Bit-Version auch unter Windows 7 64 Bit installierbar ist.

Listing 3.2 Reihenfolge der Compileranweisung

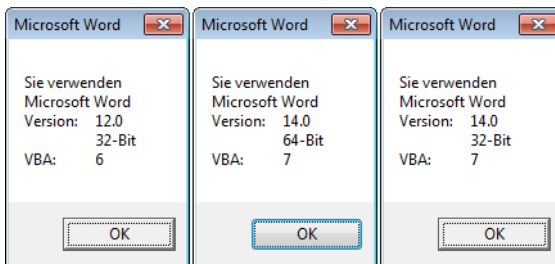
```
#If Win64 Then
    ' Win64=true, Win32=true, Win16=false
#ElseIf Win32 Then
    ' Win32=true, Win16=false
#Else
    ' Win16=true
#End If
```

Über die Compilerkonstanten für die VBA-Versionen (Vba7 und Vba6) kann dann auch zwischen Office 2010 und älteren Office-Versionen, die die VBA-Version 6 verwenden, unterschieden werden, sodass insgesamt eine gute Differenzierung zwischen den verschiedenen Office-Versionen möglich ist.

Listing 3.3 Bedingte Kompilierung zur Ermittlung der Word-Informationen unterschiedlicher Installationen

```
Function VBAVersion() As String
Dim sVer As String
sVer = "Sie verwenden " & vbCrLf & Application.Name & vbCrLf & _
"Version:" & vbTab & Application.Version & vbCrLf
#If Win64 Then
sVer = sVer & vbTab & "64-Bit" & vbCrLf
#If VBA7 Then
sVer = sVer & "VBA:" & vbTab & "7"
#Else
sVer = sVer & "VBA:" & vbTab & "6"
#End If
#ElseIf Win32 Then
sVer = sVer & vbTab & "32-Bit" & vbCrLf
#If VBA7 Then
sVer = sVer & "VBA:" & vbTab & "7"
#Else
sVer = sVer & "VBA:" & vbTab & "6"
#End If
#Else
sVer = sVer & vbTab & "16-Bit" & vbCrLf
sVer = sVer & "VBA:" & vbTab & "5"
#End If
VBAVersion = sVer
End Function
```

Abbildg. 3.2 Ausgabe der bedingten Kompilierung von Word 2007 (32 Bit unter Windows 7 64 Bit), Word 2010 (64 Bit unter Windows 7 64 Bit) und Word 2010 (32 Bit unter Windows 7 64 Bit)



CD-ROM

Alle Beispiele in der Datei *Bsp03_1.docm* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap03* sind mit den Compileranweisungen für beide Office-Versionen versehen.

Pfade kombinieren und abschließen

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.

Beispielsweise liefert in Listing 3.4 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 3.4 Unterschiedliche Rückgabewerte der *Dir*-Funktion

```
Debug.Print Dir("C:\temp",vbDirectory )
> temp
Debug.Print Dir("C:\temp\",vbDirectory )
> .
```

Ein weiteres Problem tritt bei der Kombination von Pfad und Dateinamen bzw. bei zwei Ordernamen auf: Es muss sichergestellt werden, dass der Gesamtpfad korrekt gesetzte Backslash beinhaltet. So sollten normalerweise beide Pfadteile auf beginnende bzw. endende Backslash überprüft werden, bevor sie kombiniert werden können, da andernfalls Fehler bezüglich unbekannter oder falscher Pfade auftreten können.

Listing 3.5 Fehlerhafte Auswertung bei fehlendem oder doppeltem Backslash

```
Debug.Print "C:\temp" & "Test.docx"
> C:\tempTest.docx
Debug.Print "C:\temp\" & "\Test.docx"
> C:\temp\\Test.docx
```

Diese Problematik ließe sich zwar durch eine Abfrage des letzten Zeichens prüfen und für zukünftige Verwendung in Form einer Funktion schreiben. Aber warum nicht auf vorhandene Wege zurückgreifen? So steht unter Windows ein Satz von API-Funktionen zur Verfügung, die sich um die korrekte Kombination und Erstellung von Pfaden kümmern.

Pfade abschließen

Mithilfe der API-Funktion *PathAddBackslash* wird automatisch ein Backslash an einen Pfad angefügt, sofern noch keiner vorhanden ist. Die Deklaration lautet wie folgt (die Erklärung befindet sich in Tabelle 3.3):

```
#If Win64 Then
Declare PtrSafe Function PathAddBackslash Lib "shlwapi.dll" Alias "PathAddBackslashA" _
    (ByVal pszPath As String) As Long
#Else
Declare Function PathAddBackslash Lib "shlwapi.dll" Alias "PathAddBackslashA" _
    (ByVal pszPath As String) As Long
#End If
```

Tabelle 3.3 Parameter der API-Funktion *PathAddBackslash*

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

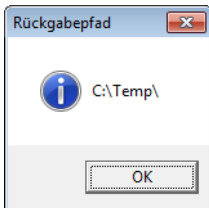
Diese Funktion erwartet als Ein- und Ausgabeparameter eine Variable vom Typ String. Da der erweiterte Pfad in derselben Variablen wie der Eingangsparameter ausgegeben wird, können Sie den Pfad nicht direkt angeben, sondern müssen diesen über eine Variable (Buffer) festlegen. Dieser Buffer muss groß genug 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 3.6 veranschaulicht dieses Prinzip.

Listing 3.6 Abschließen des Pfads 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. 3.3 Ausgabe des abgeschlossenen Pfads



Pfade kombinieren

Um zwei Pfade miteinander zu kombinieren, können Sie die API-Funktion *PathCombine* verwenden. Die Deklaration lautet wie folgt (die Erklärung befindet sich in Tabelle 3.4):

```
#If Win64 Then
Declare PtrSafe Function PathCombine Lib "shlwapi.dll" Alias "PathCombineA" _
    (ByVal szDest As String, _
    ByVal lpszDir As String, _
    ByVal lpszFile As String) As Long
#Else
Declare Function PathCombine Lib "shlwapi.dll" Alias "PathCombineA" _
    (ByVal szDest As String, _
    ByVal lpszDir As String, _
    ByVal lpszFile As String) As Long
#End If
```


Tabelle 3.4 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

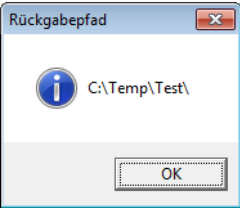
Diese Funktion erwartet neben den beiden Pfadangaben *lpszDir* und *lpszFile*, die kombiniert werden sollen, eine Variable *szDest*, in die das Ergebnis ausgegeben wird.

Die Buffervariable *szDest* muss dabei wieder so groß angelegt werden, dass der kombinierte Pfad auf jeden Fall hineinpasst, wie Listing 3.7 veranschaulicht.

Listing 3.7 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. 3.4 Ausgabe des zusammengesetzten Pfads



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 mit Dateierweiterung an Pfad anhängen

Zwei weitere nützliche API-Funktionen sind *PathAppend* und *PathAddExtension*. Mit diesen Funktionen können Sie einen Dateinamen mit Dateierweiterung an einen Pfad anhängen. Die Deklaration lautet wie folgt (die Erklärung befindet sich in Tabelle 3.5):

```
#If Win64 Then
Declare PtrSafe Function PathAppend Lib "shlwapi.dll" Alias "PathAppendA" _
    (ByVal pszPath As String, ByVal pMore As String) As Long
Declare PtrSafe Function PathAddExtension Lib "shlwapi.dll" Alias "PathAddExtensionA" ( _
    ByVal pszPath As String, _
    ByVal pszExt As String ) As Long
#Else
Declare Function PathAppend Lib "shlwapi.dll" Alias "PathAppendA" _
    (ByVal pszPath As String, ByVal pMore As String) As Long
Declare Function PathAddExtension Lib "shlwapi.dll" Alias "PathAddExtensionA" ( _
    ByVal pszPath As String, _
    ByVal pszExt As String ) As Long
#End If
```

Tabelle 3.5 Parameter der API-Funktion *PathAppend*

Parameter	Bedeutung	Ein-/Ausgabe
pszPath	Variable mit dem Basispfad	Ein/Aus
pMore	Variable mit dem Dateinamen	Ein

Dies Funktion PathAppend erwartet neben dem Pfad pszPath den Dateinamen pMore; der kombinierte Pfad mit der Datei wird dabei in der Variable pszPath wieder ausgegeben. Die Buffervariable pszPath muss dabei wieder so groß angelegt werden, dass der kombinierte Pfad auf jeden Fall hineinpasst, wie Listing 3.8 veranschaulicht.

Tabelle 3.6 Parameter der API-Funktion *PathAddExtension*

Parameter	Bedeutung	Ein-/Ausgabe
pszPath	Variable mit dem Basispfad	Ein/Aus
pszExt	Variable mit der Dateierweiterung	Ein

Diese Funktion PathAddExtension 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 3.8 veranschaulicht.

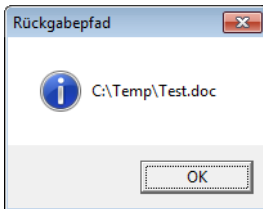
Listing 3.8 Anhängen eines Dateinamens mit Dateierweiterung an einen Pfad mittels *PathAppend* und *PathAddExtension*

```
Sub subPathAddExtension()
' Dateierweiterung an Dateinamen anhängen
Dim strFile As String
Dim strExt As String
' Dateinamen in Buffer schreiben und Buffer vergrößern
strFile = "Test" & String(254, vbNullChar)
strExt = ".doc"
' Erweiterung anhängen
PathAddExtension strFile, strExt
'
' Datei mit Pfad kombinieren
Dim strPath As String
```

Listing 3.8 Anhängen eines Dateinamens mit Dateierweiterung an einen Pfad mittels *PathAppend* und *PathAddExtension* (Fortsetzung)

```
' Basispfad in Buffer schreiben und Buffer vergrößern
strPath = "C:\Temp\" & String(254, vbNullChar)
PathAppend strPath, strFile
MsgBox strPath, vbInformation, "Rückgabepfad"
End Sub
```

Abbildg. 3.5 Ausgabe des vollständigen Dateinamens mit Dateierweiterung



CD-ROM

Die Beispiele zu diesem Abschnitt finden Sie auf der CD-ROM in der Beispieldatei `\Beispiele\Kap03\Bsp03_1.docm` im Modul *modPathAPIs*.

Datei über die Dateieindung ausführen

Wenn Sie eine Datei oder ein Programm starten möchten, klicken Sie im Windows-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 3.9, die Erläuterung der Parameter ist in Tabelle 3.7 enthalten.

Listing 3.9 Deklaration der API-Funktion *ShellExecute*

```
#If Win64
Declare PtrSafe Function ShellExecute Lib "shell32.dll" Alias "ShellExecuteA" ( _
    ByVal hwnd As LongPtr, _
    ByVal lpOperation As String, _
    ByVal lpFile As String, _
    ByVal lpParameters As String, _
    ByVal lpDirectory As String, _
    ByVal ShowTypeEnum As Long _
) As LongPtr
#Else

Declare Function ShellExecute Lib "shell32.dll" Alias "ShellExecuteA" ( _
    ByVal hwnd As Long, _
    ByVal lpOperation As String, _
    ByVal lpFile As String, _
    ByVal lpParameters As String, _
```

Listing 3.9 Deklaration der API-Funktion *ShellExecute* (Fortsetzung)

```

    ByVal lpDirectory As String, _
    ByVal ShowTypeEnum As Long _
) As Long
#End If
Private Enum ShowTypeEnum
    SW_HIDE = 0
    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 3.7 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 Null sein.	Ein
lpDirectory	Variable mit einem Standardverzeichnis; kann durch einen Leerstring ersetzt werden	Ein
ShowTypeEnum	Ein Wert aus der ShowTypeEnum -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 3.10 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 3.10 Beispiel zur Verwendung von Programmparametern

```

Sub subShellExecute5()
    Dim strTemp As String
    strTemp = Environ$("Temp") & "\Test.txt"
    fktShellExecute Environ$("COMSPEC"), "/C set Username > " & strTemp
    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.

Es gibt aber Anwendungsfälle, wo API-Funktionen den Handle eines Benutzerformulars benötigen. Ein Beispiel dazu finden Sie im Abschnitt »Benutzerformular transparent darstellen«.

Listing 3.11 und Listing 3.12 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 3.11 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 3.12 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 3.13 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 CD-ROM zum Buch.

Listing 3.13 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
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 Dateieindung zugeordnet."
Case SE_ERR_OOM
    strMSG = "Nicht genügend Speicher zum Beenden des Vorgangs verfügbar."
Case SE_ERR_PNF
```

Listing 3.13 Kapselung der API-Funktion *ShellExecute* in eine Funktion mit Fehlerausgabe (Fortsetzung)

```

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

```

CD-ROM Die Beispiele zu diesem Abschnitt finden Sie in der Beispieldatei `\Beispiele\Kap03\Bsp03_1.docm` im Modul `modShellExecuteAPI`.

Benutzerformular transparent darstellen

Es gibt auch unter VBA Anwendungsfälle, in denen das Fensterhandle (hWnd) eines Benutzerformulars (UserForm) benötigt wird. So lässt sich z.B. auch in VBA eine UserForm transparent darstellen.

Zur Ermittlung des Fensterhandles kann die API-Funktion `FindWindow` verwendet werden.

Listing 3.14 Den Fensterhandle einer *UserForm* ermitteln

```

#If Win64 Then
Private Declare PtrSafe Function FindWindow Lib "user32" Alias "FindWindowA" _
    (ByVal lpClassName As String, ByVal lpWindowName As String) As Long
#Else
Private Declare Function FindWindow Lib "user32" Alias "FindWindowA" _
    (ByVal lpClassName As String, ByVal lpWindowName As String) As Long
#End If

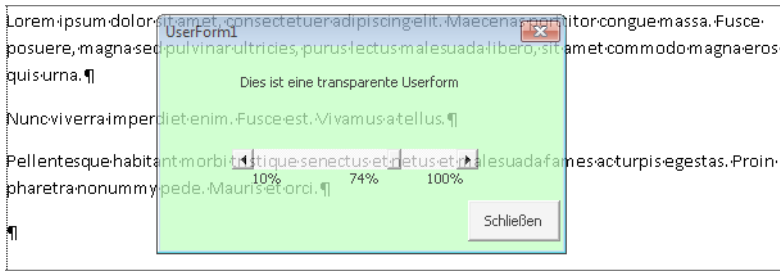
```

Tabelle 3.8 Parameter der API-Funktion *FindWindow*

Parameter	Bedeutung	Ein-/Ausgabe
lpClassName	Klassename des zu findenden Fensters	Ein
lpWindowName	Name des zu findenden Fensters	Ein
hWnd	Handle auf das Fenster	Aus

Diese Funktion erwartet als Eingabeparameter entweder den Klassennamen oder den Namen (Titel) des zu findenden Fensters. Wird das Fenster gefunden, wird ein eindeutiges Handle des Fensters zurückgeliefert, über das dann auf dieses Fenster zugegriffen werden kann. Ist das angegebene Fenster nicht vorhanden, wird »0« zurückgeliefert. Der Klassename für UserForms lautet unter Word (ab Word-Version 9) »ThunderDFrame«, was bei mehreren angezeigten UserForms keine eindeutige Identifizierung erlaubt. Einfacher ist es, das UserForm über den Titel (lpWindowName) zu finden. Da die Titelangabe exakt übereinstimmen muss, liegt es nahe, die Eigenschaft `Caption` des UserForms zu verwenden.

Abbildg. 3.6 Die Hintergrundfarbe der UserForm transparent machen



CD-ROM

Das vollständige Beispiel und ein weiteres Beispiel, wie eine UserForm über die Parameter `lpClassName` und `lpWindowName` ohne Titelleiste angezeigt werden kann, finden Sie in der Beispieldatei *Bsp03_UserformHandle.docm* auf der CD-ROM zum Buch im Ordner `\Beispiele\Kap03\`.

In der Beispieldatei *Bsp03_BrowseForFolder.docm* finden Sie als weiteres Beispiel das Makro *Ordnerauswahl* (mit dem API `SHBrowseForFolder`), das sowohl unter Word 2010 in der 64-Bit- als auch in der 32-Bit-Version funktioniert. Über dieses Makro können Sie einen beliebigen Ordner auswählen und als Vorgabe für den nächsten Aufruf zwischenspeichern.

Auf Registry-Einträge zugreifen

Der Zugriff auf die Registrierungsdateien (Registry) von Windows ist mit nicht unerheblichen Gefahren verbunden, da dieses das Herzstück des Betriebssystems darstellt. Bei falschem Zugriff und Ändern an den Einträgen besteht die Gefahr, dass Windows anschließend nicht mehr korrekt funktioniert und evtl. sogar komplett neu installiert werden muss.

Daher ist es unbedingt empfehlenswert, vor dem Zugriff auf die Registry eine Sicherung dieser Dateien (z.B. mithilfe des Sicherungs-Assistenten die Systemstatusdateien) oder der gesamten Systempartition (z.B. über einen Wiederherstellungspunkt) anzulegen!

Registry-Einträge und -Werte ermitteln

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 Untereinträge (SubKeys) ermitteln.

Dazu muss zuerst der Registry-Eintrag, dessen Untereinträge ermittelt werden soll, für den Zugriff geöffnet werden: Dieses erfolgt mit der API-Funktion `RegOpenKey`.

Die Funktion `RegOpenKey` erwartet als Parameter den Hexadezimalwert des Registryhauptzweigs, den zu öffnenden Eintrag; zurückgegeben wird eine Variable, die eine Zugriffsnummer auf den geöffneten Zweig beinhaltet.

Die Tabelle 3.9 listet die Hauptzweige der Registry auf.

Tabelle 3.9 Hexadezimalwerte der Hauptzweige

Hauptzweig	Hexadezimalwert
<i>HKEY_CLASSES_ROOT</i>	&H0000000
<i>HKEY_CURRENT_CONFIG</i>	&H80000005
<i>HKEY_CURRENT_USER</i>	&H80000001
<i>HKEY_LOCAL_MACHINE</i>	&H80000002
<i>HKEY_USERS</i>	&H80000003

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 3.10 vorgestellt.

```
#If Win64 Then
Private Declare PtrSafe Function RegOpenKey Lib "advapi32.dll" Alias "RegOpenKeyA" ( _
    ByVal hKey As Long, _
    ByVal lpSubKey As String, _
    phkResult As Long ) As Long
Private Declare PtrSafe Function RegCloseKey Lib "advapi32.dll" _
    (ByVal hKey As Long) As Long
#Else
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
#End If
```

Tabelle 3.10 Parameter der API-Funktionen *RegOpenKey* und *RegCloseKey*

Parameter	Bedeutung	Ein-/Ausgabe
<i>hKey</i>	Zugriffsnummer des Hauptzweigs	Ein
<i>lpSubKey</i>	Name des Registry-Eintrags	Ein
<i>phkResult</i>	Variable mit Zugriffsnummer zum Eintrag	Aus

Zum einfachen Auslesen der Einträge werden nicht alle Parameter des APIs `RegEnumKeyEx` benötigt, auch sind nicht alle Parameter beschrieben.

Wichtig ist vor allem die Angabe der Zugriffsnummer des auszulesenden Eintrags sowie eine Buffer-variable zur Aufnahme des ermittelten Namens eines Untereintrags. Die Parameter der folgenden Deklaration sind in Tabelle 3.11 aufgelistet.

```
#If Win64 Then
Private Declare PtrSafe 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
#Else
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
#End If
```

Tabelle 3.11 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 lpClass	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 unterschiedlichen Fehlercode.

Das Listing 3.16 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.docm* von der CD-ROM zum Buch (im Ordner *\Beispiele\Kap03*) ausgeführt haben, liefert die nachstehende Prozedur den Namen des Untereintrags zurück: *Kap03*. Wenn Sie gezielt den Key-Wert eines bestimmten Registry-Eintrags auslesen möchten, können Sie das Makro *ReadVBSetting* in der Beispieldatei *Bsp03_1.docm* verwenden.

Listing 3.15 Ermitteln aller Untereinträge eines Registry-Eintrags

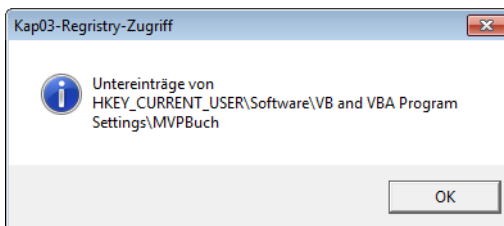
```

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&, _
            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. 3.7 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 3.12.

```
#If Win64 Then
Private Declare PtrSafe 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
#Else
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
#End If
```

Tabelle 3.12 Parameter der API-Funktion *RegEnumValue*

Parameter	Bedeutung	Ein-/Ausgabe
hKey	Zugriffsnummer des Registry-Eintrags	Ein
dwIndex	Index des zu ermittelnden Untereintrags	Ein
lpValueName	Buffer zur Aufnahme des Schlüsselnamens	Ein
lpcbValueName	Länge des Buffereintrags	Ein
lpReserved	Muss NULL sein	Ein
lpType	Schlüsseltyp REG_SZ, REG_MULTI_SZ, REG_EXPAND_SZ	Ein
lpData	Buffer zur Aufnahme des Schlüsselwerts	Ein
lpcbData	Länge des Buffereintrags lpData	Ein

Wenn kein Fehler aufgetreten ist, liefert die Funktion als Rückgabewert `ERROR_SUCCESS = 0`, anderenfalls einen von Null unterschiedlichen Fehlercode.

Zur Ermittlung aller Schlüssel und ihrer Werte wird die Prozedur aus Listing 3.15 dahingehend geändert, wie Listing 3.16 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 3.16 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
    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

```

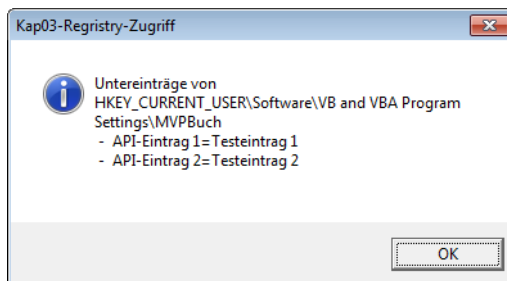
Listing 3.16 Ermitteln aller Schlüssel und Werte aller Untereinträge zu einem Registry-Eintrag (Fortsetzung)

```

intCnt = 0
Const BUFFER_SIZE As Long = 255
' Öffnen des Eintrags
' HKEY_CURRENT_USER\Software\VB and VBA Program Settings\MVPBuch\ + _
' strName(IngCnt)
If RegOpenKey(IngBase, strRoot, IngKey) = 0 Then
    ' Buffer zur Aufnahme der Schlüsselnamen und Werte erstellen
    strName = Space(BUFFER_SIZE)
    strData = Space(BUFFER_SIZE)
    lngRet = BUFFER_SIZE
    lngRetData = BUFFER_SIZE
    'Durchlaufen der Schlüssel
    While RegEnumValue(IngKey, intCnt, strName, lngRet, 0, ByVal 0&, _
        ByVal strData, lngRetData) <> ERROR_NO_MORE_ITEMS
        'sammeln der Schlüsselnamen und Werte
        If lngRetData > 0 Then
            strTemp = strTemp & " - " & Left$(strName, lngRet) & "=" & _
                Left$(strData, lngRetData - 1) & vbCrLf
        End If
        ' Zähler und Buffer für nächsten Eintrag vorbereiten
        intCnt = intCnt + 1
        strName = Space(BUFFER_SIZE)
        strData = Space(BUFFER_SIZE)
        lngRet = BUFFER_SIZE
        lngRetData = BUFFER_SIZE
    Wend
    'Schließen des Registry-Eintrags
    RegCloseKey lngKey
    fktGetKeyValues = strTemp
End If
End Function

```

Als Ergebnis erhalten Sie neben den gefundenen Untereinträgen die vorhandenen Schlüssel und Werte.

Abbildg. 3.8 Ausgabe der gefundenen Untereinträge mit ihren Schlüssel und Werten


CD-ROM Die Beispiele zu diesem Abschnitt finden Sie auf der CD-ROM zum Buch in der Beispieldatei `\Beispiele\Kap03\Bsp03_1.docm` im Modul `modRegAPIs`.

Auf INI-Dateien zugreifen

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 3.17 dargestellt.

Listing 3.17 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 mithilfe 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.

Abschnitte auslesen und schreiben

Mithilfe 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 3.13 aufgelistet):

```
#If Win64 Then
Public Declare PtrSafe Function WritePrivateProfileSection Lib "kernel32" _
    Alias "WritePrivateProfileSectionA" _
    (ByVal lpAppName As String, _
    ByVal lpString As String, _
    ByVal lpFileName As String) As Long
#Else
Public Declare Function WritePrivateProfileSection Lib "kernel32" _
    Alias "WritePrivateProfileSectionA" _
    (ByVal lpAppName As String, _
    ByVal lpString As String, _
    ByVal lpFileName As String) As Long
#End If
```

Tabelle 3.13 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 3.18 schreibt den String

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

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

Listing 3.18 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:

```
#If Win64 Then
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
#Else
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
#End If
```

Eine Auflistung der Parameter finden Sie in Tabelle 3.14.

Tabelle 3.14 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 3.19 veranschaulicht. Das Ergebnis ist in Abbildung 3.9 abgebildet.

Listing 3.19 Auslesen aller Schlüssel und Werte aus dem Abschnitt »Kap03«

```

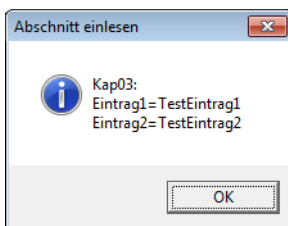
Sub subReadINISection()
    Const c_Read = "Abschnitt einlesen"
    Const c_INIPath = "C:\MVPBuch\Kap03\Bsp03-1.ini"
    Const c_SecName As String = "Kap03"
    Dim lngSec As Long
    Dim strSec As String
    lngSec = 255
    strSec = String(255, vbNullChar)
    GetPrivateProfileSection c_SecName, strSec, 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. 3.9 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.

Alle Abschnitte einer INI-Datei ermitteln

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 3.15 aufgelistet):

```
#If Win64 Then
Public Declare PtrSafe Function GetPrivateProfileSectionNames Lib "kernel32.dll" _
    Alias "GetPrivateProfileSectionNamesA" _
    (ByVal lpszReturnBuffer As String, _
    ByVal nSize As Long, _
    ByVal lpFileName As String) As Long
#Else
Public Declare Function GetPrivateProfileSectionNames Lib "kernel32.dll" _
    Alias "GetPrivateProfileSectionNamesA" _
    (ByVal lpszReturnBuffer As String, _
    ByVal nSize As Long, _
    ByVal lpFileName As String) As Long
#End If
```

Tabelle 3.15 Parameter der API-Funktion *GetPrivateProfileSectionNames*

Parameter	Bedeutung	Ein-/Ausgabe
<code>lpszReturnBuffer</code>	Buffer zur Aufnahme der gefundenen Abschnitte	Aus
<code>nSize</code>	Größe des Buffers	Ein
<code>lpFileName</code>	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 3.20 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 3.10 ersichtlich.

Listing 3.20 Ermitteln und Auslesen aller Abschnitte in der Datei *Bsp03-1.INI*

```

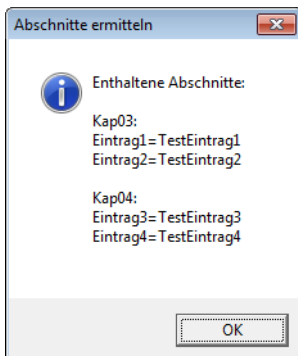
Sub subWrite2INISection()
    Const c_INIPath = "C:\MVPBuch\Kap03\Bsp03-1.INI"
    Const c_SecName As String = "Kap03"
    Const c_SecName2 As String = "Kap04"
    Dim strSection As String
    strSection = "Eintrag1=TestEintrag1" & vbCrLf & "Eintrag2=TestEintrag2"
    ret = WritePrivateProfileSection(c_SecName, 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
    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. 3.10 Ausgabe der gefundenen Abschnitte mit ihren Schlüsseln und Werten



CD-ROM Die Beispiele zu diesem Abschnitt finden Sie auf der CD-ROM zum Buch in der Beispieldatei `\Beispiele\Kap03\Bsp03_1.docm` 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 3.21 veranschaulicht.

Listing 3.21 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`, deren Deklaration sich in Listing 3.22 befindet. Eine Erläuterung der Parameter befindet sich in Tabelle 3.16.

ACHTUNG Die API-Funktion `GetUserNameEx` funktioniert nur für Windows 2000 und aktueller. Unter Windows 9x/ME können Sie die API-Funktion `GetUserName` verwenden:

```
#If Win64 Then
Private Declare PtrSafe Function GetUserName Lib "advapi32.dll" _
    Alias "GetUserNameA" ( _
        ByVal lpBuffer As String, _
        nSize As Long ) As Long
#Else
Private Declare Function GetUserName Lib "advapi32.dll" _
    Alias "GetUserNameA" ( _
        ByVal lpBuffer As String, _
        nSize As Long ) As Long
#End If
```

Listing 3.22 Deklaration der API-Funktion `GetUserNameEx`

```
#If Win64 Then
Private Declare PtrSafe Function GetUserNameEx Lib "secur32.dll" _
    Alias "GetUserNameExA" ( _
        ByVal NameFormat As EXTENDED_NAME_FORMAT, _
        ByVal lpNameBuffer As String, _
```

Listing 3.22 Deklaration der API-Funktion *GetUserNameEx* (Fortsetzung)

```

    ByRef nSize As Long _
) As Long
#Else
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
#End If
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

```

Tabelle 3.16 Parameter der API-Funktion *GetUserNameEx*

Parameter	Bedeutung	Ein-/Ausgabe
NameFormat	Wert aus der Aufzählung, die das Format des Namens angibt. Dieser Wert kann nicht NameUnknown 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 3.23 veranschaulicht den Gebrauch der API-Funktion.

HINWEIS Auf einem Windows 7-Einzelplatzsystem, das nicht an einem Domänencontroller angemeldet ist, liefert nur

```
NameSamCompatible = 2
```

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

Listing 3.23 Ermitteln des Computer- und Benutzernamens mittels *GetUserNameEx*

```

Sub subGetUserNameWindows()
' Benutzernamen unter Windows 2000/XP/Vista/7 auslesen
Dim strBuffer As String
Dim lngBSize As Long

```

Listing 3.23 Ermitteln des Computer- und Benutzernamens mittels *GetUserNameEx* (Fortsetzung)

```

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

```

CD-ROM Die Beispiele zu diesem Abschnitt finden Sie auf der CD-ROM zum Buch in der Beispieldatei `\Beispiele\Kap03\Bsp03_1.docm` 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:

```

#If Win64 Then
Private Declare PtrSafe Sub Sleep Lib "kernel32" (ByVal lngMilliseconds As Long)
#Else
Private Declare Sub Sleep Lib "kernel32" (ByVal lngMilliseconds As Long)
#End If

```

Die Erläuterung der Parameter befindet sich in Tabelle 3.17 und das Listing 3.24 veranschaulicht den Gebrauch.

Tabelle 3.17 Parameter der API-Funktion *Sleep*

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

Listing 3.24 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 3.25 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 %.

CD-ROM Die Beispiele zu diesem Abschnitt finden Sie auf der CD-ROM zum Buch in der Beispieldatei `\Beispiele\Kap03\Bsp03_1.docm` 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 Fehlersound quittieren oder das Beenden einer Verarbeitung mit einem Hinweissound. Dazu können Sie entweder eigene Wave-Dateien oder die standardmäßig installierten Systemsounds verwenden.

Zum Abspielen von Wave-Dateien können Sie die API-Funktion `sndPlaySound` in Ihre Anwendung einbinden. Im Folgenden sehen Sie Deklaration, die Parameter sind in Tabelle 3.18 aufgelistet:

```
#If Win64 Then
Public Declare PtrSafe Function sndPlaySound Lib "WinMM.dll" _
    Alias "sndPlaySoundA" ( _
        ByVal lpszSoundName As String, _
        ByVal uFlags As Long ) As Long
#Else
Public Declare Function sndPlaySound Lib "WinMM.dll" _
    Alias "sndPlaySoundA" ( _
        ByVal lpszSoundName As String, _
        ByVal uFlags As Long ) As Long
#End If
```

Tabelle 3.18 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 3.19 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 Sounddatei nicht gefunden wird, wird nicht die konfigurierte Standarddatei »Standardton Warnsignal« abgespielt
SND_LOOP	Legt fest, dass die Datei endlos abgespielt wird, bis entweder eine andere Datei abgespielt oder die Funktion mit einem Leerstring aufgerufen wird.
SND_NOSTOP	Wenn bereits eine Sounddatei 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 SND_LOOP in einer Endlosschleife abgespielt wird
SND_NOWAIT	Spielt die Datei sofort ab, auch wenn bereits eine andere Datei abgespielt wird

Das Beispiel in Listing 3.25 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 3.26 Abspielen einer beliebigen Wave-Datei unter Word 2010 und Word 2007

```

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 subPlaySoundWord2010()
    ' Wav-Datei unter Word 2010 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 der Beispieldatei `\Beispiele\Kap03\Bsp03_1.docm` in der Prozedur `subPlaySoundWord2000` des Moduls `modsndPlaySoundAPI`.

HINWEIS

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

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 3.27):

- Durch Angabe des NULL-Zeichens als Dateinamen
- Durch Verwendung des `SND_PURGE`-Parameters

Listing 3.27

Möglichkeit einer Endlosschleife oder um das Abspielen einer Datei zu beenden

```

Sub subStopSound()
    ' Endlosschleife/Abspielen mittels NULL beenden
    procPlayWAVKlang vbNull, SND_ASYNC

```

Listing 3.27 Möglichkeit einer Endlosschleife oder um das Abspielen einer Datei zu beenden (*Fortsetzung*)

```
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
```

CD-ROM Die Beispiele zu diesem Abschnitt finden Sie auf der CD-ROM zum Buch in der Beispieldatei `\Beispiele\Kap03\Bsp03_1.docm` 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 3.20 aufgelistet.

```
#If Win64 Then
Declare PtrSafe Function GetKeyState Lib "user32" ( _
  ByVal nVirtKey As Long ) As Integer
#Else
Declare Function GetKeyState Lib "user32" ( _
  ByVal nVirtKey As Long ) As Integer
#End If
```

Tabelle 3.20 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 3.21 liefert eine Übersicht über die gängigen Tastenkonstanten.

Tabelle 3.21 Virtuelle Tastenkonstanten mit Bedeutung und Hexadezimalwert

Virtuelle Tastenkonstante	Taste	Hexadezimalwert
VK_LBUTTON	Linke Maustaste	&H1
VK_RBUTTON	Rechte Maustaste	&H2
VK_CANCEL	(Strg) + (Untbr)	&H3
VK_MBUTTON	Mittlere Maustaste	&H4
VK_XBUTTON1	X1-Maustaste	&H5
VK_XBUTTON2	X2-Maustaste	&H6
VK_BACK	(Rück)	&H8
VK_TAB	(Tab)	&H9
VK_CLEAR	(Entf)	&HC
VK_RETURN	(↵)	&HD
VK_SHIFT	(⇧)	&H10
VK_CONTROL	(Strg)	&H11
VK_MENU	(Alt)	&H12
VK_PAUSE	(Pause)	&H13
VK_CAPITAL	(CapsLock)	&H14
VK_ESCAPE	(Esc)	&H1B
VK_SPACE	(Leertaste)	&H20
VK_PRIOR	(Bild ↑)	&H21
VK_NEXT	(Bild ↓)	&H22
VK_END	(Ende)	&H23
VK_HOME	(Pos1)	&H24
VK_LEFT	(←)	&H25
VK_UP	(↑)	&H26
VK_RIGHT	(→)	&H27
VK_DOWN	(↓)	&H28
VK_SELECT	Markieren	&H29
VK_PRINT	(Druck)	&H2A
VK_EXECUTE	Ausführen	&H2B
VK_SNAPSHOT	Bildschirm drucken	&H2C
VK_INSERT	(Einf)	&H2D
VK_DELETE	(Entf)	&H2E

Tabelle 3.21 Virtuelle Tastenkonstanten mit Bedeutung und Hexadezimalwert (Fortsetzung)











































Virtuelle Tastenkonstante	Taste	Hexadezimalwert
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
VK_F11		&H7A
VK_F12		&H7B
VK_F13		&H7C
VK_F14		&H7D

Tabelle 3.21 Virtuelle Tastenkonstanten mit Bedeutung und Hexadezimalwert (Fortsetzung)

Virtuelle Tastenkonstante	Taste	Hexadezimalwert
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 3.28 ersichtlich. Ist das Ergebnis wieder der Hexadezimalwert &HF000&, dann ist die Taste gedrückt.

Listing 3.28 Prüfen des Tastenstatus mittels *GetKeyState*

```
Public Function funcIsTasteGedrückt(iTastenCode As Integer) As Boolean
' Überprüft bitweise, ob eine Taste gedrückt ist
If (GetKeyState(iTastenCode) And &HF000&) = &HF000& Then
    funcIsTasteGedrückt = True
End If
End Function
```

HINWEIS Beim bitweisen Vergleich, oder auch Binärvergleich, werden zwei Zahlen in Binärschreibweise miteinander verglichen. Binärstellen, die bei beiden Zahlen identisch sind, werden ins Ergebnis übernommen, für unterschiedliche Zahlen ist das Ergebnis 0.

Liefert die API-Funktion als Rückgabewert »-128«, ist das Ergebnis des bitweisen Vergleichs mit &HF000& wieder dieser Hexadezimalwert:

&HFF80	-128	1111111110000000
&HF000	61440	1111000000000000

&HF000	61440	1111000000000000




Hingegen liefert ein Rückgabewert von 0 im Vergleich wieder 0:


&H0000	0	0000000000000000
&HF000	61440	1111000000000000

&H0000	0	0000000000000000

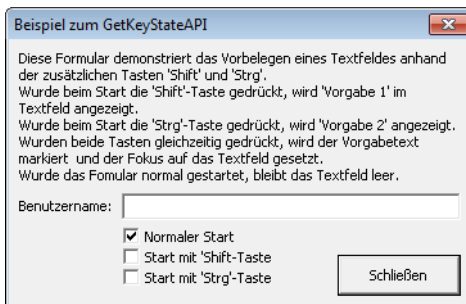
Als Beispiel wird der Aufruf der UserForm *frmGetKeyStateAPI* in der Beispieldatei *Bsp03_1.docm* verwendet. In Listing 3.29 wird die Abfrage evtl. gedrückter Tasten in einer Select Case-Anweisung geprüft, wobei mithilfe der Funktion *funcIsTasteGedrückt* aus Listing 3.28 der Tastenstatus ermittelt wird. Im ersten Case-Abschnitt wird geprüft, ob beide Tasten, und in den beiden nächsten Abschnitten, ob die beiden einzeln gedrückt wurden.

Listing 3.29 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 Menübefehl *Ausführen/ Sub/UserForm ausführen*. Es wird die in Abbildung 3.11 gezeigte UserForm mit einem leeren Feld *Benutzername* angezeigt.

Abbildg. 3.11 Normaler Start des Benutzerformulars ohne zusätzlich gedrückte Taste





Beispiel zum GetKeyStateAPI


Diese Formular demonstriert das Vorbelegen eines Textfeldes anhand der zusätzlichen Tasten 'Shift' und 'Strg'.
 Wurde beim Start die 'Shift'-Taste gedrückt, wird 'Vorgabe 1' im Textfeld angezeigt.
 Wurde beim Start die 'Strg'-Taste gedrückt, wird 'Vorgabe 2' angezeigt.
 Wurden beide Tasten gleichzeitig gedrückt, wird der Vorgabetext markiert und der Fokus auf das Textfeld gesetzt.
 Wurde das Formular normal gestartet, bleibt das Textfeld leer.

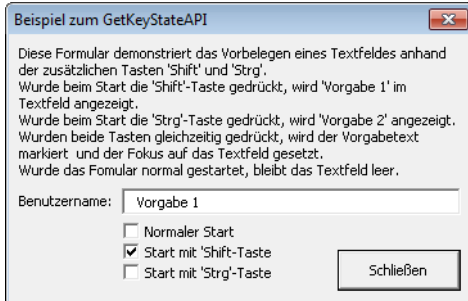
Benutzername:

☒ Normaler Start
☐ Start mit 'Shift'-Taste
☐ Start mit 'Strg'-Taste


Schließen

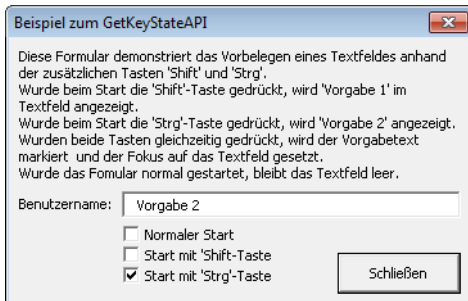
Wenn Sie beim Start zusätzlich die -Taste gedrückt halten, wird es mit dem Benutzernamen »Vorgabe 1« angezeigt. Wenn Sie die -Taste gedrückt halten, wird der Benutzername »Vorgabe 2« angezeigt.

Abbildg. 3.12 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. 3.13 Start des Benutzerformulars mit gedrückter -Taste



Bei jedem Aufruf des Benutzerformulars wird zusätzlich über die Kontrollkästchen angezeigt, wie das Benutzerformular aufgerufen wurde.

CD-ROM Die Beispiele zu diesem Abschnitt finden Sie auf der CD-ROM zum Buch in der Beispieldatei `\Beispiele\Kap03\Bsp03_1.docm` im Modul `modGetKeyStateAPI`.

Zusammenfassung

In diesem Kapitel wurde Ihnen gezeigt, welche Möglichkeiten Ihnen die Verwendung von 32-Bit-APIs auch in Word 2010 (32 Bit und 64 Bit) bietet, auch wenn nicht alle Windows-APIs in VBA genutzt werden können.

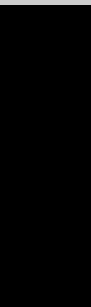
- Nach dem prinzipiellen Aufbau der API-Funktionen (Seite 118) und den Unterschieden zwischen API-Funktionen unter 32-Bit- und 64-Bit-Office-Versionen wurden Ihnen API-Funktionen vorgestellt, mit denen Sie relativ einfach mehrere Pfade (Seite 118 ff.) und Dateinamen (Seite 125) syntaktisch korrekt miteinander kombinieren
- Des Weiteren wurde gezeigt, wie Sie eine Datei über ihre Dateiendung aufrufen können (Seite 127)
- Es wurde zusätzlich gezeigt, wie Sie eine Benutzerformular finden und es transparent oder ohne Titelleiste darstellen können (Seite 131)
- Es wurden Funktionen vorgestellt, um gezielt auf Einträge in der Windows Registry zuzugreifen (Seite 131 ff.), und mit denen Sie Registry-Einträge und -werte ermitteln können (Seite 132)
- Ein weiterer Schwerpunkt dieses Kapitels war das Auslesen von und Schreiben in INI-Dateien mittels API-Funktionen (Seite 139 ff.)
- Weitere Beispiele haben Ihnen gezeigt, wie Sie den Namen des gerade angemeldeten Benutzers ermitteln (Seite 144) und die Verarbeitung von Makros für eine bestimmte Zeit unterbrechen können (Seite 146)
- Anhand eines weiteren Beispiels wurde eine API-Funktion vorgestellt, um eine beliebige Wave-Datei abzuspielen (Seite 147 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 150)

Teil B

Das Objektmodell von Word

In diesem Teil:

Kapitel 4	Überblick über die Arbeit mit Objekten	159
Kapitel 5	Grundlagen des Objektmodells	171
Kapitel 6	Professionelle Dokumente	241
Kapitel 7	Word und Datenstrukturen	341
Kapitel 8	Ereignisse in Word	453



Kapitel 4

Überblick über die Arbeit mit Objekten

In diesem Kapitel:

Arbeiten mit Objekten	160
Auflistung von Objekten	165
Auflistung bearbeiten	166
Zusammenfassung	170

Im ersten Teil dieses Buchs 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 Nichtexistenz 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:

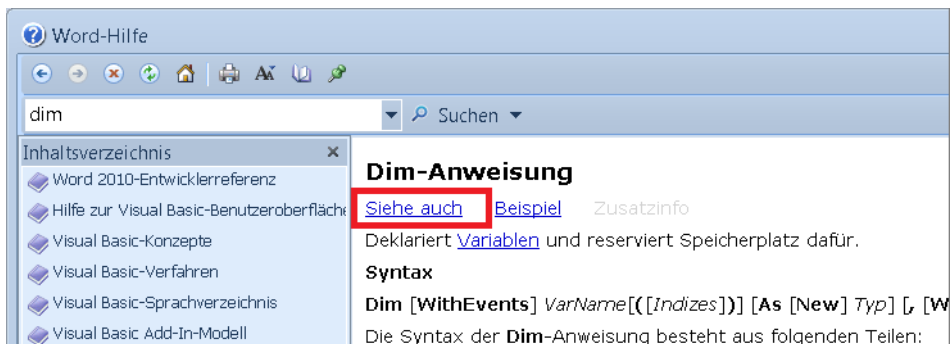
- Die *Word-Hilfe* wurde bereits in Kapitel 1 im Abschnitt »Die Objektmodell-Hilfe – eine versteckte Schatzkammer« vorgestellt. Aus unserer Sicht ist der wichtigste Hinweis auf jeder Hilfe-seite die Verknüpfung *Siehe auch*. Denn hier geht es zu den verwandten Themen, und wenn die Neugierde erst einmal geweckt wurde, dann werden viele interessante Sachen entdeckt.



Word-Hilfe

Abbildg. 4.1

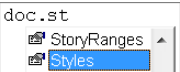
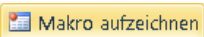
Lesen Sie die verwandten Themen, um ein Objekt aufzuspüren und besser kennenzulernen



Objekt-
katalog

Intelli-
Sense

Makrore-
korder

- Ein weiteres Hilfsmittel ist der *Objektkatalog*, der einen umfassenden Einblick in jede Software-Bibliothek freigibt und dessen Möglichkeiten bereits 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, die persönliche Neugierde sowie die wachsende 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  
As  
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 14.0 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 freigegeben 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 14.0 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

In .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

Bei 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 Objekts.

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.docx") '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
        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 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).

Each...Next 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
End Sub
```

Listing 4.3 Auflistung aller Elemente einer Auflistung mittels einer *For...Next*-Schleife (Fortsetzung)

```
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

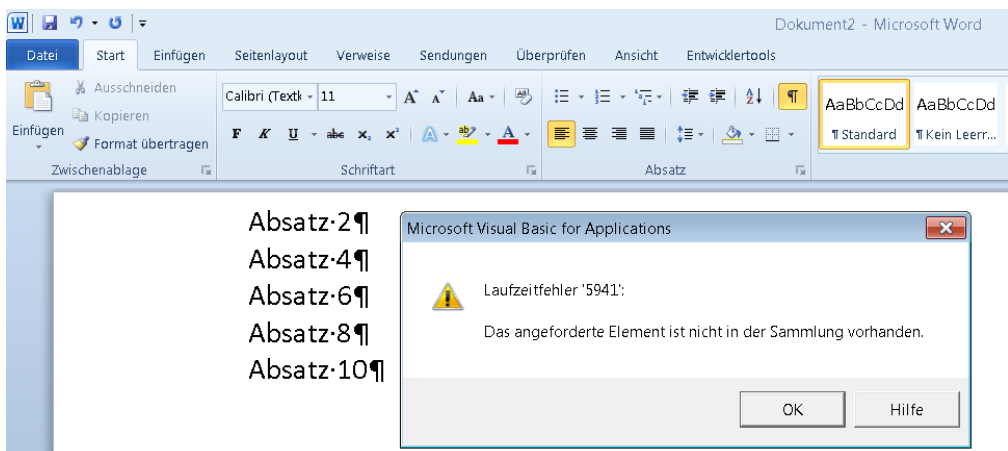
    For Each doc In Application.Documents
        strText = strText & doc.Name & vbCrLf
    Next doc

    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.

CD-ROM Die aufgeführten Codesequenzen finden Sie in der Beispieldatei *Bsp04_01.docm* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap04*.

Zusammenfassung

Dieses Kapitel hat Ihnen einen kurzen Einstieg in die Welt der Objekte vermittelt. Das Objektmodell von Word ist mächtig und deshalb sind die Themen in diesem Kapitel wichtig.

- Es wurde erläutert, wie ein gesuchtes Objekt einfach ermittelt (Seite 160) und dieses einer Variable zugewiesen werden kann (Seite 161)
- Es wurde gezeigt, wie eine Auflistung von Objekten bearbeitet und wie auf deren einzelne Objekte zugegriffen wird (Seite 165 ff.)

Kapitel 5

Grundlagen des Objektmodells

In diesem Kapitel:

Das <i>System</i> -Objekt	173
Die Anwendung: das <i>Application</i> -Objekt	176
Die gegenwärtige Markierung: <i>Selection</i> und ähnliche Objekte	186
Der Kern der Sache: das <i>Document</i> -Objekt	188
Dokumentvorlagen: das <i>Template</i> -Objekt	202
Mit Bereichen arbeiten: das <i>Range</i> -Objekt	207
Die Nadel im Heuhaufen: <i>Find/Replace</i> einsetzen	220
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. Sogar langjährige, fortgeschrittene Anwender hadern gelegentlich mit der internen Logik von Word.

Die folgenden drei Kapitel sind daher kein Wiederkäuen der VBA-Hilfdateien. Vielmehr möchten wir Ihnen einerseits interessante Möglichkeiten vorstellen und Sie andererseits mit den Word-internen Zusammenhängen vertraut machen, sodass Sie in Ihrem Code das Word-Objektmodell zweckmäßig einsetzen können. Wir stellen die meist gebrauchten (oder missverstandenen) Objekte sowie in Word 2010 neue Funktionalität vor und erläutern anhand kurzer Codebeispiele deren programmtechnischen Einsatz.

CD-ROM

Ausführlichere Beispiele zum Zusammenspiel mehrerer Teile des Objektmodells finden Sie im Bonusteil auf der CD-ROM im Ordner *\Bonus*.



Die Codebeispiele 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 Dinge 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.

HINWEIS

Der VB.NET-Entwickler muss lediglich darauf achten, konstant Werte (»Enums«) vollständig zu qualifizieren. Statt beispielsweise das Dateiformat des zu speichernden Dokuments mit `wdOpenFormatDocument` festzulegen, benötigt er `Word.WdOpenFormat.WdOpenFormatDocument`.

Dank der Einführung von optionalen benannten Parametern und Dynamics in C# 4.0 kann der C#-Entwickler, der mit dieser Version arbeitet, die VBA-Beispiele ebenso verwenden wie der VB.NET-Entwickler. Wir weisen jedoch darauf hin, dass ein solcher Code langsamer sein könnte, da der Compiler diese Arbeit übernimmt. Mehr Informationen dazu finden Sie unter [http://msdn.microsoft.com/en-us/library/bb383815\(VS.100\).aspx](http://msdn.microsoft.com/en-us/library/bb383815(VS.100).aspx).

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 Rechnertyp, Bildschirmauflösung, zur Verfügung stehende Prozessoren und Speicherplatz abfragen. Eigenschaften von besonderem Interesse stellen wir hier kurz vor.





Country-Region

In der Hilfe heißt es: »Gibt die Länder- bzw. Regionseinstellung des Systems zurück. Schreibgeschützter WdCountry-Wert«. Gemeint ist die Liste im Abschnitt *Standards und Formate* der Registerkarte *Regionale Einstellungen* in den *Regions- und Sprachoptionen* der Systemsteuerung. Die Hilfe präzisiert nicht, dass nicht allen Ländern ein WdCountry-Wert zugewiesen wurde. Deutschland ist beispielsweise mit wdGermany dabei, die Schweiz und Österreich jedoch nicht. Falls Sie diese Einstellung abfragen wollen, müssen Sie die numerischen Werte herausfinden, indem Sie die Einstellung ändern und diese Eigenschaft abfragen. Es stellt sich heraus, dass viele Werte der Ländervorwahl entsprechen, was die Sache etwas vereinfacht. Die Schweiz hat den Wert 41, Österreich den Wert 43.

Cursor

Mit Cursor ist der Mauszeiger gemeint. Wie wir alle wissen, dient er nicht nur zur Auswahl auf dem Bildschirm, sondern er teilt uns auch mit, wenn wir warten müssen (die Sanduhr), ob wir Text markieren können und Ähnliches. Die verschiedenen Formen der Mauszeiger werden von Anwendungsprogrammierern festgelegt, die sich ihrerseits auf Einstellungen in Windows stützen. Diese Eigenschaft ermöglicht es dem Word-VBA-Entwickler, die Form des Mauszeigers abzufragen und festzulegen. Da Word-Dokumente keine mausbezogenen Ereignisse wie »Mouseover« unterstützen, sind die Einsatzmöglichkeiten begrenzt, aber die Sanduhr bietet sich für lange Prozeduren geradezu an. Die Mauszeigertypen sind in Tabelle 5.1 aufgelistet.

Tabelle 5.1 Die verschiedenen Mauszeigerformen des Cursor-Objekts

WdCursor-Wert	Mauszeigerform
wdCursorIBeam	
wdCursorNormal	
wdCursorNorthwestArrow	
wdCursorWait	

HINWEIS Benutzerdefinierte Formulare (»UserForms«) unterstützen 16 verschiedene Mauszeigerformen. Schauen Sie in den Eigenschaften eines Formulars nach, wenn Sie dafür die Mauszeigerform ändern möchten. Mehr über Formulare erfahren Sie in Kapitel 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 und Version	Die Eigenschaft OperatingSystem liefert den Namen des Betriebssystems, die Eigenschaft Version die zugehörige Versionsnummer. Allerdings werden Sie nicht »Windows 2000«, »Windows XP« oder »Windows z« 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, Windows Vista der Version 6.0 und Windows 7 der Version 6.1 entspricht.
PrivateProfileString	Diese ist die nützlichste aller System-Eigenschaften. Damit können Einstellungen in der Registry (oder aber auch in einer INI-Dateien) beliebig gelesen und geschrieben werden. Dies im Gegensatz zur ProfileString-Eigenschaft, die nur den Registryabschnitt für Anwendungen anspricht:

```
HKEY_CURRENT_USER\Software\Microsoft\Office\<Version>\Word
```

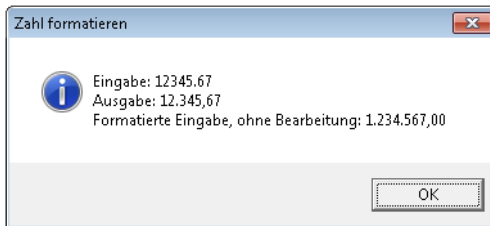
So greifen Sie beispielsweise direkt auf die Einstellungen unter *Regions- und Sprachoptionen* der Systemsteuerung zu, um das Dezimal- und Listen-Trennzeichen oder das Symbol für die Zifferngruppierung herauszufinden. Da Word nur die Zahlen- und Datumsformate umsetzt, die im Betriebssystem festgelegt sind, ist der Zugang zu dieser Information für die Arbeit im mehrsprachigen Umfeld von unschätzbarem Wert.

HINWEIS

Geben Sie »regedit« (ohne Anführungszeichen) in das Suchfeld des Windows-Startmenüs ein, dann  drücken, um den Registrierungseditor von Windows zu starten.

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 durch die deutschen ersetzt



Listing 5.1 Einstellungen aus der Registry lesen

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

Sub ZahlMitSystemEinstellungenFormatieren()
    Dim strDezimalTrennzeichenBenutzer As String
    Dim strDezimalTrennzeichen As String
    Dim strGruppierungsSymbolBenutzer As String
    Dim strGruppierungsSymbol As String
    Dim strEingabe As String, strZahlenFormat As String
    Dim sAnzahlDezimalstellen As String, sGruppierungsstellen As String
    Dim lErsteGruppierungsstelle As Long
```

Listing 5.1 Einstellungen aus der Registry lesen (Fortsetzung)

```

Dim strBearbeiteteEingabe As String, strAusgabe As String

strZahlenFormat = "#,##0.00" 'Standardisiertes VBA-Zahlenformat
strEingabe = "12345.67"
strBearbeiteteEingabe = strEingabe
With System
    sAnzahlDezimalstellen = .PrivateProfileString( _
        FileName="", _
        Section:="HKey_Current_User\Control Panel\International", _
        Key:="iDigits")
    sGruppierungsstellen = .PrivateProfileString( _
        FileName="", _
        Section:="HKey_Current_User\Control Panel\International", _
        Key:="sGrouping")
    strDezimalTrennzeichenBenutzer = .PrivateProfileString( _
        FileName="", _
        Section:="HKey_Current_User\Control Panel\International", _
        Key:="sDecimal")
    strGruppierungsSymbolBenutzer = .PrivateProfileString(FileName="", _
        Section:="HKey_Current_User\Control Panel\International", _
        Key:="sThousand")
End With
sGruppierungsstellen = Left(sGruppierungsstellen, 1)
If IsNumeric(strBearbeiteteEingabe) Then
    'Wurde vom System als gültige Zahl erkannt.
    'Das Dezimaltrennzeichen der eingegebenen Zahl ermitteln
    strDezimalTrennzeichen = Mid(strBearbeiteteEingabe, Len(strBearbeiteteEingabe) _
        - CInt(sAnzahlDezimalstellen), 1)
    'Das DezimalTrennzeichen mit einem "x" ersetzen, da es
    'das gleiche Symbol wie das Gruppierungssymbol sein könnte.
    strBearbeiteteEingabe = Replace(strBearbeiteteEingabe, strDezimalTrennzeichen, "x")
    'Die Zahl könnte Tausendertrennzeichen enthalten.
    lErsteGruppierungsstelle = CInt(sGruppierungsstellen) + CInt(sAnzahlDezimalstellen) _
        + Len(strDezimalTrennzeichen)
    'Falls die Zahl lang genug ist, um Gruppierungssymbole zu enthalten...
    If Len(strBearbeiteteEingabe) >= lErsteGruppierungsstelle Then
        '...das Symbol ermitteln - es wird nicht numerisch sein.
        If Not IsNumeric( _
            Left(Right(strBearbeiteteEingabe, lErsteGruppierungsstelle), 1)) Then
            strGruppierungsSymbol = Left(strBearbeiteteEingabe, 1)
            'Das vorhandene Gruppierungssymbol mit dem vom System ersetzen.
            strEingabe = Replace( _
                strBearbeiteteEingabe, strGruppierungsSymbol, strGruppierungsSymbolBenutzer)
        End If
    End If
    strAusgabe = Replace(strBearbeiteteEingabe, "x", strDezimalTrennzeichenBenutzer)
    strAusgabe = Format(strAusgabe, strZahlenFormat)
    MsgBox "Eingabe: " & strEingabe & vbCrLf & "Ausgabe: " & strAusgabe & vbCrLf & _
        "Formatierte Eingabe, ohne Bearbeitung: " & Format(strEingabe, strZahlenFormat), _
        vbInformation + vbOKOnly, strMSGTITEL
Else
    MsgBox "Die Eingabe ist keine gültige Zahl."
End If
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 3).

CD-ROM Die Beispieldatei *Bsp05_01_System.docm* 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 das Kapitel 13.

Die Anwendung: das *Application*-Objekt

Das `Application`-Objekt ist die Word-Anwendung. Denken Sie zurück an die erste Abbildung dieses Buchs 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,
- das Menüband (Multifunktionsleiste in Word 2007; auch »Ribbon« genannt), 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 Codemodul eintippen. Visual Basic for Applications ist Anwendungs-spezifisch und weiß, zu welcher Anwendung es gehört. Das erspart dem Programmierer die »zusätzliche« Arbeit, Objektnamen der obersten Ebene mit `Application` qualifizieren zu müssen.

Wird Word dagegen aus einer anderen Umgebung heraus gesteuert, sei es aus Excel, Access oder .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 Buchs über die Interoperabilität behandelt und taucht auch in manchem .NET-Codebeispiel in diesem Buch auf.

Einzelne Objekte von besonderem Interesse werden in anderen Abschnitten und Kapiteln behandelt. Hier verweisen wir lediglich auf die folgenden Elemente:

- `UndoRecords` (neu in Word 2010)
- `ScreenUpdating`
- `Visible`
- `DisplayAlerts`
- `AutomationSecurity`
- `PathSeparator`
- `Language`
- Optionen (`Options`-Objekt)

- DefaultFilePath
- WordBasic-Befehle

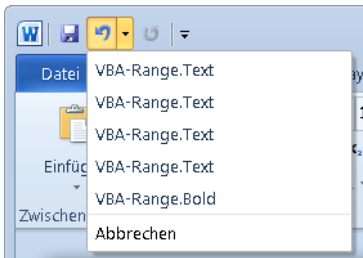


2010
Undo-
Record

Seit Einführung des Befehls *Rückgängig machen* ärgert sich der Entwickler (und seine Benutzer) darüber, dass fast jede vom Code ausgeführte Handlung einzeln in dieser Liste aufgeführt wird. Mit diesem neuen Befehl können mehrere Handlungen in einem einzigen Eintrag zusammengefasst werden.

Nehmen wir als Beispiel Listing 5.2, den bearbeiteten Makrocode für die Formatierung einer Tabelle. Wie aus Abbildung 5.2 ersichtlich, werden mehrere Einträge in die Liste geschrieben. Falls der Benutzer die Makrohandlung rückgängig machen will, muss er den Anfangspunkt erkennen und anwählen, was viele Benutzer überfordert.

Abbildg. 5.2 Makrohandlungen in der Liste *Rückgängig machen*



Wird UndoRecord angewendet, wie in Listing 5.2 veranschaulicht, enthält die Liste einen einzigen Eintrag (Abbildung 5.3). Falls Sie den Parameter Name der Methode StartCustomRecord weglassen, besteht die Eintragsbeschriftung aus der erst ausgeführten VBA-Methode (wie »VBA.Range.Text«). Beachten Sie auch die Verwendung der neuen Compilerkonstante VBA7, um Fehlermeldungen zu vermeiden, falls das Makro unter einer älteren Version von Word ausgeführt wird.

Listing 5.2 Anwendung der Eigenschaft *UndoRecord*

```
Sub BefehleInUndoListeGruppieren()
    #If VBA7 Then
        Dim UndoListe As Word.UndoRecord
        'Ein UndoRecord-Objekt erstellen, um Befehle zu gruppieren
        Set UndoListe = Application.UndoRecord
        'Die Gruppierung der Befehle anfangen
        'und der Gruppierung einen Namen geben.
        UndoListe.StartCustomRecord Name:="Formatierte Tabelle"
        'Handlungen ausführen.
        TabelleVorbereiten3
        'Die Gruppierung der Makro-Handlungen beenden.
        UndoListe.EndCustomRecord
        Set UndoListe = Nothing

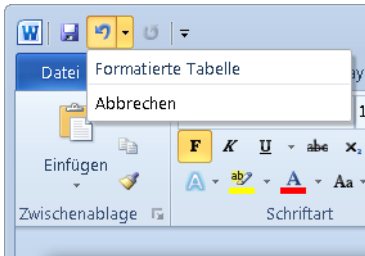
        'Sicherstellen, dass Benutzer-Handlungen in der Liste wieder aufgenommen werden.
        Dim lAnzahlCustomRecords
        Do While Application.UndoRecord.IsRecordingCustomRecord
            lAnzahlCustomRecords = lAnzahlCustomRecords + 1
            Application.UndoRecord.EndCustomRecord
        Loop
        If lAnzahlCustomRecords > 0 Then
            MsgBox "UndoRecord wurde " & CStr(lAnzahlCustomRecords) & _
```

Listing 5.2 Anwendung der Eigenschaft *UndoRecord* (Fortsetzung)

```

        " ausgeführt und nicht explizit beendet."
    End If
#End If
End Sub

```

Abbildg. 5.3 *UndoRecord* gruppiert Makrohandlungen in einem einzigen Eintrag in der Liste *Rückgängig machen*

ACHTUNG

Im Gegensatz zu den meisten anderen Befehlen des Word-Objektmodells läuft die Gruppierung von Handlungen nach Beendigung der Prozedur weiter. Dies bedeutet, dass auch Benutzerhandlungen nicht in der Liste *Rückgängig machen* erscheinen. Zudem sind Ausführungen des Befehls `StartCustomRecord` kumulativ – jedes Anstoßen muss explizit mit `EndCustomRecord` angehalten werden. Die letzten Zeilen des Listings 5.2 veranschaulichen, wie alle laufenden Instanzen der Gruppierung angehalten werden können.

ScreenUp
dating

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 Dokumentfenster zu minimieren oder sogar das Dokument oder die ganze Anwendung mittels der `Visible`-Eigenschaft unsichtbar zu machen:

```
Application.Visible = False
```

Falls Sie Word von einer anderen Umgebung aus steuern, startet Word automatisch unsichtbar. Um mit dem Benutzer interaktiv zu arbeiten, müssen Sie zuerst die Anwendung sichtbar machen, indem Sie diese Eigenschaft auf `True` stellen.

WICHTIG Die Word-Anwendung wurde als interaktive Schnittstelle konzipiert. Das Layout eines Dokuments stützt sich auf den dynamischen Textfluss auf der virtuellen Seite. Ist das Dokumentfenster unsichtbar, wird das Fertigstellen des Layouts eventuell beeinträchtigt, was abweichende Automatisierungsergebnisse liefern kann. Dies wird hauptsächlich bei der Arbeit mit dem Selection-Objekt zum Problem, kann aber auch in anderen Fällen auftreten. Liefert Ihr Code bei nicht sichtbarem Dokument-Fenster ein unterschiedliches Resultat, müssen Sie das Dokument-Fenster sichtbar machen.

Display-
Alerts

Wie bereits erwähnt, ist Word als interaktive Anwendung konzipiert und blendet für den Benutzer gelegentlich auch Meldungen ein. Für eine Automatisierung sind diese oft hinderlich, lassen sich jedoch nicht gänzlich abschalten. Immerhin können wir einige der (aus Sicht der Anwendung) weniger kritischen Meldungen mit der Eigenschaft `DisplayAlerts` unterbinden:

```
Application.DisplayAlerts = wdAlertsNone
```



In der .NET-Umgebung muss der Konstantenwert voll qualifiziert sein. Das Beispiel in C#:

```
WordApplication.DisplayAlerts = Word.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 Codeausführung wird diese Eigenschaft *nicht* automatisch auf `wdAlertsAll` zurückgesetzt. Sie müssen in Ihrem Code selbst dafür sorgen. Das gilt sowohl für die Automatisierung aus einer anderen Umgebung als auch für Code in Word-VBA-Modulen.

Automa-
tionSecu-
rity

Nicht selten muss Automatisierungscode Dokumente in Word öffnen. Eine Meldung, die `DisplayAlerts` nicht unterdrückt, ist die Makrosicherheitsmeldung (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
```



Dieser Konstantenwert befindet sich im Office-, nicht im Word-Objektmodell. Das Beispiel in C#:

```
WordApplication.AutomationSecurity = _  
Microsoft.Office.Core.MsoAutomationSecurity.msoAutomationSecurityForceDisable;
```

HINWEIS

Ein großes Problem für das automatisierte Öffnen von Dokumenten sind darin enthaltene Auto-Makros, die beim Öffnen eine Handlung ausführen. Diese kommen oft der Ausführung des eigenen Codes in die Quere. Mit `ApplicationSecurity` auf `msoAutomationSecurityForceDisable` kann das Problem umgangen werden, die übrige Makro-Funktionalität bleibt aber gesperrt. Es gibt einen alten WordBasic-Befehl, `DisableAutoMacros`, der nur die Auto-Makros ausschaltet. Mehr über die Verwendung von WordBasic finden Sie weiter unten in diesem Abschnitt.

Path-Separator	Nicht nur Word für Windows (WinWord), sondern auch viele Versionen von Word für Macintosh (MacWord) bietet VBA als Programmierschnittstelle. Die beiden Betriebssysteme definieren Pfadangaben jedoch anders. In Plattform-übergreifendem Code ist daher die <code>PathSeparator</code> -Eigenschaft des <code>Application</code> -Objekts sehr hilfreich, die das Pfadtrennzeichen für das aktuelle Betriebssystem – für Windows einen Backslash (\); für Macintosh ein Doppelpunkt (:) – zurückgibt.
Language	Die <code>Language</code> -Eigenschaft gibt eine Ganzzahl zurück, die einem <code>MsoLanguageID</code> -Konstantwert entspricht. Daraus wird ermittelt, welche Sprache in der Word-Umgebung herrscht. Eine Liste der <code>MsoLanguageID</code> -Konstantwerte finden Sie im Objektkatalog des VB-Editors.

HINWEIS

Für das englische Word haben Firmen mit Lizenzverträgen ab der Version 2000 bis einschließlich 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, sodass der Startpunkt nicht zwingend eine englische Version voraussetzt. Zudem stehen die Schnittstellen als Teil eines »Microsoft Office [Version] Multi-Language Packs« allen Kunden zur Verfügung. Mehr Informationen erhalten Sie unter <http://office.microsoft.com/de-de/suites/FX102113661031.aspx>.

Options	Mittlerweile enthält Word 2010 mehr als 230 Eigenschaften zum <code>Options</code> -Objekt (in Word 2003 waren es etwa 120). Die meisten sind in den Kategorien der <i>Word-Optionen</i> auf der Registerkarte <i>Datei</i> zu finden. Manche sind in der deutschen Version nicht sichtbar, da es – vor allem für die asiatischen Versionen – auch sprachspezifische Einstellungen gibt.
---------	--

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 eines Abschnitts aufzeichnet und nicht nur diejenigen, deren Einstellungen geändert wurden. Dies bedeutet:

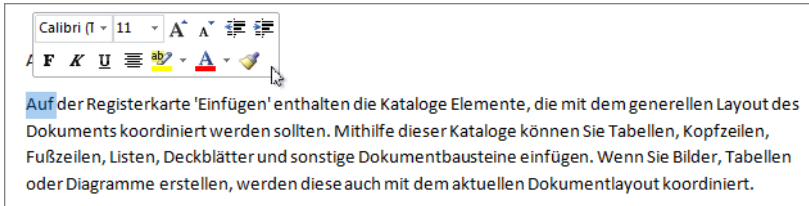
- Um die Übersicht zu behalten, sollten Sie beim Aufzeichnen nur einen Abschnitt bearbeiten und nur wenige Optionen ändern
- Unter Umständen müssen Sie zwei Makros aufzeichnen – eines, das die Einstellungen einschaltet, und ein zweites, das sie ausschaltet – und den resultierenden Code vergleichen, um 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 der Zweck einer Prozedur es ist, eine Einstellung zu ändern.)

Hier als Beispiel dient die kleine Prozedur in Listing 5.3 bzw. Listing 5.4 (.NET), die die Einblendung der beiden »Mini-Symbolleisten« (Abbildung 5.4) unterdrückt. `ShowSelectionFloaties` entspricht dem

Kontrollkästchen *Minisymbolleiste für die Auswahl* in *Datei/Optionen/Allgemein*; für ShowMenuFloaties gibt es keine Option in der Benutzerschnittstelle.

Abbildg. 5.4 Eine »Mini-Symbolleiste« wird bei Markierung von Text oder per Rechtsklick darauf eingeblendet



Listing 5.3 Die »Mini-Symbolleisten« ausschalten

```
Sub MiniSymbolleistenAusschalten()
    'Bei Rechtsanklicken
    Application.Options.ShowMenuFloaties = False
    'Bei Markierung
    Application.Options.ShowSelectionFloaties = False
End Sub
```

Listing 5.4 (,NET): C#-Version: Eine bestehende Verbindung zur Word-Anwendung über wdApp wird angenommen



```
private void MiniSymbolleistenAusschalten_CS()
{
    //Bei Rechtsanklicken
    wdApp.Options.ShowMenuFloaties = false;
    //Bei Markierung
    wdApp.Options.ShowSelectionFloaties = false;
}
```

Default-
FilePath-
Option

Eine weitere wichtige Option entspricht den Einstellungen unter der Schaltfläche *Dateispeicherorte* in *Erweitert/Allgemein*. 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# muss die Methode `get_DefaultFilePath` eingesetzt werden, da C# Parameter für eine Eigenschaft nicht akzeptiert.

```
Word.WdDefaultFilePath sup = Word.WdDefaultFilePath.wdStartupPath;
string startupPfad = Word.Application.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>ClipArt-Grafiken</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.
wdProgramPath	9	Installationsort der Datei <i>winword.exe</i>
wdProofingToolsPath	12	Installationsort der *.lex-Dateien. Bis Word 97 befanden sich hier standardmäßig auch die *.dic-Dateien. Seit Word 2000 werden diese im Benutzerprofil gespeichert.
wdStartupPath	8	Entspricht dem Dateityp <i>AutoStart</i> im Dialogfeld. Vorlagen in diesem Ordner werden von Word automatisch beim Starten geladen. Ferner wird allen darin stehenden Makros automatisch vertraut.
wdStyleGalleryPath	15	Wo der Formatvorlagenkatalog die aufgelisteten Vorlagen findet. (Die Funktionalität steht seit Word 2002 nicht mehr in der Benutzerschnittstelle, ist aber im Objektmodell noch vorhanden.)
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. Seit 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 werden automatisch vertraut.
wdWorkgroupTemplatesPath	3	Entspricht dem Dateityp <i>Arbeitsgruppenvorlagen</i> im Dialogfeld. Speicherort, meistens im Netzwerk, von Vorlagen, die mehrere Benutzer teilen.

Word-
Basic

Der Kerncode der Word-Anwendung wurde in den späten achtziger Jahren geschrieben. Die damalige Philosophie war, dass Word möglichst anpassungsfähig sein sollte. Deshalb wurden die Schnittstellen zu allen internen Befehlen über die Programmiersprache WordBasic offen gelegt. Es war möglich, Dialogfeldeinstellungen zu lesen und festzulegen sowie einen internen Ablauf durch den eigenen Code zu ersetzen. Viele dieser Möglichkeiten sind im VBA-Zeitalter noch vorhanden und werden in Teil 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 5.5 bzw. in Listing 5.6, dessen Resultat in Abbildung 5.5 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, kann Ihnen das Hilfethema »Visual Basic-Entsprechungen zu WordBasic-Befehlen« gute Dienste leisten.

Tabelle 5.3

Nützliche *WordBasic*-Befehle

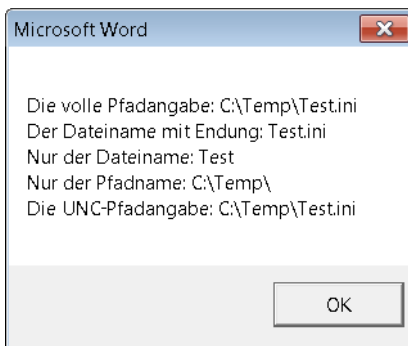
WordBasic-Befehl	Beschreibung
<code>WordBasic.SelectSimilarFormatting</code>	<p>In Word 2002 wurde eine Funktionalität eingeführt, die es dem Benutzer ermöglicht, nicht zusammenhängende Dokumentbereiche gleichzeitig zu markieren (<code>[Strg]</code>-Taste festhalten und mit der Maus markieren). Zudem 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.</p> <p>Für die letztere gibt es jedoch diesen Befehl im WordBasic-Bereich der Anwendung. Er sucht im Dokument weitere Vorkommen der Formatierung, in der sich die Einfügemarke gegenwärtig befindet, und markiert sie alle.</p>
<code>WordBasic.DisableAutoMacros</code>	<p>Die Word-Anwendung unterstützt seit jeher einen Satz »Auto-Makros«, die bei bestimmten Handlungen (z.B. Erstellen, Öffnen oder Schließen eines Dokuments) automatisch ausgeführt werden. Obwohl diese unter VBA noch laufen, müsste der Entwickler nach den Vorstellungen von Microsoft Dokument- und Application-Ereignisse einsetzen. Was in beiden Fällen in der VBA-Schnittstelle fehlt, ist die Möglichkeit, die Ausführung zu unterbinden*. Dieser WordBasic-Befehl reguliert die Ausführung von »Auto-Makros«.</p> <p>WordBasic.DisableAutoMacros 1 schaltet sie aus; WordBasic.DisableAutoMacros 0 lässt die Ausführung zu.</p>

Tabelle 5.3 Nützliche *WordBasic*-Befehle (Fortsetzung)

WordBasic-Befehl	Beschreibung
<code>WordBasic.SortArray aArray()</code>	Um per VBA etwas zu sortieren, müssen Sie selber die Funktionalität einbauen; es gibt dafür keine internen Befehle. Mit WordBasic.SortArray 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 Application.ActivePrinter den Drucker ändern, wird auch die standardmäßige Druckereinstellung des Systems geändert. Der WordBasic-Befehl ändert den Drucker in Word, ohne die Systemeinstellung zu ändern.
<code>WordBasic.FileNameInfo\$()</code>	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.
<code>WordBasic.MailMergeUseOutlookContacts</code>	Verbindet ein Seriendruckdokument mit den Outlook-Kontakten als Datenquelle (Neu in Word 2007)
<code>WordBasic.CreateCommonFieldBlockFromSel()</code>	Erstellt aus der Markierung einen Schnellbaustein, der auf dem Menüband »Einfügen/Schnellbaustein« angezeigt wird. Einziger möglicher Parameter ist Description zur Angabe einer Beschreibung (siehe in Kapitel 6 den Abschnitt »Schnellbausteine«).
* In Word 2002 wurde die Eigenschaft AutomationSecurity eingeführt, die alle Makros eines Dokuments beim Öffnen sperren kann (mehr dazu lesen Sie weiter oben unter Application).	

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) zum Herunterladen unter <http://www.microsoft.com/downloads/details.aspx?displaylang=en&FamilyID=1A24B2A7-31AE-4B7C-A377-45A8E2C70AB2> bereitgestellt und sie wurde zusätzlich auf der CD-ROM zum Buch im Ordner `\Beilagen\Word95 Wordbasic Hilfe` abgelegt.

Abbildg. 5.5 Ergebnisse der Funktion *WordBasic.FileNameInfo\$*


Listing 5.5 Mit der Funktion *WordBasic.FileNameInfo\$()* Pfadinformationen ermitteln

```

Sub PfadangabenInfoAuslisten()
    Dim strPfadangabe As String

    If Len(Dir(strPfadangabe)) <> 0 Then
        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 & "Der UNC-Pfadangabe: " & .FileNameInfo(strPfadangabe, 6)
        End With
    Else
        MsgBox "Dateipfad konnte nicht gefunden werden. " & _
            "Bitte passen Sie den Makrocode an."
    End If
End Sub

```

Listing 5.6 (.NET): In C# kann WordBasic nur über »Late Binding« (*GetType().GetMember()*) angesprochen werden

```

using System.Reflection;
private void Pfadangaben(string pfadangabe)
{
    object oWdBsc = InvokeHelper("WordBasic", GetProp, null, wdApp, null);
    object oFN = (object) pfadangabe;
    object oFI1;
    object oFI3;
    object oFI4;
    object oFI5;
    object oFI6;
    int dateiPfad = 1;
    int dateiNameMitEndung = 3;
    int dateiName = 4;
    int nurPfad = 5;
    int pfadUNC = 6;
    oFI1 = InvokeHelper("FileNameInfo$", InvMethod | GetProp, null, oWdBsc, oFN, dateiPfad);
    oFI3 = InvokeHelper("FileNameInfo$", InvMethod | GetProp, null, oWdBsc, oFN,
        dateiNameMitEndung);
    oFI4 = InvokeHelper("FileNameInfo$", InvMethod | GetProp, null, oWdBsc, oFN, dateiName);
    oFI5 = InvokeHelper("FileNameInfo$", InvMethod | GetProp, null, oWdBsc, oFN, nurPfad);
    oFI6 = InvokeHelper("FileNameInfo$", InvMethod | GetProp, null, oWdBsc, oFN, pfadUNC);
    MessageBox.Show(String.Format(
        "Die volle Pfadangabe: {0}\nDer Dateiname mit Endung: {1}\n" +
        "Nur der Dateiname: {2}\nNur der Pfadname: {3}\nDie UNC-Pfadangabe: {4}",
        oFI1.ToString(), oFI3.ToString(), oFI4.ToString(), oFI5.ToString(), oFI6.ToString()));
    oWdBsc = null;
}

private BindingFlags InvMethod = BindingFlags.InvokeMethod;
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);
}

```

CD-ROM Die Beispieldatei *Bsp05_01_App.docm* finden Sie auf der CD-ROM im Ordner *\Beispiele\Kap05*.

Die gegenwärtige Markierung: *Selection* und ähnliche Objekte

Während unserer Arbeit als MVPs in den Newsgroups 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 soll-

ten Sie einer Objektvariablen zuweisen, etwa: `Set dok = ActiveDocument` bzw. `Set win = ActiveWindow`. Beispielen für solche Zuweisungen werden Sie in den Listings dieses Buchs 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« ab Seite 207).

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
<code>wdSelectionBlock</code>	6	Ein Rechteck von Text wurde mit Festhalten der <code>[Alt]</code> -Taste markiert
<code>wdSelectionFrame</code>	3	Ein Positionsrahmen ist markiert
<code>wdSelectionIP</code>	1	Die Einfügemarke blinkt im Text
<code>wdSelectionRow</code>	5	Eine oder mehrere Tabellenzeilen sind markiert
<code>wdNoSelection</code>	0	(Die Autoren haben diesen Zustand (keine Markierung) nie feststellen können)
<code>wdSelectionColumn</code>	4	Eine oder mehrere Tabellenspalten sind markiert
<code>wdSelectionInlineShape</code>	7	Eine Grafik »mit Text in Zeile« ist markiert
<code>wdSelectionNormal</code>	2	Text (ein oder mehrere Zeichen) wurde normal markiert
<code>wdSelectionShape</code>	8	Eine mit Textfluss formatierte Grafik ist markiert

Select-Current 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« ab Seite 220) kann das, aber meistens erstreckt sich der gefundene Bereich nicht über mehrere Absätze hinweg. Die Methoden `SelectCurrentAlignment` (Absatzausrichtung), `SelectCurrentColor` (Schriftfarbe), `SelectCurrentFont` (Schriftart), `SelectCurrentIndent` (Absatzeinzug), `SelectCurrentSpacing` (Zeilenabstand), `SelectCurrentTabs` (Tabstopps) unterliegen dieser Beschränkung nicht. Die Markierung wird erweitert bis zu der Stelle, wo die bestimmte Formatierung aufhört.

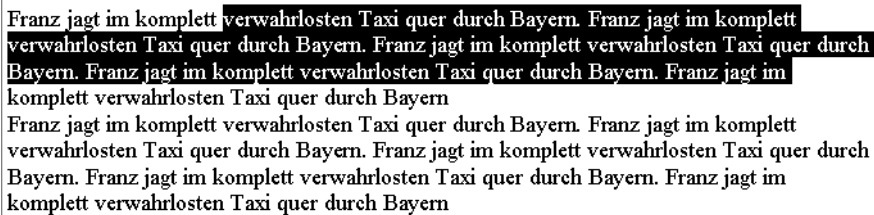
wdLine: zeilenweise Mit den verschiedenen `Move`-Methoden wird ein Bereich oder eine Markierung vergrößert oder verschoben. Über das Argument `Unit` wird festgelegt, um welche Einheit dies geschieht. Das Argument `Count` bestimmt, um wie viele Einheiten der Bereich verschoben oder erweitert wird. Gültige Werte für `Unit` sind `wdCharacter` (Zeichen), `wdWord` (Wort), `wdSentence` (Satz), `wdParagraph` (Absatz),

wdSection (Abschnitt), wdStory (Dokumentteil), wdCell (Zelle), wdColumn (Tabellenspalte), wdRow (Tabellenzeile) oder wdTable (Tabelle). Zusätzlich 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 (sodass sie sich über drei Zeilen erstreckt), wie in Abbildung 5.6 ersichtlich. Vor Ausführung der obigen Codezeile blinkte die Einfügemarke am Anfang der abgebildeten Markierung.

Abbildg. 5.6 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. 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
```



In C#:

```
Word.Document doc = WordApplication.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.docx")
Set docNeu = Documents.Add
```

Das C#-Beispiel fällt etwas länger aus, da in C# (vor Version 4.0) *alle* Argumente einer Methode oder Eigenschaft angegeben werden müssen, auch wenn sie als »optional« bezeichnet sind. Dazu verlangt die Schnittstelle zu COM, dass die optionalen Argumente in der Form *ref object* übergeben werden müssen. Das Listing 5.7 verwendet die Methoden *Open* und *Add* des Document-Objekts in Word 2007 und 2010.

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# 3.5 und früher hingegen müssen die Funktionsaufrufe genau mit den Definitionen der Objektbibliothek übereinstimmen.

Listing 5.7

(.NET): Beispiele für die Methoden *Open* und *Add* des *Document*-Objekts in Word 2003



```
private void btnDoks_Click(object sender, System.EventArgs e)
{
    string dateiName = System.IO.Directory.GetCurrentDirectory();
    System.IO.DirectoryInfo di = new System.IO.DirectoryInfo(dateiName);
    string pfaad = (di.Parent.Parent.FullName + "\\Bsp05 Test.doc");
    wd.Document doc = WordDokumentOeffnen_CS(wdApp, pfaad);
    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 anderen die dokumentspezifischen Einstellungen aus den *Word-Optionen*, wie etwa *Formulardaten als durch Trennzeichen getrennte Textdatei speichern* (SaveFormsData) aus der Kategorie *Erweitert*, Abschnitt *Genauigkeit beim Freigeben dieses Dokuments beibehalten*. 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« ab Seite 176.

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.

Compu-
teStatis-
tics

Um zu ermitteln, wie viele Wörter, Zeichen, Sätze, Zeilen oder Absätze sich in einem geöffneten Word-Dokument befinden, bietet das Objektmodell die Methode `ComputeStatistics`. Dabei kann der Rückgabewert End- und Fußnoten mit einbeziehen oder auch weglassen. Die Enumeration `WdStatistic` (Tabelle 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 (.docx-Datei) in eine Vorlage (.dotx-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 Es kommt auch vor, dass wir den Dateinamen, den Pfadnamen oder die vollständige Pfadangabe eines Dokuments herausfinden müssen. 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.docx"
```

Attached Template 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 *Vorlagen und Add-Ins*, das Sie über *Entwicklertools/Dokumentvorlage* erreichen. Im Objektmodell gibt die Eigenschaft AttachedTemplate diese Information als ein Template-Objekt (dieses Objekt wird im Abschnitt »Dokumentvorlagen: das Template-Objekt« ab Seite 202 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.dotm* 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.dotm* hergestellt, was AttachedTemplate widerspiegelt. Aber aufgepasst! Im Dialogfeld *Dokumentvorlagen 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 5.8 (C#-Version in Listing 5.9) 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 Normal-Vorlage, *Normal.dotm* 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.dotm* verbunden wird. Beim Speichern dieses neuen Dokuments erscheint automatisch das Dialogfeld *Speichern unter*, da es noch nie gespeichert wurde.

Listing 5.8 Ein Dokument für die Übermittlung außer Haus vorbereiten und mit der *Normal.dotm* verbinden

```
Sub DokFuerTransportVorbereiten()
    Dim doc As Word.Document
    Dim strTemplatePath As String
    Dim docNeu As Word.Document
```

Listing 5.8 Ein Dokument für die Übermittlung außer Haus vorbereiten und mit der *Normal.dotm* verbinden (Fortsetzung)

```

Set doc = Application.ActiveDocument
If doc.Type = wdTypeDocument Then
    DokMitNormalVerbinden doc
ElseIf doc.Type = wdTypeTemplate Then
    MsgBox "Das aktuelle Dokument ist eine Vorlage, und darf nicht außer Haus " & _
        "geschickt werden. Ein neues Dokument wird daraus erstellt. " & _
        "Bitte speichern und verschicken Sie dieses.", vbOKOnly + vbInformation
    strTemplatePath = doc.FullName
    Set docNeu = Application.Documents.Add(strTemplatePath)
    DokMitNormalVerbinden docNeu
End If
End Sub

Sub DokMitNormalVerbinden(doc As Word.Document)
    doc.AttachedTemplate = NormalTemplate
    'Wenn der Benutzer Speichern unter abbricht,
    'soll die Ausführung nicht abgebrochen werden.
    On Error Resume Next
    doc.Save
    MsgBox "Das Dokument ist mit der Normal Vorlage verbunden.", vbOKOnly + vbInformation
End Sub

```

Listing 5.9 (.NET): C#-Version, um ein Dokument mit der Vorlage *Normal.dotm* 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 {}
}

```

CD-ROM

Die Beispieldatei *Bsp05_01_Document.docm* finden Sie auf der CD-ROM im Ordner *\Beispiele\Kap05*.

Dokumente speichern

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).

SaveAs

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

ACHTUNG

Die Enumeration *WdSaveFormat* enthält einen Konstantwert *wdFormatDocument*. Es wäre naheliegend, diesen für Word 2007/2010-Dokumente zu verwenden, jedoch aufgepasst! Das Word-Objektmodell enthält diesen Wert seit der Einführung von VBA: *wdFormatDocument* speichert in das Word 97-2003-Dateiformat und wurde aus Gründen der Rückwärtskompatibilität *nicht* geändert. Benutzen Sie stattdessen besser *wdFormatDocument97*, *wdFormatDocumentDefault*, *wdFormatXMLDocument* bzw. *wdFormatXMLDocumentMacroEnabled*.

- 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.

Save

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, sodass 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 5.8 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.

Saved

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 5.10 bzw. Listing 5.11. 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 5.10 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 5.11 (.NET): Die *Saved*-Eigenschaft in C# wirft keine besonderen Probleme auf



```
private void DokSpeichern()
{
    object objMissing = System.Reflection.Missing.Value;
    wd.Document doc = wdApp.Documents.Add(ref objMissing, ref
        ref objMissing, ref objMissing);
    doc.Saved = false;
    while (!doc.Saved)
    {
        try
        {doc.Save();}
        catch {}
    }
}
```

PROFITIPP

Eine Alternative zu diesem Vorgang wäre, das Dialogfeld *Datei/Speichern unter* explizit durch den Code einzublenden. Dadurch kann die Benutzerhandlung direkt ausgewertet werden. Der Umgang mit den internen Dialogfeldern von Word ist in Kapitel 15 beschrieben.

CD-ROM

Die Beispieldatei *Bsp05_01_Document.docm* finden Sie auf der CD-ROM im Ordner *\Beispiele\Kap05*.

PDF &
XPS

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

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

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

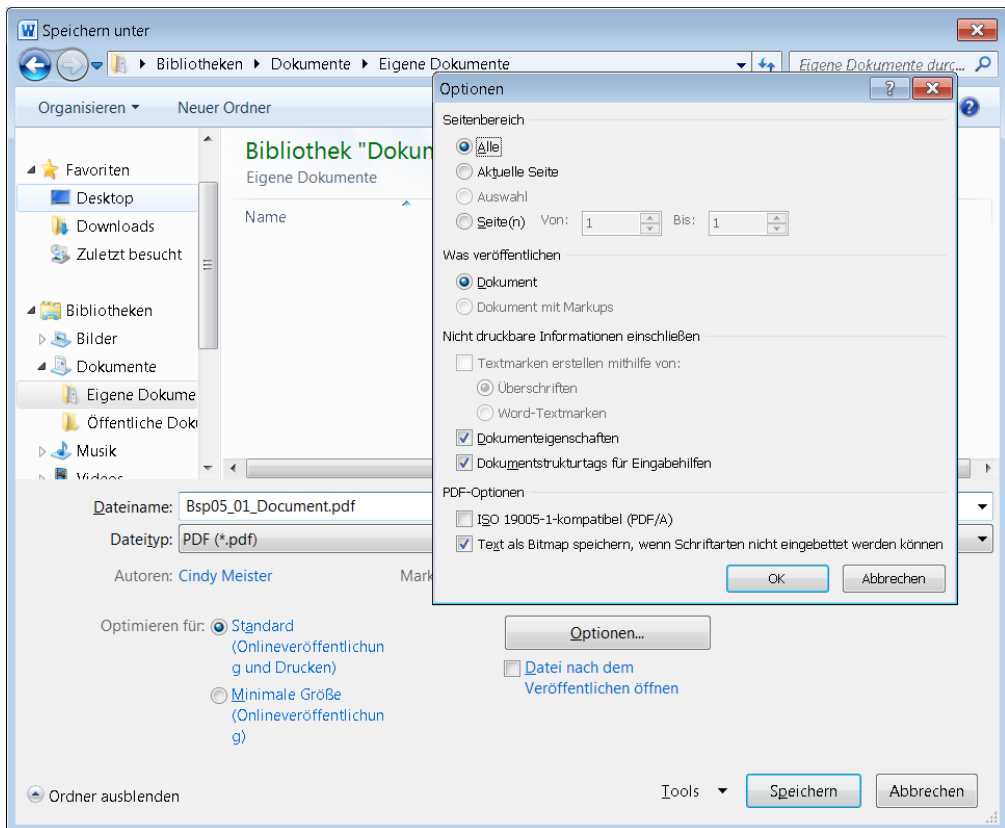
HINWEIS

Voraussetzung dazu für Word 2007 ist allerdings, dass das entsprechende Add-In »Add-In für 2007 Microsoft Office: »Speichern unter – PDF oder XPS« installiert wird. Sie können das Add-In nach erfolgreicher Gültigkeitsprüfung von der Webseite <http://www.microsoft.com/downloads/details.aspx?displaylang=de&FamilyID=4d951911-3e7e-4ae6-b059-a2e79ed87041>

herunterladen und installieren. Nach der Installation stehen die beiden neuen Dateitypen PDF und XPS unter Office 2007 zur Verfügung. Office 2010-Benutzern bleibt dieser Schritt erspart, da Microsoft sich inzwischen mit Adobe über die Verteilungsbedingungen einigen konnte.

Wenn Sie über den Menüpunkt *Speichern unter* das Dokument als PDF speichern möchten, werden Ihnen eine Reihe von Optionen angeboten, mit denen Sie den Export als PDF-Datei beeinflussen können, wie aus Abbildung 5.7 ersichtlich.

Abbildg. 5.7 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 5.12.

Listing 5.12 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
```

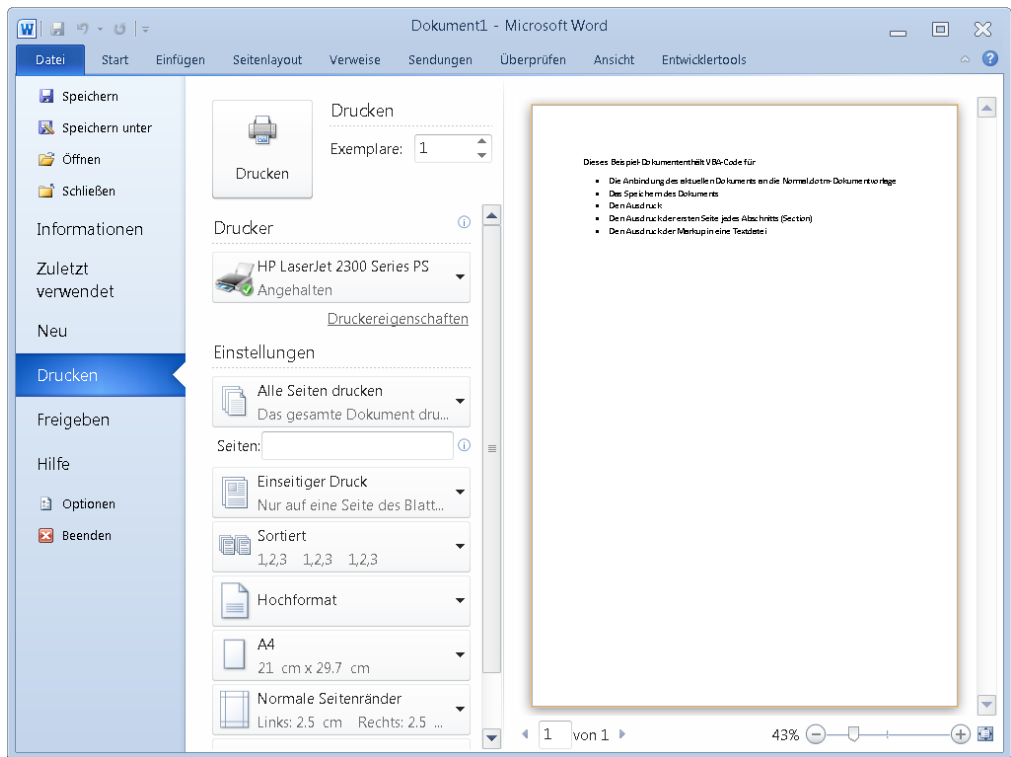
Die einzelnen Parameter der `Document.ExportAsFixedFormat`-Methode finden Sie in der Programmhilfe ausführlich erläutert, sodass an dieser Stelle nicht weiter darauf eingegangen wird.

Dokumente drucken mit der Methode *PrintOut*

Die `PrintOut`-Methode stellt den Großteil der Funktionalität der Office Backstage-Ansicht *Datei/Drucken* (Abbildung 5.8) (in Word 2007 der Menüeintrag *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. 5.8 Die Kategorie *Datei/Drucken*



Back-
ground

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 *Datei/Optionen/Erweitert*) im Abschnitt *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 jedem Fall raten wir dringlichst, den Druck im Hintergrund während des Codeablaufs zu unterbinden, indem Sie das Background-Argument oder die Option `PrintBackground` auf `False` festlegen.

In Listing 5.13 bzw. in Listing 5.14 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 Es gibt auch die Application-Eigenschaft `BackgroundPrintingStatus`. 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 5.13 Drucken im Hintergrund unterbinden. Die Anwender-Einstellung wird am Schluss wieder hergestellt.

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

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

Listing 5.14 (.NET): Der C#-Code für das Unterbinden des Druckens im Hintergrund



```
private void DokAusdrucken_CS(wd.Document doc)
{
    object objMissing = System.Reflection.Missing.Value;
    bool druckImHintergrund = wdApp.Options.PrintBackground;
    wdApp.Options.PrintBackground = false;
    DokDrucken(doc)
    object objSaveChanges = wd.WdSaveOptions.wdSaveChanges;
    wdApp.Options.PrintBackground= druckImHintergrund;
    doc.Close(ref objSaveChanges, ref objMissing, ref objMissing);
}
private void DokDrucken(wd.Document doc)
```

Listing 5.14 (.NET): Der C#-Code für das Unterbinden des Druckens im Hintergrund (*Fortsetzung*)


```
{
    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);
}
```

Range Diese Argumente beziehen sich auf die erste Dropdownliste unter *Einstellungen* unter *Drucken* sowie
 From das Textfeld *Seiten* (in Word 2007 entspricht dies dem Abschnitt *Seitenbereich* im Dialogfeld *Dru-*
 To *cken*). Ausschlaggebend, ob From und To oder Pages zu verwenden ist, ist die Einstellung für Range,
 Pages dessen mögliche Werte in Tabelle 5.7 aufgelistet sind.

Tabelle 5.7 Die *WdPrintOutRange*-Werte

WdPrintOutRange-Enum	Wert	Beschreibung
wdPrintAllDocument	0	Das gesamte Dokument; entspricht der Option <i>Alles</i> im Dialogfeld
wdPrintCurrentPage	2	Die aktuelle Seite
wdPrintFromTo	3	Die angegebenen Seiten von <i>n</i> bis <i>m</i> drucken
wdPrintRangeOfPages	4	Alle angegebenen Seiten drucken; entspricht dem Feld <i>Seiten</i> im Dialogfeld
wdPrintSelection	1	Den markierten Text ausdrucken

Wird Range auf *wdPrintFromTo* gesetzt, müssen den Argumenten From sowie To ganze Zahlen zugewiesen werden, die den Seitenzahlen im gegenwärtigen Dokument entsprechen. From und To werden ignoriert, wenn Range einen anderen Wert beträgt.

Das Argument Pages kommt nur zum Zug, wenn Range auf *wdPrintRangeOfPages* gesetzt wird. Es erwartet eine Zeichenkette, wie sie ins Feld *Seiten* einzugeben ist. Mehr Informationen zu diesem Thema finden Sie in Word 2010, wenn Sie den Mauszeiger über das Info-Symbol  stehen lassen. (In Word 2007 steht diese Information im Dialogfeld *Drucken*).

Ein Beispiel, wie die erste Seite jedes Dokumentabschnitts gedruckt wird, sehen Sie in Listing 5.15 bzw. Listing 5.16. Der Inhalt der Variable *strDruckbereich* für ein Dokument mit drei Abschnitten könnte so aussehen: "P1S1;P1S2;P1S3". 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 5.15 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
```

Listing 5.15 Als Druckbereich die erste Seite jedes Abschnitts festlegen (Fortsetzung)

```

For lAbschnittZaehler = 1 To lAnzahlAbschnitte
    strDruckbereich = strDruckbereich & "P1S" & 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 5.16 die PrintOut-Methode in eine getrennte Prozedur ausgelagert wird, und vergleichen Sie *DokDrucken* mit der Prozedur gleichen Namens in Listing 5.14. 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# 3.5 und früher ohne optionale Argumente aus. Wenn Sie häufig eine COM-Methode mit vielen Argumenten aufrufen müssen, lohnt es sich, eine solche »Bibliothek« anzulegen.

Listing 5.16 (.NET): Die C#-Version von Listing 5.15



```

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 Seitennummerangaben im Pages-Argument den Seitenzahlen im Dokument entsprechen, wie sie ausgedruckt werden. Dies ist vor allem wichtig, wenn das Dokument mehrere Abschnitte enthält. Wird durchnummeriert, hat die Verwendung von Seitenzahlen zusammen mit Abschnittsnummern keine Bedeutung. Dies ist nur sinnvoll, wenn in jedem Abschnitt die Nummerierung neu gestartet wird.

Item

Im unteren Teil dieser Dropdownliste befindet sich eine zweite Liste, die entspricht einer selten beachteten Dropdownliste im Dialogfeld *Drucken* vorheriger Word-Versionen. Diese Einträge entsprechen dem Argument *Item*. Sie bestimmen, was anstelle des Dokuments auszudrucken ist: AutoText-Einträge, Änderungen und Kommentare, Tastaturbelegungen oder Formatvorlagen. Die gültigen Werte sind in Tabelle 5.8 aufgelistet.

Dieses Argument ist nützlich, wenn Sie in einem Dokument die Änderungen verfolgen, möchten es jedoch ohne Markup ausdrucken. Standardmäßig wird das Dokument mit Markup gedruckt; der Benutzer muss die Einstellung jedes Mal anpassen. Leider ist diese Option nicht sofort sichtbar, und nicht schnell zu erreichen. Mit einem kleinen Makro gestaltet sich die Arbeit effizienter:

```
Sub DokumentOhneAenderungenDrucken()  
    ActiveDocument.PrintOut Item:=wdPrintDocumentContent  
End Sub
```

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
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 ein 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

PageType

In der zweiten Dropdownliste, die 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 Duplexdrucker verfügbar ist und trotzdem doppelseitig gedruckt werden soll.

Append
Output-
Filename
PrintTo-
File

Am Ende der Dropdownliste unter *Drucker* finden Sie den Befehl *Ausgabe in Datei umleiten*. (In Word 2007 befindet sich das Kontrollkästchen *Ausgabe in Datei* im oberen Teil des Dialogfelds *Drucken*). Seit Word in PDF-Format drucken kann, wird diese Option selten benutzt. Nur wenn der Anwender eine Liste, Formatvorlagen, AutoText-Einträge oder Tastaturbelegungen – was mit dem Argument *Item* bestimmt wird – bearbeiten möchte, findet sie noch Anwendung. Eine mit dieser Funktionalität erstellte Textdatei kann in Word oder einem Texteditor geöffnet werden.

Zuerst muss sichergestellt werden, dass die richtige Druckerart ausgewählt ist. Für das Erstellen einer Textdatei 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 *Geräte und Drucker*.
2. Führen Sie den Befehl *Drucker hinzufügen* aus.
3. Übernehmen Sie die Standardeinstellungen und klicken Sie auf *Weiter*, bis das Dialogfeld *Den Druckertreiber installieren* erscheint.
4. Aus der Liste *Hersteller* wählen Sie den Eintrag *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 5.17 bzw. das Listing 5.18 enthält ein Codebeispiel, das eine Liste der Markup im aktuellen Dokument in eine Textdatei ausgibt.

Listing 5.17 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 MarkupListeErstellen()
    Dim doc As Word.Document
    Dim strPfad As String
    Dim strAktuellerDrucker

    Set doc = ActiveDocument
    strPfad = "C:\test\MarkupListe " & 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:=wdPrintMarkup, Append:=False, _
        OutputFileName:=strPfad, PrintToFile:=True
    Application.ActivePrinter = strAktuellerDrucker
    Application.Documents.Open strPfad
End Sub
```

Listing 5.18 (NET): Aus Platzgründen wird die overloaded *DokDrucken*-Prozedur hier nicht nochmals aufgeführt



```
private void MarkupListeErstellen_CS()
{
    wd.Document doc = wdApp.ActiveDocument;
    System.DateTime dt = System.DateTime.Now;
    string pfad = String.Format("C:\\test\\MarkupListe_{0}{1}_{2}.prn",
        doc.Name, dt.ToString("MMMdd"), dt.ToString("hh.ss"));
    string aktuellerDrucker= wdApp.ActivePrinter;
    wdApp.ActivePrinter = "Generic / Text Only";
}
```

Listing 5.18 (NET): Aus Platzgründen wird die overloaded *DokDrucken*-Prozedur hier nicht nochmals aufgeführt (*Fortsetzung*)

```

    wdApp.Options.PrintBackground = false;
    bool append = false;
    wd.WdPrintOutItem printOutItem = wd.WdPrintOutItem.wdPrintMarkup;
    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 Druckeroptionen und -einstellungen. Einzig die Eigenschaft `Application.ActivePrinter` kann den ausgewählten Zieldrucker ändern. Dafür braucht sie den genauen Druckernamen, wie er auf dem Rechner definiert ist. Alle weiteren Einstellungen müssen über die Windows-API-Schnittstelle stattfinden. Weitere Informationen hierzu finden Sie im Artikel »Controlling the Printer from Word VBA« unter <http://pubs.logicalexpressions.com/Pub0009/LPMArticle.asp?ID=101>.

CD-ROM

Die Beispieldatei *Bsp05_01_Document.docm* 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.

Programmtechnisch wird ein neues Dokument durch die `Documents.Add`-Methode erstellt, die im Abschnitt »Der Kern der Sache: das *Document*-Objekt« ab Seite 188 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, sodass die gleiche Änderung in allen verbundenen Dokumenten vorgenommen wird. Die einzige Ausnahme bilden Formatvorlagen. Falls im Dialogfeld *Dokumentvorlagen und Add-Ins* das Kontrollkästchen *Dokumentformatvorlagen automatisch aktualisieren* (Abbildung 5.9) 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. Sie finden die Schaltfläche *Dokumentvorlage* auf der Registerkarte *Entwicklertools*.

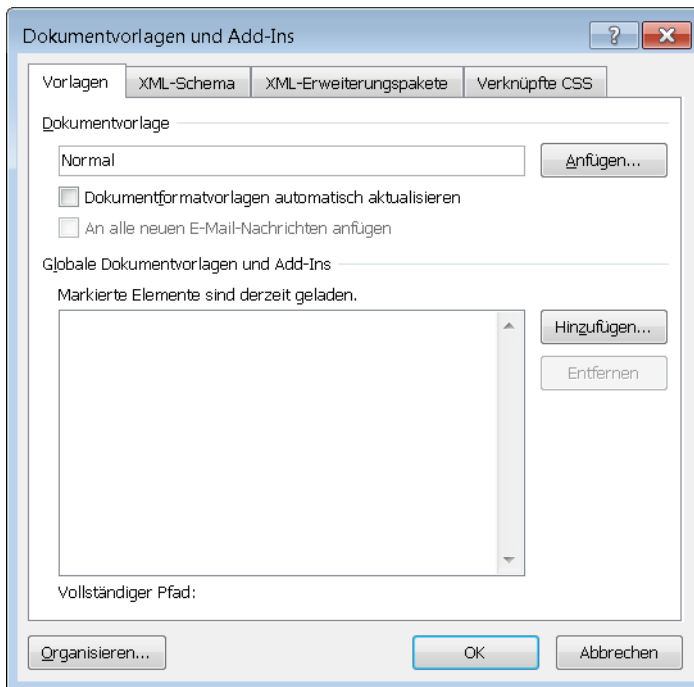
In zweiter Linie dienen Vorlagen als Behälter nützlicher Werkzeuge, die eine beliebige Kombination von Makros, Steuerelementen im Menüband und der QAT, Bausteinen und Tastaturbelegungen umfassen dürfen. In dieser Funktion gibt es zwei Arten von Vorlagen: die dokumentspezifische und das globale Add-In.

Die Werkzeuge einer dokumentspezifischen Vorlage sind nur für die damit verbundenen Dokumente sichtbar 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.dotm*. 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 *Dokumentvorlagen und Add-Ins* (Abbildung 5.9) geladen werden, indem man sie in der Liste aktiviert (oder hinzufügt und aktiviert).

Abbildg. 5.9 Dokumentenspezifische 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« ab Seite 188 mit einem Beispiel vorgestellt.

Für die *Normal.dotm* gibt es sogar ein eigenes Objekt: `NormalTemplate`. Als Beispiel dafür wird in Listing 5.19 über die Eigenschaft `Saved Word` vorgetäuscht, die Vorlage wurde nicht geändert und muss

folglich nicht gespeichert werden (siehe auch die Diskussion zu dieser Eigenschaft im Abschnitt »Der Kern der Sache: das *Document*-Objekt« ab Seite 188). Während der programmtechnischen Anpassung der Word-Anwendungsschnittstelle werden oft Änderungen in eine Vorlage geschrieben. Eventuelle Meldungen diesbezüglich an den Benutzer können mit dieser Eigenschaft unterdrückt werden.

Listing 5.19 Speicheraufforderungen der *Normal.dotm* unterbinden

```
Sub NormalAlsGespeichertBezeichnen()
    NormalTemplate.Saved = True
End Sub
```

Listing 5.20 (.NET): Die C#-Version



```
private void btnListing5_20_Click(object sender, System.EventArgs e)
{
    wdApp.NormalTemplate.Saved = true;
}
```

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 5.21 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« ab Seite 176) die Information zurück.

Listing 5.21 Ist eine Vorlage nicht vorhanden, kann sie in den Speicher als Add-In geladen werden

```
Sub MakroInAddinAusfuehren()
    Dim tmpl 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 tmpl In Application.Templates
        If tmpl.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
```




Dieses Konzept ist auch für den .NET-Entwickler wichtig, der die Benutzerschnittstelle anpassen will. Im Gegensatz zu Word 2003 und früher können Anpassungen im Menüband nicht extern vorgenommen werden; sie sind nur als Teil eines Add-ins oder Word-Dokuments erlaubt. Zudem können Tastaturbelegungen nur VBA-Makros zugewiesen werden (die ihrerseits .NET-Code anrufen). Für beide Fälle ist es naheliegend, eine Dokumentvorlage als Bestandteil der Lösung zu integrieren. Diese Vorlage wird von der .NET-Anwendung nach Bedarf geladen und wieder entfernt.

Listing 5.22 (.NET): Die C#-Version



```
private void Listing5_22_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);
}
```

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.dotm</i>

Das Word-Objektmodell enthält sonst nur wenige Eigenschaften für das Template-Objekt, hauptsächlich die Dokumenteigenschaften (DocumentProperties) und die Baustein- sowie AutoText-Einträge. Über das Document-Objekt kann zusätzlich auf Makros (mit der Run-Methode), Symbolleisten (über die CommandBars-Auflistung) und Tastaturbelegungen (über die KeyBindings-Auflistung) in einer Vorlage zugegriffen werden.

OpenAs-
Docu-
ment

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 Word-Anwendung 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 wechseln* in *Ansicht/Fenster* erscheinen und ansprechbar sein. Sie können notfalls mithilfe von Anwendungsereignissen den Zugriff verweigern (mehr zum Thema Ereignisse steht in Kapitel 8 beschrieben).

Listing 5.23 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 5.24 (.NET): Die C#-Version


```
private void Listing5_24_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);
}
```

CD-ROM

Die Beispieldatei *Bsp05_01_Template.doc* mit den Codebeispielen sowie *Bsp05_02_Template.dot* mit dem Makro »MakroAddin« 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 Dokument vorstellen, die unabhängig von der sichtbaren Markierung manipuliert werden kann. Es ist möglich, im Code mehrere Range-Bereiche gleichzeitig zu definieren und damit zu arbeiten.

Die Zeichen in einem Dokumentteil (Story) werden intern von 1 bis n durchnummeriert, und es ist möglich, einen Range anhand von Start- und Endpunktwerten zu definieren. Da der Inhalt eines Word-Dokuments jedoch ständig im Fluss ist und zudem nicht alle Zeichen sichtbar sind, ist diese Methode nicht frei von Risiken. Allgemein empfiehlt es sich, Alternativen zur Arbeit mit numerischen Werten zu suchen. Im Folgenden stellen wir Ihnen einige nützliche Arbeitsweisen vor.

Einen Bereich definieren

Range.
Select

Weil ein Range unsichtbar ist, hat man manchmal das Gefühl, die Arbeit damit gleicht einer Lotterie. Während der Programmentwicklung ist es hilfreich, den Code schrittweise auszuführen und die momentane Position des Bereichs anzuzeigen. Dies wird erreicht, indem man ihn mit der Select-Methode markiert und so im Dokument sichtbar macht:

```
rng.Select
```

Diese Methode wird auch genutzt, um die Markierung an die gewünschte Stelle zur Weiterbearbeitung durch den Benutzer zu setzen, bevor ihm die Kontrolle über das Dokument zurückgegeben wird.

Range
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 = WordApplication.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« ab Seite 220 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
Formatie-
rung

Sobald ein Range vorliegt, kann ihm Text zugewiesen und der Text formatiert werden:

```
rng.Text = "Dieser Text ist fett."
rng.Font.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 Punkt angeben
```

HINWEIS

Da die Syntax für Zeichen- und Absatzformatierungen problemlos mit dem Makrorekorder ermittelt werden kann, gehen wir hier nicht näher darauf ein. Die meisten Befehle dafür befinden sich auf der Registerkarte *Start*.

Collapse-Methode

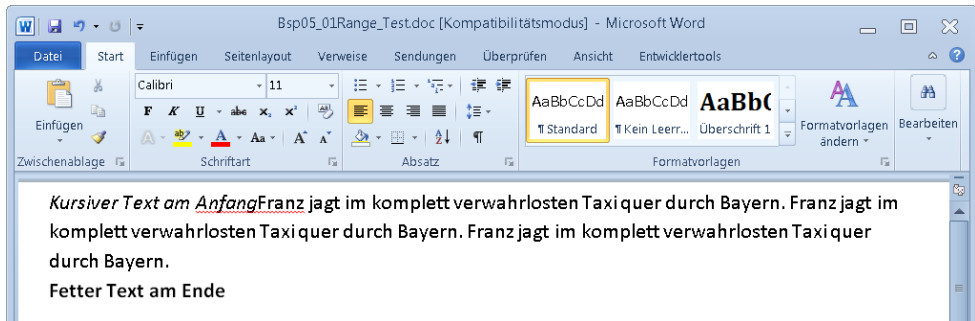
Wenn Sie ein Dokument, das schon Text enthält, öffnen und den Inhalt einem Range zuweisen, umfasst der Range den gesamten Inhalt des Dokument-Haupttextes (MainTextStory). Wird diesem Range dann Text zugewiesen, ersetzt er den im Dokument bereits vorhandenen Text.

Soll stattdessen der neue Text zusätzlich zum vorhandenen hinzugefügt und danach bearbeitet werden, sollte der Range zuerst auf einen Punkt verkleinert werden, wie in Listing 5.25 bzw. in Listing 5.26 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. 5.10

Das Resultat von Listing 5.25



Listing 5.25

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.docx"
    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
```

Listing 5.25 Den Bereich (*Range*) auf einen Punkt verkleinern, bevor Text eingefügt wird (*Fortsetzung*)

```
Set rng = doc.Range
'Der neue Text erscheint am Dokumentanfang
rng.Collapse wdCollapseStart
rng.Text = "Kursiver Text am Anfang"
rng.Italic = True
End Sub
```

Listing 5.26 (NET): Hier ist Code aus der Klasse für den Abschnitt über das *Document*-Objekt integriert, um das Dokument zu öffnen



```
private void Listing5_26_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.FullName + "\\Bsp05_01Range_Test.docx");
    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;
}
```

CD-ROM

Die Beispieldateien *Bsp05_01Range.docm* und *Bsp05_01Range_Test.docx* finden Sie auf der CD-ROM im Ordner *\Beispiele\Kap05*.

Insert-
Metho-
den

Es ist möglich, Text einem Bereich hinzuzufügen, ohne den Bereich vorab zu verkleinern. Es besteht jedoch keine Möglichkeit, auf diesen Text direkt wieder einzuwirken, um ihn beispielsweise zu formatieren; er »verschmilzt« mit dem im Bereich bestehenden Text. Zu diesem Zweck bietet das Word-Objektmodell einige Methoden an: *InsertAfter* (Text am Schluss des Bereichs einfügen), *InsertBefore* (Text am Anfang des Bereichs einfügen), *InsertBreak* (einen manuellen Umbruch (Wechsel) einfügen), *InsertParagraphAfter* (Absatz nach dem Bereich einfügen) und *InsertParagraphBefore* (Absatz vor dem Bereich einfügen).

```
rng.Text = "Text im Bereich."
rng.InsertAfter " Text dem Bereichende anfügen."
MsgBox rng.Text
'Ergebnis: Text im Bereich. Text dem Bereichende anfügen.
```

Die wichtigste dieser Methoden (weil es dafür keine Alternative gibt) ist *InsertBreak*. Damit werden Seiten- sowie Abschnittswchsel ins Dokument eingefügt. Ein Seitenwechsel ersetzt den Bereichsinhalt; ein Abschnittswchsel wird *vor* dem Bereich eingefügt und der Bereich auf den Punkt *zwischen* dem eingefügten Wechsel und dem ursprünglichen Bereich verkleinert (mehr über Abschnitte und

Abschnittswechsel lesen Sie in Kapitel 6). Um beispielsweise einen Abschnittswechsel 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-Metho-
den

Ein Bereich kann schrittweise verkleinert oder erweitert werden. Dazu stehen mehrere Move-Methoden zur Verfügung: Move, MoveEnd, MoveEndUntil, MoveEndWhile, MoveStart, MoveStartWhile, MoveStartUntil, MoveUntil und MoveWhile.

Für Move, MoveEnd und MoveStart wird über das Argument Unit festgelegt, um welche Einheit dies geschieht. Das Argument Count bestimmt, um wie viele Einheiten der Bereich verschoben oder erweitert wird. Gültige Werte für Unit sind wdCharacter (Zeichen), wdWord (Wort), wdSentence (Satz), wdParagraph (Absatz), wdSection (Abschnitt), wdStory (Dokumentteil), wdCell (Zelle), wdColumn (Tabellenspalte), wdRow (Tabellenzeile) oder wdTable (Tabelle).

Um einen Bereich um einen Absatz in Richtung des Dokumentanfangs zu erweitern:

```
rng.MoveStart wdUnit:=wdParagraph, Count:=-1
```



In C#:

```
object objWordUnit = (object) wd.WdUnits.wdWord;
object objCountNeg1 = -1;
rng.MoveStart(ref objWordUnit, ref objCountNeg1);
```

ACHTUNG

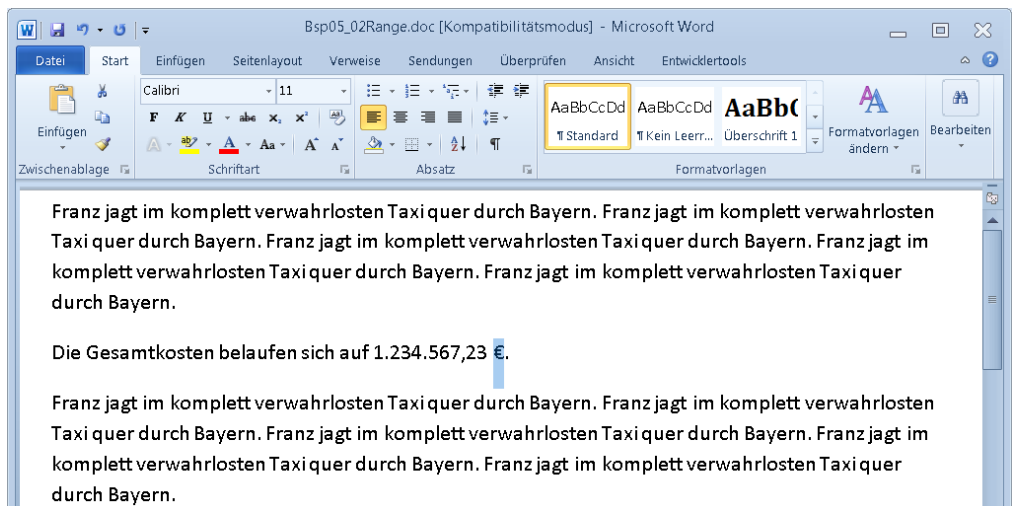
Denken Sie daran: Wenn eine MoveEnd-Methode mit einem positiven Unit-Wert verwendet wird, wird der Bereich erweitert; mit einem negativen Wert wird er verkleinert. Überschneiden sich dabei die End- und Startpunkte, wird der Bereich auf einen Punkt verkleinert und bleibt ein Punkt.

Für den Startpunkt ist es genau umgekehrt: Negative Unit-Werte erweitern den Bereich; positive verkleinern ihn, bis er zum Punkt wird.

Die Methoden MoveEndUntil, MoveEndWhile, MoveStartWhile, MoveStartUntil, MoveUntil und MoveWhile ermöglichen eine bedingte Anpassung des Bereichsumfangs. Sie nehmen zwei Argumente: CSet sowie Count. CSet besteht aus einer Liste von Textzeichen und legt die Bedingung fest. Der Anfangs- bzw. Endpunkt wird verschoben, bis eines der Zeichen in der Liste angetroffen wird (»until«) bzw. bis keines der Zeichen angetroffen wird (»while«). Count ist fakultativ (»optional«) und 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 5.11). Jetzt soll, wie in Listing 5.27 vorgestellt, die voranstehende Zahl (falls vorhanden) auch in den Bereich aufgenommen werden.

Abbildg. 5.11 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

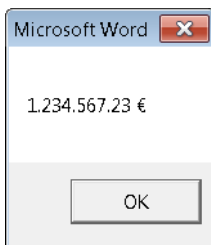
zwei Schritten angepasst. Zuerst wird er um maximal eine Stelle erweitert, und zwar nur dann, wenn dieses Zeichen ein Leerzeichen ist.

Danach wird frei gegen Dokumentanfang erweitert, solange eine Ziffer, ein Komma (Dezimaltrennzeichen) oder ein Punkt (Tausendertrennzeichen) vorliegt. Am Schluss enthält der Bereich das Währungssymbol und die Zahl (Abbildung 5.12).

PROFITIPP

Dieses Beispiel kann dynamischer gestalten werden, um Trennzeichen, Gruppierungssymbol und Währungssymbol der gegenwärtigen Windows-Einstellungen einzusetzen. Diese sind im Windows-Registry festgehalten und können mit der Funktion `PrivateProfileString` ermittelt werden. Siehe dazu den Abschnitt »Das System-Objekt« ab Seite 173.

Abbildg. 5.12 ... und der Bereich mit der *MoveStartWhile*-Methode erweitert, bis er die ganze Zahl enthält



Listing 5.27 Einen Bereich bedingt erweitern

```
Sub ZahlPlusWaehrungssymbol()
    Dim rng As Word.Range
    'Markieren Sie zuerst das Euro-Symbol im Beispieltext
    Set rng = Selection.Range
    rng.MoveStartWhile CSet:= " ", Count:=-1
    rng.MoveStartWhile CSet:="1234567890.", Count:=wdBackward
    MsgBox rng.Text
End Sub
```

Listing 5.28 (.NET): Die C#-Version



```
private void ZahlPlusWaehrungssymbol_CS()
{
    //Markieren Sie zuerst ein Zeichen rechts neben einer Zahl
    wd.Range rng = wdApp.Selection.Range;
    object objCSet = (object) " ";
    object objCount = (object) -1;
    rng.MoveStartWhile(ref objCSet, ref objCount);
    objCSet="1234567890.";
    objCount= (object) wd.WdConstants.wdBackward;
    rng.MoveStartWhile(ref objCSet, ref objCount);
    MessageBox.Show(rng.Text);
}
```

CD-ROM

Die Beispieldatei *Bsp05_02Range.docm* finden Sie auf der CD-ROM im Ordner *\Beispiele\Kap05*.

GoTo-
Methode

Die GoTo-Methode entspricht in etwa dem Dialogfeld *Bearbeiten/Gehe zu*, das Sie über die Schaltfläche *Suchen*, die sich in der Gruppe *Bearbeiten* der Registerkarte *Start* befindet, aufrufen. 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 Dinge, die im Objektmodell kein direktes Gegenstück haben, wie etwa Seiten. In Word 2003 wurde das Objektmodell zwar um ein Page-Objekt ergänzt, dieses dient jedoch lediglich 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 bekommen: `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« ab Seite 186.

Informa-
tion

Das Word-Objektmodell stellt die Eigenschaft `Information` für die `Selection`- und `Range`-Objekte zur Verfügung. Diese ist etwas eigenartig, da sie eher einem Sammelsurium von Funktionen als einer Eigenschaft gleicht. Der Grund dafür liegt in der Urgeschichte der Anwendung, in der Word-Basic-Sprache. Um die einigermaßen problemlose Konvertierung von WordBasic-Makros in VBA zu ermöglichen, wurden viele Konstruktionen beibehalten, u.a. die von der Funktion `SelInfo()` – neu als `Information`-Eigenschaft – gelieferten Auskünfte.

Beim Aufruf dieser Eigenschaft müssen Sie ein Argument des Typs `wdInformation` übergeben, das festlegt, welche Art von Auskunft erfragt wird. Im Gegensatz zu vielen Enumerationen sind diese in der VBA-Hilfe gut beschrieben, weshalb wir hier auf eine Auflistung verzichten.

Nehmen wir als Beispiel ein oft eingesetztes Argument: `wdWithinTable`. Damit wird festgestellt, ob sich der Bereich oder die Markierung innerhalb einer Tabelle befindet:

```
' Wenn »wahr«, befindet sich der Bereich innerhalb einer Tabelle
If rng.Information(wdWithinTable) Then
```



In C#:

```
//Wenn »wahr«, befindet sich der Bereich innerhalb einer Tabelle
if (rng.get_Information(wd.WdInformation.wdWithInTable))
```

Weitere nützliche Argumente sind: `wdHorizontalPositionRelativeToPage`, `wdHorizontalPositionRelativeToTextBoundary`, `wdVerticalPositionRelativeToPage` und `wdVerticalPositionRelativeToTextBoundary`. Sie bieten die einzige Möglichkeit, herauszufinden, wo sich der Bereich oder die Markierung waagrecht und senkrecht auf der Seite befindet. Die Angabe erfolgt in Punkten (Points).

HINWEIS Suchen Sie die Bildschirmkoordinaten einer Markierung oder eines Bereichs, schlagen Sie `GetPoint` in der allgemeinen VB-Hilfe nach.

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 (WordApplication.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.

CD-ROM

Die Beispieldatei *Bsp05_02Range.docm* 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« ab Seite 220 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?

Text-
Retrieval-
Mode

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 5.29 enthält eine Prozedur, die die Wirkung der verschiedenen Einstellungen demonstriert. Wird sie auf das Dokument in Abbildung 5.13 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`

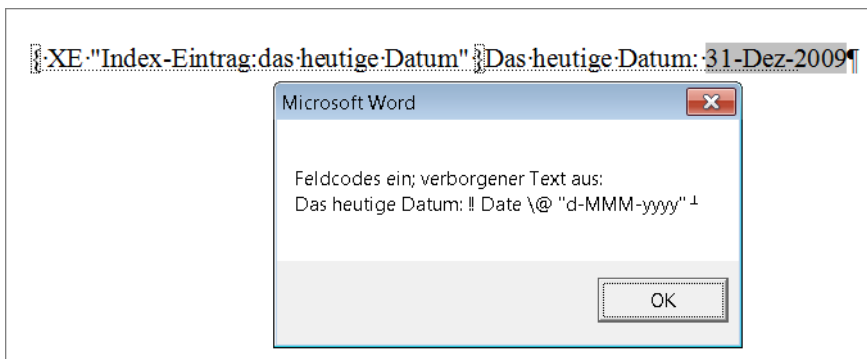
HINWEIS Die Zeichen »!!« und »!« in Abbildung 5.13 entsprechen den Feldklammern und geben die ANSI-Zeichen 19 bzw. 21 zurück. Diese Zeichen sind aber nur der »sichtbare« Teil der Feldcodeklammer. Im Hintergrund enthalten sie noch viel mehr Information. Es ist also nicht möglich, durch Einfügen dieser 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 5.29 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. 5.13 *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.

Format-
tedText

`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 (auch in einem anderen Dokument) zu kopieren, bietet das Word-Objektmodell die Eigenschaft `FormattedText` an. Damit können Sie schnell und bequem Text kopieren, ohne die Zwischenablage zu beanspruchen.

Listing 5.30 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
```

CD-ROM

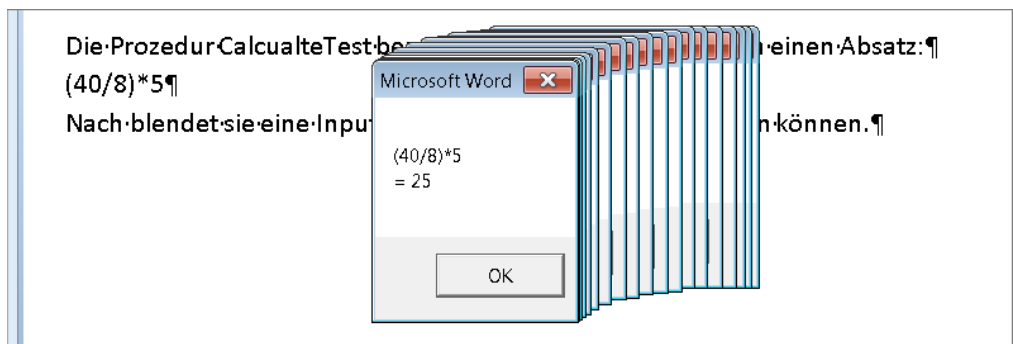
Die Beispieldatei *Bsp05_03Range.docm* finden Sie auf der CD-ROM (siehe Einleitung) im Ordner `\Beispiele\Kap05`.

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 for Applications 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 5.31 zeigt, wie es geht, und in Abbildung 5.14 sehen Sie einen Teil des Ergebnisses.

Abbildg. 5.14 Das passiert, wenn *ScreenUpdating* auf »falsch« gesetzt und ein Meldungsfeld verschoben wird



Listing 5.31 Mit Formeln im Text rechnen

```
Sub CalculateTest()
    Dim rng As Word.Range
    Dim strFormel As String
```

Listing 5.31 Mit Formeln im Text rechnen (Fortsetzung)

```

Application.ScreenUpdating = False
Set rng = ActiveDocument.Content.Paragraphs(5).Range
MsgBox rng.Text & " = " & rng.Calculate
rng.Collapse Direction:=wdCollapseEnd
'Den Benutzer zur Eingabe einer Formel auffordern
strFormel = InputBox("Bitte eine Formel eingeben:")
'Abbrechen, wenn nichts eingegeben wurde
If Len(strFormel) = 0 Then
    MsgBox "Keine Formel wurde eingegeben!", vbOKOnly
    Exit Sub
End If
'Diese am Ende des Bereichs einfügen
rng.Text = strFormel
'Die Berechnung ausführen
MsgBox "Das Ergebnis ist: " & rng.Calculate
'Die Formel wieder entfernen
rng.Delete
End Sub

```

Listing 5.32 (NET): Eine Textbox in einer Windows-Form wird für die Benutzereingabe eingeblendet



```

wd.Document docCalculateTest = null;
private void CalculateTest_CS()
{
    object objFalse = false;
    object objMissing = System.Type.Missing;

    try
    {
        wdApp.ScreenUpdating = false;
        string dateiName = System.IO.Directory.GetCurrentDirectory();
        System.IO.DirectoryInfo di = new System.IO.DirectoryInfo(dateiName);
        string pfad = (di.Parent.Parent.FullName + "\\Bsp05_04Range_Test.docx");
        docCalculateTest = Kap05_CS.Kap05Document.WordDokumentOeffnen_CS(wdApp, pfad,
false);
        wd.Range rng = docCalculateTest.Content.Paragraphs[2].Range;
        MessageBox.Show("Ergebnis der Formel im unsichtbaren Dokument: " + rng.Text + " = " +
rng.Calculate());
        object objCollapseEnd = wd.WdCollapseDirection.wdCollapseEnd;
        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;

```

Listing 5.32 (NET): Eine Textbox in einer Windows-Form wird für die Benutzereingabe eingeblendet (Fortsetzung)

```

        wd.Range rng = docCalculateTest.Content.Paragraphs[1].Range;
        object objCollapseEnd = wd.WdCollapseDirection.wdCollapseEnd;
        rng.Collapse(ref objCollapseEnd);
        string formel = txtFormel.Text;
        //Diese am Ende des Bereichs einfügen
        rng.Text = formel;
        //Die Berechnung ausführen
        MessageBox.Show("Das Ergebnis ist: " + rng.Calculate());
        //Die Formel wieder entfernen
        object objMissing = System.Reflection.Missing.Value;
        rng.Delete(ref objMissing, ref objMissing);
    }
    finally
    {
        lblFormel.Visible = false;
        txtFormel.Visible = false;
        btnFormel.Visible = false;
        if (!(docCalculateTest == null))
        {
            //Test Dokument einblenden, zufoerst bringen
            docCalculateTest.ActiveWindow.Visible = true;
            docCalculateTest.ActiveWindow.Activate();
            //Aufforderung zum Speichern unterbinden
            docCalculateTest.Saved = true;
            docCalculateTest = null;
        }
        wdApp.ScreenUpdating = true;
    }
}

```

CD-ROM

Die Beispieldatei *Bsp05_04Range.docm* finden Sie auf der CD-ROM 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 dessen Automatisierung vorstellen.

HINWEIS

Die Funktion *Suchen und Ersetzen* in Word kann viel mehr als nur Zeichenketten im Text aufspüren, und diese, wenn gewünscht, mit einer anderen ersetzen. Es können auch Formatierungen gesucht und ersetzt sowie mit Platzhalterzeichen (die Regular Expressions (RegEx) ähnlich, aber nicht gleich sind) gearbeitet werden. Da eine Diskussion der Funktionalität der Benutzerschnittstelle die Grenzen dieses Buches sprengen würde, haben wir umfangreiche Informationen in der Datei *SuchenErsetzen.pdf* auf der CD-ROM zum Buch im Ordner *\Beilagen\SuchenErsetzen* bereitgestellt. Falls Sie mit den Möglichkeiten nicht schon vertraut sind, lohnt es sich, einen Blick darauf zu werfen, da überraschend viele Aufgaben ohne zusätzliches Programmieren lösbar sind.

Vor- und Nachteile des Makrorekorderergebnisses

Im Allgemeinen liefert der Makrorekorder einen brauchbaren Code für die *Suchen und Ersetzen*-Funktionalität. Ein Beispiel hierfür sehen Sie in Listing 5.33. Mit nur wenigen Ergänzungen kann das Resultat problemlos als Automatisierungscode eingesetzt werden. Viele der Argumente stimmen mit den englischen Bezeichnungen der Dialogfeld-Steuerelemente überein, sodass es relativ einfach ist, die Handlung nachzuvollziehen. Die deutsche Version des Dialogfelds sehen Sie in Abbildung 5.16, die Tabelle 5.11 bietet eine Übersicht der entsprechenden englischen Begriffe.

Ab Word 2010 wird dieses Dialogfeld nicht mehr direkt über die Schaltfläche *Start/Bearbeiten/Suchen* oder die Tastenkombination **[Strg] F** eingeblendet. Dieser Befehl aktiviert den neuen Aufgabenbereich *Navigation* (Abbildung 5.15). Um an das Dialogfeld zu kommen, muss *Erweiterte Suche* aus dem vollen Menü der Schaltfläche gewählt werden.

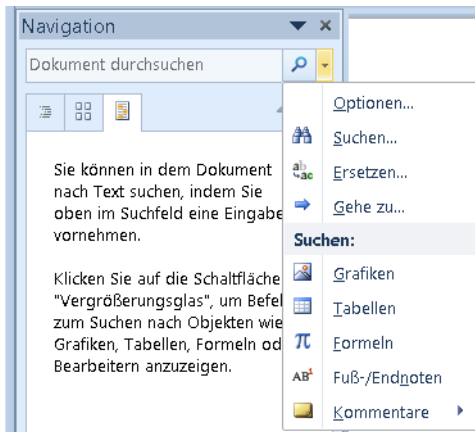
HINWEIS

Falls Sie das Dialogfeld regelmäßig einsetzen möchten, lohnt es sich, ein Makro mit folgender Codezeile der Symbolleiste für den Schnellzugriff oder es einer Tastaturkürzel zuzuweisen.

```
Dialogs(wdDialogEditFind).Show
```



Abbildg. 5.15 Die erweiterten Optionen des neuen Aufgabenbereichs Navigation



Abbildg. 5.16 Das Dialogfeld veranschaulicht einen Großteil der Funktionalität, die auch dem Entwickler zur Verfügung steht

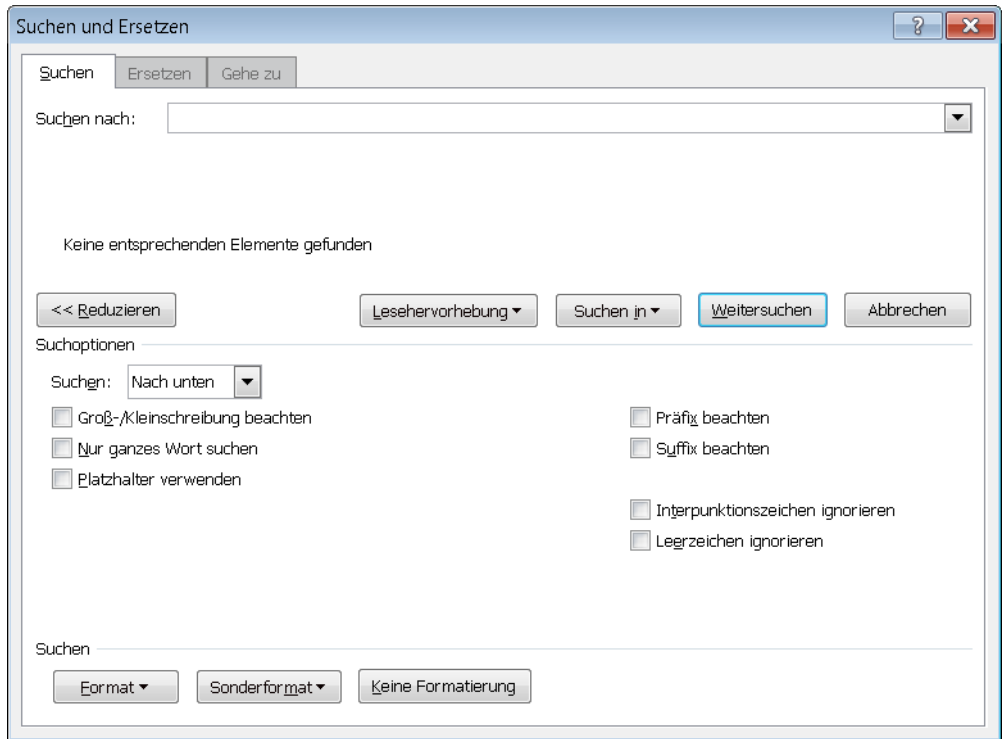


Tabelle 5.11 Übersicht der Parameter für das *Suchen und Ersetzen* mit VBA

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 beachten</i>
MatchWholeWord	<i>Nur ganzes Wort suchen</i>
MatchWildcards	<i>Platzhalter verwenden</i>
MatchSoundsLike	<i>Ähnl. Schreibweise (Englisch)</i>
MatchAllWordForms	<i>Alle Wortformen suchen (Englisch)</i>
ClearFormatting	<i>Keine Formatierung</i>
MatchPrefix	<i>*Präfix beachten</i>

Tabelle 5.11 Übersicht der Parameter für das *Suchen und Ersetzen* mit VBA (Fortsetzung)

Parameter	Beschriftung in der deutschen Umgebung
MatchSuffix	* <i>Suffix beachten</i>
IgnorePunct	* <i>Interpunktionszeichen ignorieren</i>
IgnoreSpace	* <i>Leerzeichen ignorieren</i>
HitHighlight bzw. ClearHighlight	** <i>Lesehervorhebung</i> Diese Methode kann anstelle der Execute -Methode verwendet werden, um alle gefundenen Stellen hervorzuheben
* Neu in Word 2007	
** Steht in Word 2010 in der Benutzerschnittstelle nicht zur Verfügung, da die Funktionalität sich neu im Aufgabenbereich befindet.	

Listing 5.33 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 5.33 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 in der Dropdownliste *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 5.33 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.

Formatierungen suchen und ersetzen

Es gibt einen Bereich, wo der Makrorekorder gelegentlich 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 5.17 mit dem Code in Listing 5.34. Die fünf Zeilen mit dem Vermerk »hinzugefügt« wurden vom Makrorekorder nicht aufgezeichnet. In einem solchen Fall müssen Sie die Anweisungen `.Font.Bold`, `.Font.Size` usw. selber eingeben.

Abbildg. 5.17 Diese Kriterien für die Schriftformatierung werden vom Makrorekorder nicht aufgezeichnet

Listing 5.34 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 = wdColorRed '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
```

Seit Word 2007 zeichnet der Makrorekorder deutlich mehr auf als in Word 2002 oder Word 2003. Zudem ist der Makrocode strukturierter. Hoffentlich müssen Sie bei der Arbeit mit einer neuen Version ihren aufgezeichneten Code nicht wesentlich korrigieren.

Anpassung des aufgezeichneten Codes

Ein aufgezeichnetes Makro muss für einfache Aufgaben oft nicht oder nur unwesentlich geändert werden. Für komplexere Anwendungen hingegen sind gewisse Anpassungen notwendig, wenn beispielsweise die Suche durch andere Handlungen unterbrochen wird oder wenn die Suche in mehreren Dokumentteilen (StoryRanges) durchgeführt werden soll.

In diesem Abschnitt stellen wir einige der meist gebrauchten Szenarien vor.

Suche mit einer Handlung unterbrechen (in einer Schleife ausführen)

Wie schon häufiger im Buch erwähnt, soll das `Selection`-Objekt möglichst gemieden und durch das `Range`-Objekt ersetzt werden. Diese Regel gilt grundsätzlich auch für die *Suchen und Ersetzen*-Funktion, ist aber für ein »reines« Suchen oder Ersetzen (»alles Ersetzen«) weniger zwingend. Für die folgenden Aufgaben ist ein Beibehalten des `Selection`-Objekts sinnvoll bzw. unbedenklich:

- Wenn die Suche die Markierung im Dokument verschiebt, weil der Anwender an der gefundenen Stelle eine Handlung ausführen soll
- Wenn alle Vorkommen des Suchbegriffs durch den Ersatzbegriff ersetzt werden

Range.
Find

Das `Range`-Objekt wird also hauptsächlich gebraucht, wenn als Ziel einer Prozedur bei jedem gefundenen Vorkommen eines Suchbegriffs eine Handlung ausgeführt werden soll, die mit der *Ersetzen*-Funktionalität nicht erreichbar ist. Nehmen wir als Beispiel ein Dokument, in welches der Benutzer

einige vordefinierte Begriffe eingibt. Unsere Anwendung sucht diese und fügt, anhand des letzten Wortes, einen AutoText-Eintrag ein. Es könnte natürlich etwas viel Aufwändigeres sein, z.B. die Erstellung einer Tabelle mit aktuellen Daten aus einer Datenbank. Aber unsere Beispiele sollen überschaubar bleiben ...

Dies bedeutet, dass die Suche nach jeder Handlung erneut aufgenommen werden muss. Wird mit dem `Select ion`-Objekt gearbeitet, 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.

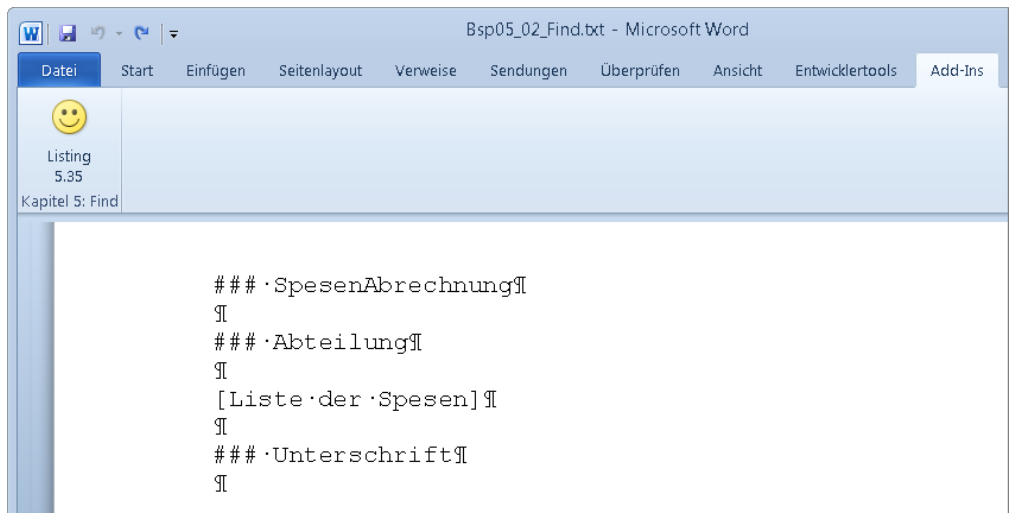
Ähnliches gilt auch für die Suche mit dem `Range`-Objekt, aber

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

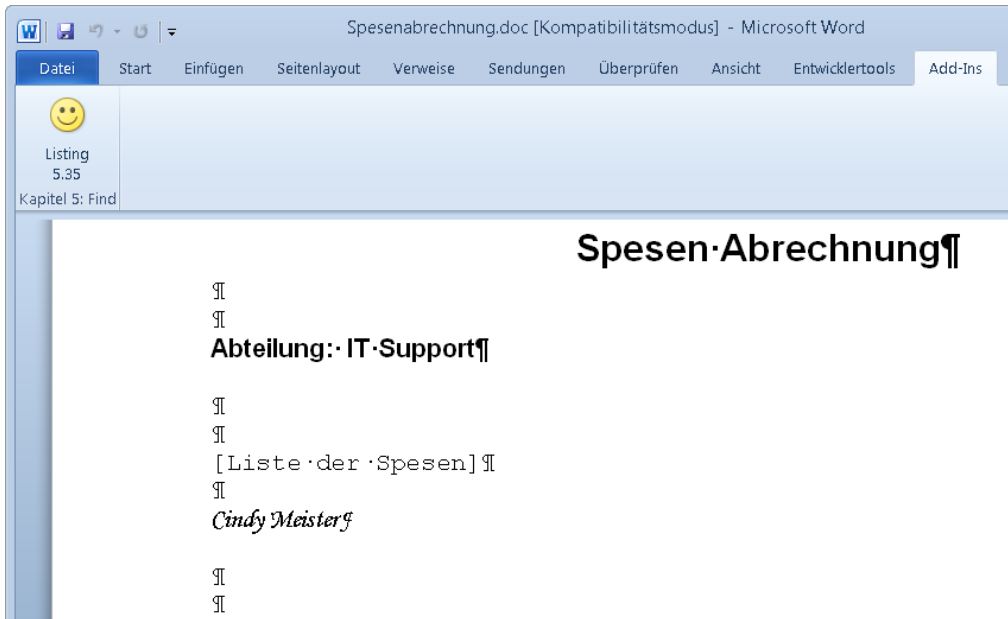
Der Hauptunterschied hier ist, dass die Weitersuche sich auf den *gefundenen Bereich* beschränkt, was meistens nicht erwünscht ist. Folglich muss vor Weiterführung der Suche der Bereich neu gesetzt werden.

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 5.18) 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 5.19 dargestellt.

Abbildg. 5.18 Eine einfache Spesenabrechnung. Den zu ersetzenden Begriffen stehen die Zeichen `###` voran.



Abbildg. 5.19 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. 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-Eigen- Danach wird Find mit dem Range-Objekt rngErgebnis ausgeführt. Sie sehen, dass dieses nicht anders schaft 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.dotm* 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 *Entwicklertools/Code/Makros* finden werden. (Die Datei *Bsp05_01_Find.dot*, die die AutoText-Einträge enthält, muss sich im gleichen Ordner mit *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 tmp1AutoTextBehaelter As Word.Template
    Dim bFound As Boolean

    Set rngSuchBereich = doc.Content
    Set rngErgebnis = rngSuchBereich.Duplicate
    Set tmp1AutoTextBehaelter = doc.AttachedTemplate

    AllgemeineSuchkriterienFestlegen rngErgebnis
    Do
        With rngErgebnis.Find
            .Text = strSuchBegriff
            .Forward = True
            .Wrap = wdFindStop
            .Execute
            bFound = .Found
        End With
    Loop
```


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

```

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-Eintrags
    'Falls der AutoText-Eintrag nicht vorhanden ist, einfach fortfahren
    On Error Resume Next
    Set rngErgebnis = tmp1AutoTextBehaelter.AutoTextEntries( _
        rngErgebnis.Text).Insert(Where:=rngErgebnis, RichText:=True)
    On Error GoTo 0
    rngErgebnis.End = rngSuchBereich.End
    lEintraegeZaehler = lEintraegeZaehler + 1
End If
Loop While bFound
BenutzerInfoEinfuegen = lEintraegeZaehler
End Function

Private Sub AllgemeineSuchkriterienFestlegen(ByRef rng As Word.Range)
    With rng.Find
        'Weitere Einstellungen wurden aus Platzgründen weggelassen.
        .ClearFormatting
        .MatchCase = False
        .MatchWholeWord = True
    End With
End Sub

```

Listing 5.36 (.NET): 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 = "### ";
    object spesenVorlage = (object) doc.Path + "\\Bsp05_01_Find.dotm";
    object objMissing = System.Reflection.Missing.Value;
    int anzEintraege = 0;
    if (doc.Type == wd.WdDocumentType.wdTypeDocument)
    {
        //Die Benutzer-spezifische Vorlage der Spesenrechnung anhängen
        doc.set AttachedTemplate(ref spesenVorlage);
        //Die mit AutoText zu ersetzenden Einträge suchen
        anzEintraege = BenutzerInfoEinfuegen(doc, suchBegriff);
    }
    //Den Benutzer auffordern, das Dokument zu speichern
    wd.Dialog dlgSaveAs = wdApp.Dialogs[wd.WdWordDialog.wdDialogFileSaveAs];
    object[] parameters = new object[1];
    parameters[0] = (object) 0;
    //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.");
}

```

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

```
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 fortzufahren
            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;
}
```

CD-ROM

Die Beispieldateien *Bsp05_01_Find.dot* (benutzerspezifische Add-In-Vorlage, enthält die AutoText-Einträge), *Bsp05_02_Find.txt* sowie *Bsp05_02_Find.dotm* (globale Add-In-Vorlage, enthält den Add-In-Code) finden Sie auf der CD-ROM im Ordner *\Beispiele\Kap05*.

Schleifen und Tabellen

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 5.20 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.

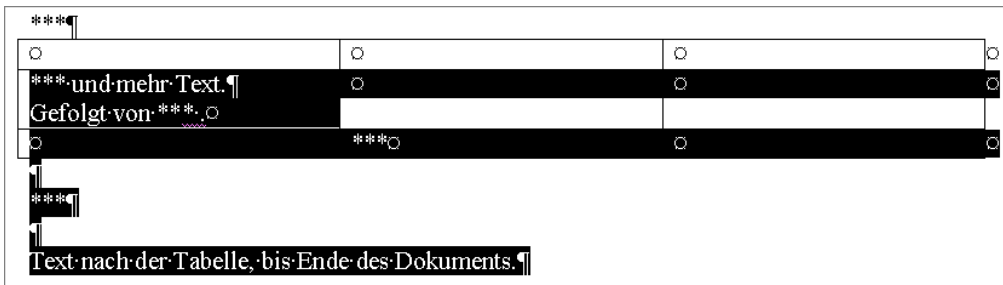
Listing 5.37 Bereich in einer Tabelle bis zum Dokumentende erweitern

```
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. 5.20 Das Resultat des 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 Endlosschleife 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
    Dim lngZaehler As Long
    Dim tbl As Word.Table

    strSuchBegriff = "****"
    Set rngSuchBereich = ActiveDocument.Content
```

Listing 5.38 Ein Dokument mit Tabelle durchsuchen (Fortsetzung)

```

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 rngErgebnisBereich.Cells(1).Range <> _
                tbl.Range.Cells(tbl.Range.Cells.Count).Range 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.
                'die Tabelle muss explizit verlassen werden, sonst Endlosschleife
                rngErgebnisBereich.Collapse wdCollapseEnd
                rngErgebnisBereich.MoveStartWhile Asc(13), 1
                rngErgebnisBereich.MoveStart Unit:=wdCharacter, Count:=1
                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

```

Listing 5.38 Ein Dokument mit Tabelle durchsuchen (Fortsetzung)

```
Private Sub GefundeneStelleBearbeiten(strNeuerText As String, ByRef rng As Word.Range)
    rng.InsertAfter strNeuerText
    rng.Collapse Direction:=wdCollapseEnd
End Sub
```

CD-ROM Die Beispieldatei *Bsp05_03_Find.docm* finden Sie auf der CD-ROM im Ordner *\Beispiele\Kap05*.

Das ganze Dokument mit VBA durchsuchen

Es ist noch die Frage 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 und wurden im Abschnitt »Text aus einem Bereich lesen« eingehender vorgestellt.

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\\"
```

Listing 5.39 Alle Dokumentkomponenten durchsuchen (Fortsetzung)

```

strErsatzBegriff = Replace((ThisDocument.Path & Application.PathSeparator), _
    "\", "\\")

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 (NET): Auch für *Find.Execute* lohnt es sich, »Wrapper«-Funktionen bereitzustellen


```

//Alle Vorkommen eines alten Pfadnamens durch den aktuellen ersetzen
//Hauptsächlich nützlich für Hyperlink, IncludeText, IncludePicture
//und Link-Feldfunktionen.
private void GanzesDokDurchsuchen_CS()
{
    wd.Document doc = wdApp.ActiveDocument;
    //Für Pfadnamen außerhalb einer Feldfunktion.
    //string suchBegriff = "\\AlterServer\Dokumente\Mein Projekt\"
    //string ersatzBegriff = "\\NeuerServer\Projekte\Projekt1\"
    //Für Pfadnamen innerhalb Feldfunktionen (doppelte Backslashes!).
    string suchBegriff = "C:\\\\Test\\";
    string ersatzBegriff = @di.Parent.Parent.Parent.FullName + @"\";
    ersatzBegriff = ersatzBegriff.Replace(@"\", @"\");

    foreach (wd.Range sty in doc.StoryRanges)
    {
        AlleInstanzenErsetzen_CS(sty, suchBegriff, ersatzBegriff);
        sty.Fields.Update
    }
}

```

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

```

        while (sty.NextStoryRange != null)
        {
            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 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 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 *AllenKopfUndFußzeilenFuerInhaltTesten* in der Beispieldatei zu diesem Abschnitt dafür, dass Word alle Kopf- und Fußzeilen korrekt erkennt. Führen Sie diese Prozedur vor dem Suchvorgang aus.

CD-ROM

Die Beispieldatei *Bsp05_04_Find.docm* sowie die verknüpften Dateien *Bsp05_04Find_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 und 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
            End If
        Loop
    Next sty
    MsgBox "Der Suchbegriff wurde " & CStr(lAnzahlVorkommen) & " Male ersetzt.", _
        vbInformation + vbOKOnly
End Sub

Function AlleInstanzenErsetzenMitMustervergleich(rng As Word.Range, _
    ByVal strSuchbegriff As String, ByVal strErsatzbegriff As String) As Boolean

    rng.Find.ClearFormatting
    rng.Find.Replacement.ClearFormatting
    With rng.Find
        .Text = strSuchbegriff
        .Replacement.Text = strErsatzbegriff
    End With
    If rng.Find.Execute Then
        AlleInstanzenErsetzenMitMustervergleich rng, strSuchbegriff, strErsatzbegriff
    End If
End Function
```


Listing 5.41 Einen Text auch in AutoFormen der Kopf- und Fußzeilen finden und ersetzen (Fortsetzung)

```

        .Forward = True
        .Wrap = wdFindContinue
        .Format = False
        .MatchCase = False
        .MatchWholeWord = False
        .MatchWildcards = True
        .MatchSoundsLike = False
        .MatchAllWordForms = False
    End With
    rng.Find.Execute Replace:=wdReplaceAll
    AlleInstanzenErsetzenMitMustervergleich = rng.Find.Found
End Function

```

Listing 5.42 (.NET): Die C#-Version



```

private void EntwurfDurchFinalErsetzen_CS()
{
    wd.Document doc = wdApp.ActiveDocument;
    string suchText = "Entwurfsversion vom [0-9]{1;2}.[0-9]{1;2}.[0-9]{2;4}";
    string ersatzText = "Finalversion vom " + DateTime.Now.ToLongDateString();
    int anzahlVorkommen = 0;

    foreach (wd.Range sty in doc.StoryRanges)
    {
        //Handelt es sich um die normale Kopfzeile des ersten Abschnitts,
        //werden alle grafischen Objekte mit Text bearbeitet
        if (sty.StoryType == wd.WdStoryType.wdPrimaryHeaderStory)
        {
            wd.WdHeaderFooterIndex hp = wd.WdHeaderFooterIndex.wdHeaderFooterPrimary;
            wd.HeaderFooter header = doc.Sections[1].Headers[hp];
            foreach (wd.Shape shp in header.Shapes)
            {
                if (shp.TextFrame.HasText!=0)
                {
                    wd.Range rngShapeTextRange = shp.TextFrame.TextRange;
                    if (AlleInstanzenErsetzenMitMustervergleich_CS(
                        rngShapeTextRange, suchText, ersatzText) )
                    {
                        anzahlVorkommen += 1;
                    }
                }
            }
        }
        //Den StoryRange bearbeiten
        if (AlleInstanzenErsetzenMitMustervergleich_CS(sty, suchText, ersatzText))
        {
            anzahlVorkommen += 1;
            //Mit allen Unterbereichen
            while (! (sty.NextStoryRange==null))
            {
                wd.Range styFolgender = sty.NextStoryRange;
                if (AlleInstanzenErsetzenMitMustervergleich_CS(sty, suchText, ersatzText))
                {
                    anzahlVorkommen += 1;
                }
            }
        }
    }
}

```

Listing 5.42 (.NET): Die C#-Version

```

    }
    MessageBox.Show(String.Format("Der Suchbegriff wurde {0} Male ersetzt.",
        anzahlVorkommen.ToString()));
}

private bool AlleInstanzenErsetzenMitMustervergleich_CS(wd.Range rng,
    string suchBegriff, string ersatzBegriff)
{
    rng.Find.ClearFormatting();
    rng.Find.Replacement.ClearFormatting();
    object objMissing = System.Reflection.Missing.Value;
    object objTrue = (object) true;
    object objFalse = (object) false;
    object objFindText = (object) suchBegriff;
    object objReplaceText = (object) ersatzBegriff;
    object objWrapContinue = wd.WdFindWrap.wdFindContinue;
    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;
}

```

CD-ROM Die Beispieldatei *Bsp05_05_Find.docm* 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 CD.

CD-ROM 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 mit Word 97 eingeführt wurde.

Wenn Sie auf das Bällchen zwischen den Doppelpfeilen klicken, blendet Word eine Palette der gültigen Objekttypen (Abschnitt, Grafik, Kommentar, Fußnote, Endnote, Feldfunktion, Tabelle und

Überschrift) zur Auswahl ein. Nach einer Operation in einer der Registerkarten des Dialogfelds *Suchen und Ersetzen* wird der entsprechende Objekttyp automatisch ausgewählt, sodass **Strg**+**Bild**↓ die nächste Instanz anspringt. Im Objektmodell entspricht diese Funktionalität der *GoTo*-Methode des *Selection*-Objekts, aber darum geht es hier nicht.

Oft würde der Anwender diese Funktionalität gerne ausschalten, um weiterhin problemlos mit der Tastatur im Dokument navigieren zu können. In Listing 5.43 zeigen wir, wie das zu erreichen ist. Die drei Prozeduren fangen die Word-eigenen Befehle *Suchen*, *Ersetzen* und *Gehe zu* ab, blenden die Dialogfelder ein und setzen anschließend das Browserobjekt zurück, sodass die Tastenkombinationen der Seitennavigation dienen.



In Word 2010 funktionieren diese Makros nicht wie in früheren Versionen. Dies hängt mit der Einführung des neuen *Navigation*-Aufgabenbereichs sowie die Umstellung auf das Menüband zusammen, die diese alten WordBasic-Menübefehle ersetzen. Sie können durchaus diese Makros der Symbolleiste für den Schnellzugriff zuweisen und sie werden wie gewünscht funktionieren. Aber um den Standardbefehl *Suchen* zu ersetzen, muss der Befehl im Menüband-XML abgefangen werden. Diese Möglichkeit wird in Kapitel 16 vorgestellt.

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
```

CD-ROM

Die Beispieldatei *Bsp05_07_Find.docm* 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 wurden das *System*-Objekt (Seite 173 ff.) und das *Application*-Objekt (Seite 176 ff.) vorgestellt
- Anschließend wurde die übergeordnete Objekte – *Selection*- (Seite 186 ff.), *Document*- (Seite 188 ff.) und *Template*- (Seite 202 ff.) – erläutert
- Danach wurde das Schlüsselobjekt *Range* (Seite 207 ff.) eingehend behandelt
- Am Schluss des Kapitels wurden die Möglichkeiten der *Find/Replace*-Eigenschaft des *Range*- und *Selection*-Objekts erörtert (Seite 220 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
Seite definieren: das <i>PageSetup</i> -Objekt	248
Seite gestalten: das <i>HeaderFooter</i> -Objekt	256
Lange Dokumente	262
Bausteine: das <i>BuildingBlocks</i> -Objekt	266
Formatieren mit Stil: das <i>Style</i> -Objekt	282
Automatische Nummerierung mit Listen	297
Grafiken: die <i>InlineShape</i> - und <i>Shape</i> -Objekte	308
Zusammenfassung	339

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 »Seite gestalten: das *HeaderFooter*-Objekt« ab Seite 256 aufgeführt.

Manuell wird ein Abschnittswechsel durch den Aufruf *Seitenlayout/Seite einrichten/Umbrüche* eingefügt.

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

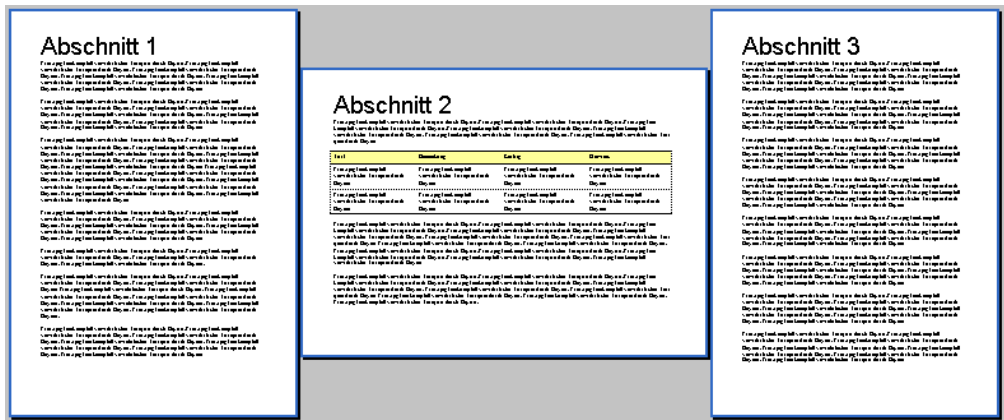
```
ActiveDocument.Range.InsertBreak Type:=wdSectionBreakNextPage
```

Eine komplette Liste aller möglichen Umbrüche ist 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 Buchs beginnt.

Abbildg. 6.1

Unterschiedliches Seitenlayout: die Abschnitte 1 und 3 werden im Hochformat, der Abschnitt 2 wird hingegen im Querformat ausgedruckt



TIPP

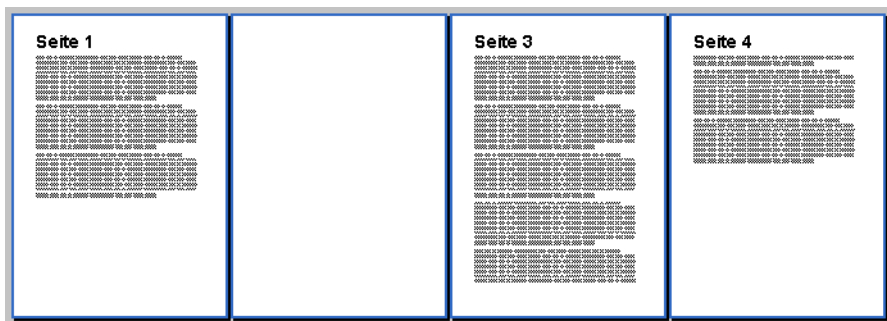
Die Abschnittswchsel innerhalb eines Dokuments sind sehr fragile Strukturen. Vor allem in großen Dokumenten mit mehreren Abschnittswchseln kann es vorkommen, dass der gespeicherte Inhalt eines Dokuments durcheinander gerät und so das Dokument beschädigt wird. In vielen Fällen können bei einem beschädigten 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 Abschnittswchsel (Linie mit feinen Doppelpunkten) kopiert wird. Beim letzten Abschnitt muss der gesamte Inhalt *ohne* die letzte Absatzmarke kopiert werden. Anschließend kann das Dokument neu aufgebaut werden.

Das Problem des Datenverlusts durch beschädigte Dokumente wurde ab 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«, bleiben 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 Abschnittswchsel vom Typ *wdSectionBreakOddPage* eingefügt wird



Headers-Footers	<p>Grundsätzlich verfügt jeder Abschnitt über eigene Kopf- und Fußzeilen. Dies gilt auch für einen Abschnittswechsel vom Typ <code>wdSectionBreakContinuous</code> (siehe Abbildung 6.4). Hier werden die zugehörigen Kopf- bzw. Fußzeilen erst sichtbar, wenn der betreffende Abschnitt größer als eine Seite ist.</p> <p>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 <code>HeaderFooter</code>-Objekt vorgenommen werden (siehe Listing 6.11).</p> <p>Die <code>Headers</code>- und <code>Footers</code>-Eigenschaften geben je eine <code>HeadersFooters</code>-Auflistung zurück. Die Eigenschaften und Methoden dieser Auflistung werden im Abschnitt »Seite gestalten: das <i>HeaderFooter</i>-Objekt« ab Seite 256 beschrieben.</p>
Page-Setup	<p>Die <code>PageSetup</code>-Eigenschaft repräsentiert das Seitenlayout des Abschnitts. Die Eigenschaft gibt ein <code>PageSetup</code>-Objekt zurück. Die Eigenschaften und Methoden dieser Auflistung sind im Abschnitt »Seite definieren: das <i>PageSetup</i>-Objekt« ab Seite 248 erläutert.</p>
ProtectedForms	<p>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 <code>ProtectedForForms</code>-Eigenschaft wird festgelegt, in welchen Abschnitten die Formularfelder gesperrt bzw. zur Erfassung freigeschaltet sind:</p>

```
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, 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.docm*) 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, besteht das StoryRanges-Objekt vom Typ `wdPrimaryHeaderStory` aus zwei Elementen. Also müssten beide Elemente bei der Zählung berücksichtigt werden. Dies ist im Listing 6.1 jedoch nicht der Fall.

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, finden Sie in Kapitel 5. Gerade beim Suchen und Ersetzen müssen 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 ^a
<code>wdPrimaryHeaderStory</code>	7	Standardkopfzeile

Tabelle 6.1 Die einzelnen Elemente der *StoryRanges*-Auflistung (Fortsetzung)

WdStoryType-Enum	Wert	Bedeutung
wdEvenPagesFooterStory	8	Fußzeilen auf den Seiten mit gerader Seitenzahl, sofern <i>Gerade/Ungerade anders</i> aktiviert wurde ^a
wdPrimaryFooterStory	9	Standardfußzeile
wdFirstPageHeaderStory	10	Kopfzeile auf der ersten Seiten des Dokuments, sofern <i>Erste Seite anders</i> aktiviert wurde ^a
wdFirstPageFooterStory	11	Fußzeile auf der ersten Seiten des Dokuments, sofern <i>Erste Seite anders</i> aktiviert wurde ^a
wdFootnoteSeparatorStory	12	Fußnotentrennlinie
wdFootnoteContinuationSeparatorStory	13	Fußnoten-Fortsetzungstrennlinie
wdFootnoteContinuationNoticeStory	14	Fußnoten-Fortsetzungshinweis
wdEndnoteSeparatorStory	15	Endnotentrennlinie
wdEndnoteContinuationSeparatorStory	16	Endnoten-Fortsetzungstrennlinie
wdEndnoteContinuationNoticeStory	17	Endnoten-Fortsetzungshinweis

^a Die Optionen *Gerade/Ungerade anders* bzw. *Erste Seite anders* stehen im Zusammenhang mit den möglichen Einstellungen für die Kopf- und Fußzeilen. Diese Eigenschaften können über den Dialoglauncher der Gruppe *Seite einrichten* in der Registerkarte *Seitenlayout* geändert werden.

Bei der *StoryRanges*-Auflistung handelt es sich um eine Auflistung von *Range*-Objekten. Aus diesem Grunde verfügt das Objekt, mit Ausnahme der *NextStoryRange*-Eigenschaft, über keine besonderen Eigenschaften, auf die hier näher eingegangen werden müsste. Die Eigenschaften und Methoden des *Range*-Objekts wurden bereits in Kapitel 5 detailliert aufgeführt.

Die *StoryRanges*-Auflistung kann nicht erweitert werden. Die *Add*-Methode wird nicht unterstützt, denn die möglichen Elemente der Auflistung sind, wie in Tabelle 6.1 dargestellt, klar definiert.

Next-
Story-
Range

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 Nothing zurück, da diese Dokumentkomponente <i>nie</i> über zusätzliche Elemente verfügen kann
wdTextFrameStory	Das nächste Element von verknüpften Textfeldern oder Nothing
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 Nothing

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 (.NET): Die C#-Version: *Sections*- sowie *Headers*-Auflistungen werden wie Datenfelder behandelt



```

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

```

Listing 6.3 (.NET): Die C#-Version: *Sections*- sowie *Headers*-Auflistungen werden wie Datenfelder behandelt (*Fortsetzung*)

```
//Kopfzeile Abschnitt Eins bis Drei setzen

wd.HeaderFooter hdr1 = doc.Sections[1].Headers[
    wd.WdHeaderFooterIndex.wdHeaderFooterPrimary];
hdr1.Range.Text = "Kopfzeile A";

//Kopfzeile Abschnitt Drei setzen
wd.HeaderFooter hdr3 = doc.Sections[3].Headers[
    wd.WdHeaderFooterIndex.wdHeaderFooterPrimary];
hdr3.LinkToPrevious = false;
hdr3.Range.Text = "Kopfzeile B";

//Alle Kopfzeilen ausgeben
wd.Range rng = hdr1.Range;
MessageBox.Show(rng.Text);
while (rng.NextStoryRange != null)
{
    rng = rng.NextStoryRange;
    MessageBox.Show(rng.Text);
}
}
```

Das Beispiel legt ein neues Dokument an und erzeugt zwei zusätzliche Abschnitte. Der primären Kopfzeile im ersten Abschnitt wird ein Text zugewiesen. Die anderen beiden Abschnitte übernehmen die Kopfzeile, da diese standardmäßig mit der vorherigen verknüpft sind. Die Verknüpfung zur vorherigen Kopfzeile wird im dritten Abschnitt aufgelöst und gleichzeitig wird eine eigene Kopfzeile definiert. Anschließend wird mit einer Schleife das *StoryRanges*-Objekt (*wdPrimaryHeaderStory*) durchforstet. Die Schleife wird so lange durchlaufen, bis kein nachfolgendes Element in dieser Dokumentkomponente mehr gefunden wird. Beachten Sie dabei, dass zweimal eine Meldung (der Inhalt der Kopfzeile) am Bildschirm ausgegeben wird. Dies deshalb, weil das Dokument zwar über drei Abschnitte, aber nur über zwei unterschiedliche Kopfzeilen verfügt.

CD-ROM

Die in diesem Abschnitt dargestellten Beispiele finden Sie in der Beispieldatei *Bsp06_01a.docm* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap06*.

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 Standardeinstellungen definiert werden. Dazu steht das Dialogfeld *Seite einrichten* zur Verfügung. Dieses kann durch Aufruf des Dialoglaunchers 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
Page-
Height
Page-
Width

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-Methode zur Verfügung.

Tabelle 6.3 Zusammenstellung der wichtigsten Konstanten für das Papierformat

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

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

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

**Margin-
Bottom** 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 Eigenschaft 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)
        .LeftMargin = CentimetersToPoints(sngEinzugH)
        .RightMargin = CentimetersToPoints(sngEinzugH)
        .Gutter = CentimetersToPoints(0)
    End With

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

```

Listing 6.5 (.NET): Die C#-Version von Listing 6.4



```
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();
}
```

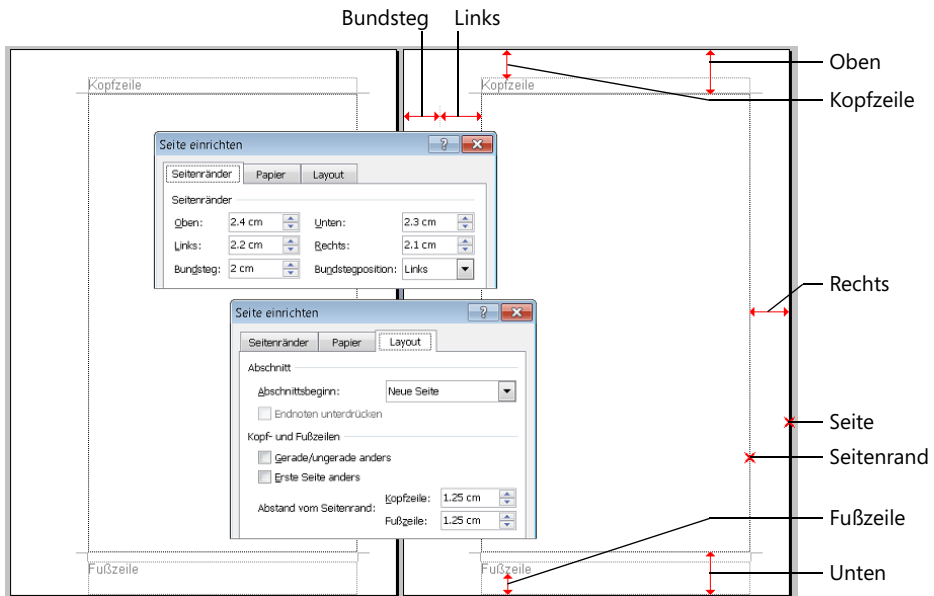
Gutter Die Gutter-Eigenschaft dient zum Festlegen des Bundstegs. Der Bundsteg entspricht dem zusätzlichen Abstand, der zum Binden des Dokuments zum Seitenrand hinzugefügt wird (siehe Abbildung 6.3).

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

Footer-Distance **HINWEIS** Enthält die Kopfzeile mehr Text, als zwischen der Oberkante der Kopfzeile und dem Seitenrand eingefügt werden kann, wird der beschreibbare Bereich des Abschnitts automatisch verkleinert. Der Wert der MarginTop-Eigenschaft bleibt bestehen. Das gleiche Verhalten gilt analog für die Fußzeile.

Alle Maßeinheiten zum Festlegen des Satzspiegels werden in Punkten festgelegt.

Abbildg. 6.3 Die Einstellungen des Satzspiegels und deren Auswirkungen



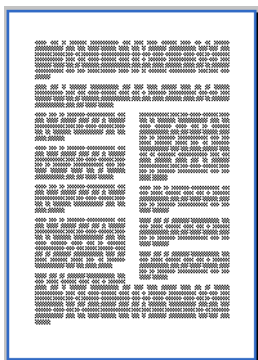
HINWEIS In Abbildung 6.3 sind zusätzlich die Positionen der »Seite« und des »Seitenrands« markiert. Diese beiden Ränder können beim Positionieren von Shape-Objekten auch als Ursprungskoordinaten verwendet werden (siehe Abbildung 6.34 auf Seite 323).

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 (wdSectionBreakContinuous) eingefügt werden.

Abbildg. 6.4 Eine unterschiedliche Anzahl von Spalten kann auf der gleichen Seite zum Einsatz kommen



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.

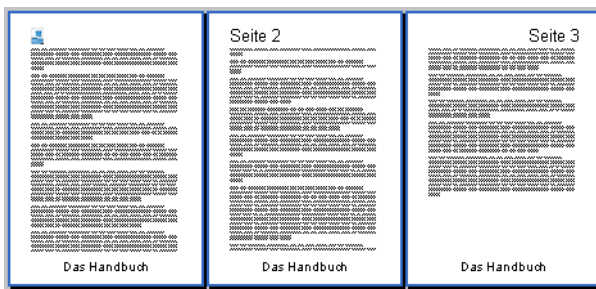
Different-
First-
Page-
Header-
Footer
OddAnd-
Even-
Pages-
Header-
Footer

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. Information zum Ansteuern der Kopf- und Fußzeilen werden im Abschnitt »Seite gestalten: das `HeaderFooter`-Objekt« ab Seite 256 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
    End With
End Sub
```

Listing 6.6 Synchronisieren der restlichen Fußzeilen anhand der Fußzeile auf der ersten Seite (*Fortsetzung*)

```
.Range.Paragraphs(i).Range.Delete
End With
End Sub
```

Durch die verwendete Art des direkten Zuweisens des formatierten Textes von einer Fußzeile an eine andere wird diese zwar ersetzt, doch bleibt die letzte Absatzmarke der ehemaligen Fußzeile bestehen. Aus diesem Grunde wird der letzte Absatz nachträglich aus der Fußzeile entfernt.

Die letzten interessanten Eigenschaften des PageSetup-Objekts haben eigentlich nur einen indirekten Einfluss auf das Seitenlayout des Abschnitts. Hier geht es um die Steuerung des Papierschachts für den Ausdruck.

Jedem Abschnitt kann für den Ausdruck eine andere Papierquelle (Papierschacht) zugewiesen werden. Zusätzlich kann innerhalb eines jeden Abschnitts für die erste Seite das zu bedruckende Papier aus einem anderen Papierschacht eingezogen werden, als für die restlichen Seiten.

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

    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 Dialoglauncher über *Seitenlayout/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 Papierschachtsteuerung

WdPaperTray-Konstante	Wert	Bedeutung
wdPrinterDefaultBin	0	Standardpapierschacht gemäß den Einstellungen des Druckertreibers bzw. der Papierschacht, der in den Optionen festgelegt wurde
wdPrinterOnlyBin	1	Einziger Papierschacht (entspricht Oberer Papierschacht)
wdPrinterUpperBin	1	Oberer Papierschacht
wdPrinterLowerBin	2	Unterer Papierschacht
wdPrinterMiddleBin	3	Mittlerer Papierschacht
wdPrinterManualFeed	4	Manuelle Papierzufuhr, in den meisten Fällen wartet der Drucker, bis das Papier eingelegt und vom Anwender bestätigt wird
wdPrinterEnvelopeFeed	5	Briefumschlag
wdPrinterManualEnvelopeFeed	6	Manuelle Papierzufuhr für Briefumschlag
wdPrinterAutomaticSheetFeed	7	Einzelblatteinzug für Matrixdrucker
wdPrinterTractorFeed	8	Endlos garnitur für Matrixdrucker
wdPrinterSmallFormatBin	9	Papierschacht mit kleinem Papierformat
wdPrinterLargeFormatBin	10	Papierschacht mit großem Papierformat
wdPrinterLargeCapacityBin	11	Zusätzlicher optionaler Papierschacht
wdPrinterPaperCassette	14	Zusätzlicher Papierschacht
wdPrinterFormSource	15	Automatisch auswählen

HINWEIS Die aktuellen Werte für die Papierzufuhr werden im Dokument gespeichert. Dies hat den Vorteil, dass bei einem wiederholten Ausdruck des gleichen Dokuments die gewünschte Papierzufuhr nicht erneut ausgewählt werden muss.

Dieser Vorteil kann jedoch auch ein Nachteil sein. Werden Dokumente intern oder auch extern ausgetauscht und auf einem anderen Drucker ausgegeben, können die gespeicherten Einstellungen dazu führen, dass der Ausdruck auf einen falschen Papierschacht zugreift.

Wird der FirstPageTray- bzw. OtherPagesTray-Eigenschaft ein Wert zugewiesen, der vom aktuellen Druckertreiber nicht unterstützt wird, wird automatisch auf den Papierschacht »Automatisch auswählen« (wdPrinterFormSource) gewechselt. Dies bedeutet, dass die Standardeinstellungen des Druckertreibers berücksichtigt werden.

CD-ROM Die in diesem Abschnitt dargestellten Beispiele finden Sie in der Beispieldatei *Bsp06_02a.docm* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap06*.

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 im Menüband der Befehl *Einfügen/Kopf- und Fußzeile/Kopfzeile/Kopfzeile bearbeiten* aufgerufen werden. 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 »Seite definieren: das `PageSetup`-Objekt« ab Seite 248 erläutert.

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

WdHeaderFooter-Enum	Wert	Bedeutung
<code>wdHeaderFooterEvenPages</code>	3	Bei aktivierter <code>OddAndEvenPagesHeaderFooter</code> -Eigenschaft wird die Kopf- bzw. Fußzeile auf allen Seiten mit gerader Seitennummer ausgegeben
<code>wdHeaderFooterFirstPage</code>	2	Bei aktivierter <code>DifferentFirstPageHeaderFooter</code> -Eigenschaft wird die Kopf- bzw. Fußzeile auf der ersten Seite des Abschnitts ausgegeben
<code>wdHeaderFooterPrimary</code>	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 Markorekorder aufgezeichneten Codes sollte unbedingt verzichtet werden. Diese Programmzeilen bauen auf dem `Selection`-Objekt auf. Dies hat zur Folge, dass während der Laufzeit des Makros ein unschönes Bildschirmflackern sichtbar ist.

Zusätzlich muss berücksichtigt werden, dass nicht sichergestellt werden kann, dass die gewünschte Kopf- bzw. Fußzeile auch tatsächlich bearbeitet wird. Dies kann anhand eines einfachen Beispiels verdeutlicht werden.

Ist die `DifferentFirstPageHeaderFooter`-Eigenschaft auf `True` gesetzt und die Kopfzeile wird durch Aufruf des Menübefehls *Einfügen/Kopfzeile/Kopfzeile bearbeiten* 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 im Dialogfeld *Seite einrichten* 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, 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 gleichbleibenden 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"
```

Listing 6.10 Unterschiedliche Kopfzeilen bei gleichbleibenden Fußzeilen erstellen (*Fortsetzung*)

```

    .Footers(wdHeaderFooterPrimary).Range.Text = "Fußzeile Eins"
End With

doc.Range.InsertBreak wdSectionBreakNextPage

With doc.Sections(2).Headers(wdHeaderFooterPrimary)
    .LinkToPrevious = False
    .Range.Text = "Kopfzeile Zwei"
End With
End Sub

```

Exists Mit der Exists-Eigenschaft kann festgestellt werden, ob ein spezifisches Objekt der HeaderFooters-Auflistung bereits vorhanden ist. Die beiden nachstehenden Programmzeilen führen die gleiche Prüfung aus.

```

If ActiveDocument.Sections(1).Headers(wdHeaderFooterFirstPage).Exists Then
If ActiveDocument.Sections(1).PageSetup.DifferentFirstPageHeaderFooter Then

```

Bei dieser Prüfung wird nur getestet, ob das Objekt vorhanden ist. Dies entspricht einer Prüfung des Status des entsprechenden Kontrollkästchens im Dialogfeld *Seite einrichten*. Eine Kontrolle, ob bereits ein Text in die Kopfzeile eingetragen wurde, findet auf diese Weise nicht statt.

Page-Numbers Die PageNumbers-Eigenschaft gibt eine PageNumbers-Auflistung zurück. Diese Auflistung entspricht allen Seitenzahlfeldern, die in der entsprechenden Kopf- bzw. Fußzeile vorhanden sind.

Aus der Sicht der Autoren kann auf das Einfügen eines PageNumber-Objekts ins Dokument verzichtet werden. Denn in den Programmversionen vor Word 2007 wurde das Einfügen einer Seitennummer mit dem Menübefehl *Einfügen/Seitenzahlen* erzeugt. Die Seitennummer wurde innerhalb eines Positionsrahmens oder eines Textfelds erstellt. Diesen Positionsrahmen und Textfelder mittels VBA zu bearbeiten, ist weit aufwendiger als das Einfügen und Bearbeiten eines *Page*-Felds innerhalb des Textflusses. Dieses Vorgehen ist seit Word 2007 Standard. Die 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 Word 2010 wird das in Abbildung 6.6 abgebildete Dialogfeld über *Einfügen/Kopf- und Fußzeile/Seitenzahl/Seitenzahlen formatieren* aufgerufen. Bei den Programmversionen vor Word 2007 ist es wichtig, dass das Dialogfeld *Seitenzahlenformat* nicht durch Betätigen der Schaltfläche OK verlassen wird. Sonst wird eine Seitenzahl in einen Positionsrahmen zusätzlich eingefügt.

Muss die Seitennummerierung in einem Abschnitt bei einer bestimmten Zahl beginnen, 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
    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
```

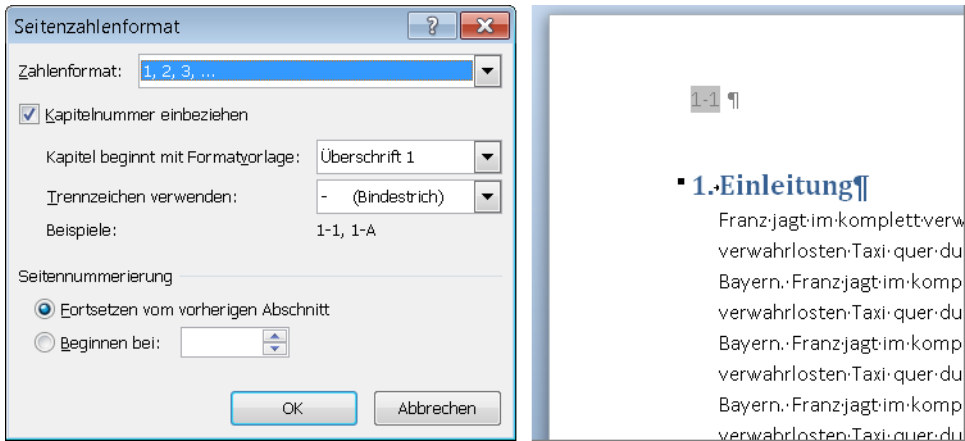
Listing 6.12 (.NET): Die C#-Version von Listing 6.11



```
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-Objekt 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.

CD-ROM Die dargestellten Beispiele finden Sie in der Beispieldatei *Bsp06_02a.docm* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap06*.

Lange Dokumente

Word dient vielen Bedürfnissen, die von kurzen Memos über Briefe und Berichte bis zu Büchern und Handbücher reichen. In den vorangehenden Abschnitten wurden Objekte vorgestellt, die in allen Dokumentarten benutzt werden können. In diesem Abschnitt werden wir einige Aspekte und Objekte diskutieren, die primär mit der Dokumentverwaltung und der Erstellung von langen Dokumenten zu tun haben.

Mit der Einführung eines XML-Dateiformats in Word 2003 und 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 eintausend Seiten umfassen. Die Frage ist jetzt eher, ob es für Autoren oder Anwender wünschenswert ist, den gesamten Text eines Werks in einem einzigen Dokument zu verwalten. Einige Beispiele für eine gegenteiligen Entscheidung:

- Word unterstützt die gleichzeitige Bearbeitung eines einzelnen Dokuments nicht. Arbeiten mehrere Personen am gleichen Werk, ist es sinnvoll, dies in mehrere Dokumente aufzuteilen.
- In manchen Werken bleiben einige Teile statisch, während andere häufiger bearbeitet oder ausgetauscht werden. Es ist vorteilhaft, wenn der dynamische Text getrennt verwaltet wird.
- Viele Dokumentationen unserer globalen Welt müssen mehrsprachig vorliegen. Es kann wünschenswert sein, die verschiedenen Sprachversionen getrennt zu speichern, und diese gleichzeitig nebeneinander im gleichen Dokument zu sehen oder auszudrucken.

Zentraldokumente

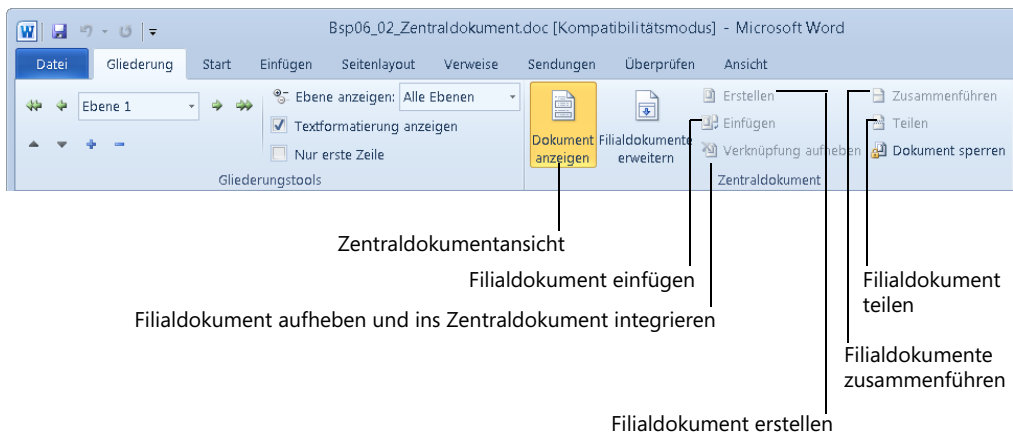
Für die Bewältigung solcher Aufgaben stellt Word zwei Funktionalitäten zur Verfügung: Zentral- und Filialdokumente sowie die Dokumentenverknüpfung (*IncludeText*-Feldfunktion). Die Zentraldokument-Funktionalität ermöglicht das Einfügen von »Filialdokumenten« mit automatischer Erhaltung der Seitenlayout-Eigenschaften jedes einzelnen Dokuments. Dies wird erreicht durch das Einfügen von zwei Abschnittswechsels zwischen jedem Dokument, zusammen mit dem Ausschalten der Verknüpfungen zwischen Kopf- und Fußzeilen.

Allgemein raten Word-Kenner von Zentraldokumenten ab. Unsachgemäß eingesetzt führen sie zu Dokumentbeschädigungen und kostspieligem Daten- und Zeitverlust. Vorausgesetzt einer korrekten Handhabung ist es jedoch sinnvoll, sich ihrer Vorteile zu bedienen. Wenn Sie sich für eine Zentraldokument-Lösung entscheiden, achten Sie auf die folgenden Punkte:

- Die Zentral- und Filialdokumente sollten alle von der gleichen Dokumentvorlage erstellt werden, die ihrerseits von einer neuen Kopie der *Normal.dotm* 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 Abschnittswchsel) 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 dokumentübergreifende Elemente wie Listennummerierungen, Verzeichnisse, Verweise und Indexeinträge zur Verfügung zu stellen.
- Querverweise sowie Inhaltsverzeichnisse und Indexe 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 Gruppe Zentraldokument durch Anklicken der Schaltfläche *Dokument anzeigen* (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. Ein Filialdokument kann in zwei Dokumente geteilt werden; mehrere Filialdokumente dürfen zu einem vereint werden.

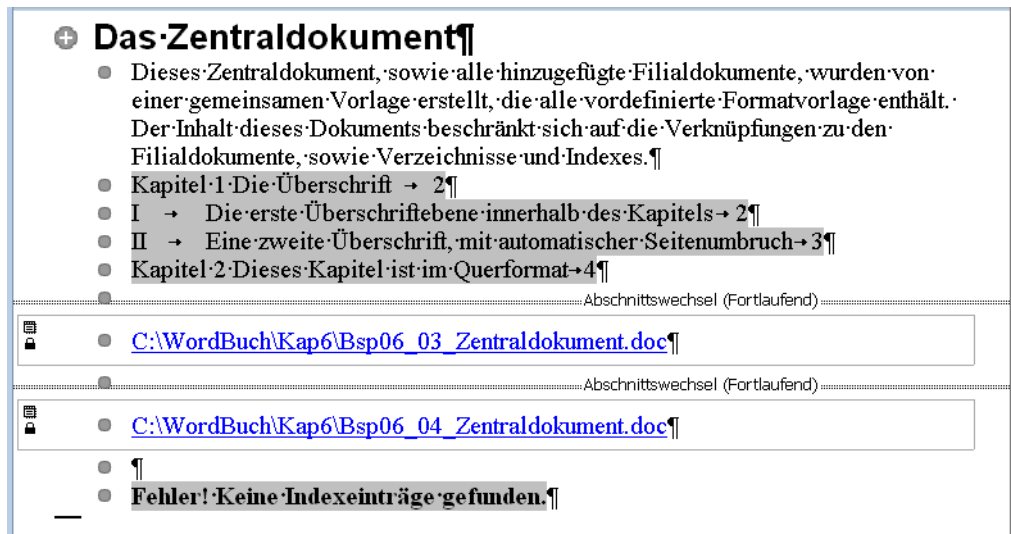
Abbildg. 6.7 Die Symbolleiste für die Arbeit mit Zentral- und Filialdokumenten



Wird ein Filialdokument aufgrund 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 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



CD-ROM

Die Beispieldatei *Bsp06_01_Zentraldokument.dotx* ist eine Vorlage für das Beispiel in Abbildung 6.8. Die Beispieldatei *Bsp06_02_Zentraldokument.docx* ist das eigentliche Zentraldokument; *Bsp06_03_Zentraldokument.docx* sowie *Bsp06_04_Zentraldokument.docx* die Filialdokumente. Sie finden alle Dateien auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap06*.

Beim Erstellen eines Filialdokuments aus einem Bereich des Zentraldokuments muss das neue Filialdokument in einem eigenen 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.

SubDocuments

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 Zentraldokument wird. Bitte beachten Sie, dass

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

Um der ersten Forderungen 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 Zentraldokument basiert, blendet Word eine entsprechende Meldung ein, die nicht unterdrückt werden kann.

Beim Speichern des Zentraldokuments prüft Word, 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 *docx*-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\Kap06\"
    strDateiname = Dir$(strPfad & "Test*.docx")
    Set docZentral = ActiveDocument
    docZentral.ActiveWindow.View = wdMasterView
    Selection.HomeKey Unit:=wdStory
    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
```

CD-ROM

Die Beispieldatei *Bsp06_05_Zentraldokument.docm* mit der Prozedur finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap06*.

Dokumente verknüpfen

Die alternative Methode ist, Dokumente über *Text aus Datei* unter *Einfügen/Objekt/Test* 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 vonstattengeht, 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 diese der Methode des `UpdateSource` des `Field`-Objekts.

Bausteine: das *BuildingBlocks*-Objekt



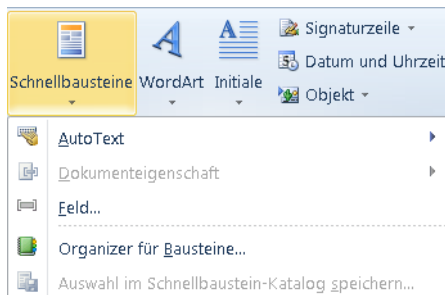
Mit Word 2007 wurden neben der neuen Oberfläche eine weitere Neuerung eingeführt: die *Schnellbausteine* zum Hinzufügen vorformatierter Inhalte zu Dokumenten. Sie rufen diesen Menübefehl über die Schaltfläche *Schnellbausteine* auf der Registerkarte *Einfügen* in der Gruppe *Text* auf (Abbildung 6.9).

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

- *Dokumenteigenschaft* (Abschnitt »Die Dokumenteigenschaften« ab Seite 270)
- *Feld* (Abschnitt »Felder (Feldfunktionen)« ab Seite 272)
- *Organizer für Bausteine* (Abschnitt »Der Organizer für Bausteine« ab Seite 280)
- *Auswahl im Schnellbaustein-Katalog speichern* (Abschnitt »Der Schnellbaustein-Katalog« ab Seite 272)

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 Menübefehlen *Organizer für Bausteine* und *Auswahl im Schnellbaustein-Katalog speichern* verbergen.

In diesem Abschnitt soll nun ein Überblick über die Funktionen dieses Menüpunkts 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.

Word unterscheidet dabei zwischen den benutzerdefinierten und integrierten Bausteinen.



In Word 2007 können benutzerdefinierte Bausteine in beliebigen Vorlagen gespeichert werden, während die integrierten Bausteine in der Vorlage *Building Blocks.dotx* gespeichert sind. Wird die Vorlage *Building Blocks.dotx* gelöscht, wird sie von Word 2007 beim nächsten Start wieder mit den Standard-Schnellbausteinen neu erstellt.



In Word 2010 sind die integrierten Bausteine zur Unterscheidung in einer neuen Vorlage *Built-In Building Blocks.dotx* gespeichert. Die bisherige *Building Blocks.dotx* steht nun dem Benutzer zur Verfügung. Wird die Vorlage *Built-In Building Blocks.dotx* gelöscht, wird sie von Word 2010 beim nächsten Start wieder mit den Standard-Schnellbausteinen neu erstellt. Wird die Vorlage *Building Blocks.dotx* gelöscht, erzeugt Word 2010 beim nächsten Start eine *leere* neue Vorlage.

**HINWEIS**

Die Vorlage *Building Blocks.dotx* liegt unter Windows XP im Verzeichnis

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

Und unter Windows Vista und Windows 7 im Verzeichnis

`C:\Users\<Benutzername>\AppData\Roaming\Microsoft\Document Building Blocks\1031\`



Für Word 2010 liegen die beiden Vorlagen *Building Blocks.dotx* und *Built-In Building Blocks.dotx* unter Windows Vista und Windows 7 im Verzeichnis

`C:\Users\<Benutzername>\AppData\Roaming\Microsoft\Document Building Blocks\1031\14\`

Mit Word 2010 sind keine weiteren Bausteintypen hinzugekommen, sondern nur zusätzliche Beispiele, sodass weiterhin folgende Building Block-Typen (Kataloge) zur Verfügung stehen:

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
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	Benutzerdefiniertes 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

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

ID	BuildingBlockType	wdBuildingBlockTypes-Konstante
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« ab Seite 280.

Die zentrale Bausteinvorlage *Built-In Building Blocks.dotx*

Normalerweise werden in Word 2007 und Word 2010 Bausteine geladen, wenn sie zum ersten Mal benötigt werden, z.B. wenn ein Benutzer mithilfe des Menübands einen Katalog aufruft. Da die integrierten Bausteine jedoch für Word 2007 in der Datei *Building Blocks.dotx* bzw. für Word 2010 in der Datei *Built-In 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, sodass Sie für den direkten Zugriff auf diese Vorlage erst noch den Index der Datei *Building Blocks.dotx* und *Built-In Building Blocks.dotx* ermitteln müssen. Wenn keine Add-Ins geladen sind, ist in Word 2010 die *Built-In Building Blocks.dotx* die erste Vorlage und Sie erhalten über die folgende Definition Zugriff auf diese Vorlage:

```
Templates(1)
```

Die zweite Vorlage ist die nun dem Benutzer zur Verfügung stehende *Building Blocks.dotx*.

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

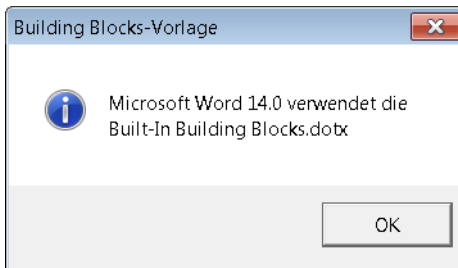
TIPP

Wenn Sie in einem Makro je nach Word-Version automatisch zwischen den beiden Bausteinvorlagen umschalten möchten, können Sie das nachstehende Makro *FindWordBuildingBlocksDOTX* verwenden, welches einen Verweis auf die jeweilige Baustein-Vorlage zurückliefert.

Dazu wird in dieser Funktion anhand der Compilerkonstanten für die VBA-Versionsnummer die jeweils korrekte Bausteinvorlage ermittelt.

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

```
Function FindWordBuildingBlocksDOTX() As Template
Dim objTemplate As Template
#If VBA7 Then ' Word 2010
    For Each objTemplate In Templates
        If objTemplate.Name = "Built-In Building Blocks.dotx" Then
            Set FindWordBuildingBlocksDOTX = objTemplate
            Exit For
        End If
    Next objTemplate
#ElseIf VBA6 Then ' Word 2007
    For Each objTemplate In Templates
        If objTemplate.Name = "Building Blocks.dotx" Then
            Set FindWordBuildingBlocksDOTX = objTemplate
            Exit For
        End If
    Next objTemplate
#End If
If FindWordBuildingBlocksDOTX Is Nothing Then
    Templates.LoadBuildingBlocks ' Vorlagen neu laden
    Set FindWordBuildingBlocksDOTX = FindWordBuildingBlocksDOTX
End If
End Function
```

Abbildg. 6.10 Automatische Ermittlung der Bausteinvorlage

HINWEIS

Sollte einmal der Fall auftreten, dass keine integrierten Bausteine/Kopfzeilen/Fußzeilen oder ähnliche sonst verfügbare Auswahllisten angezeigt werden, könnte die *Built-In 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 Standardauswahllisten und Bausteinen.

Die Dokumenteigenschaften

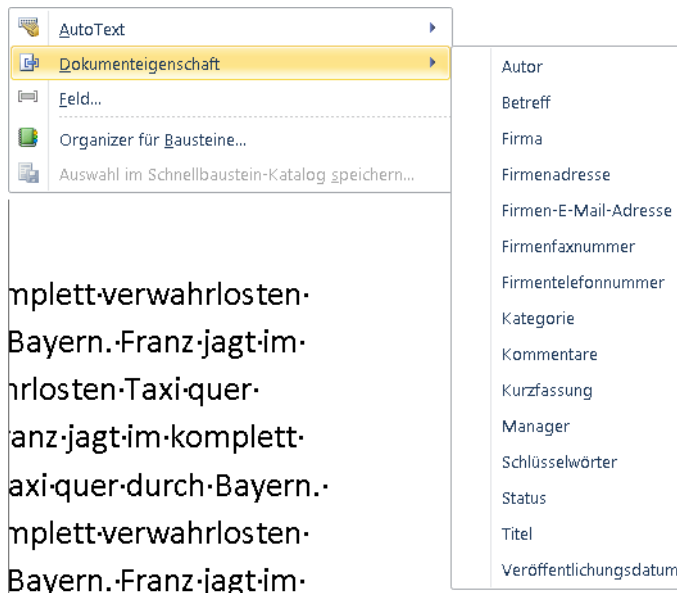
Der erste Menüeintrag in den Schnellbausteinen betrifft die Dokumenteigenschaften.

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

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. 6.11 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.

Diese Dokumenteigenschaften entsprechen denen, die Sie auch auf der neuen Informationsübersicht, aufzurufen über *Datei/Informationen*, im rechten Bereich über die Auswahlliste *Eigenschaften/Erweitere Eigenschaften* aufrufen und ändern können. Wenn Sie aus der Auswahl *Eigenschaften/Dokumentbereich anzeigen* auswählen, werden die wichtigsten Dokumenteigenschaften oberhalb des Dokumentes eingeblendet und können dort auch geändert werden.

Mit der folgenden Anweisung können Sie diesen Dokumentbereich 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 6.11). 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.

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.

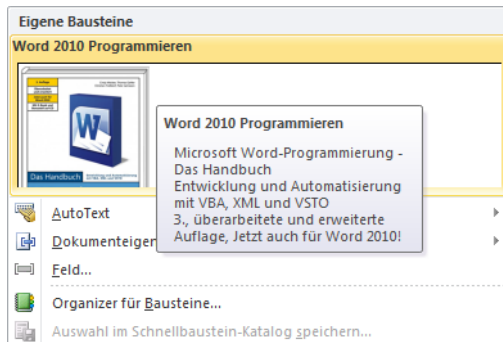
Auch für diesen Schnellbaustein hat Microsoft keine besonderen Befehle zur Verfügung gestellt, sodass Sie ganz normal über die Fields-Eigenschaft Felder erstellen und darauf zugreifen können.

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 Menübefehls).

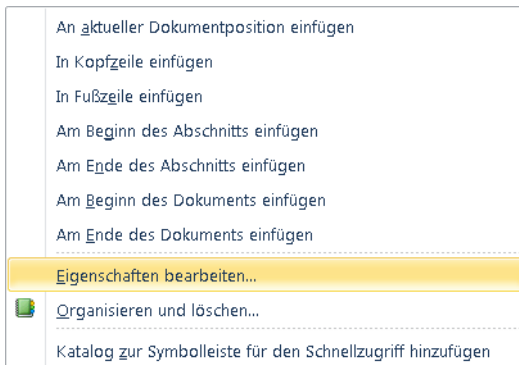
Dazu müssen Sie den gewünschten Bereich (Text und ggf. weitere Elemente) markieren, bevor Sie ihn über den Befehl *Auswahl im Schnellbaustein-Katalog speichern* in das Schnellbaustein-Menü übernehmen können. Der ausgewählte Text (ggf. mit weiteren Elementen) wird dann im Menü mit einer Voransicht angezeigt.

Abbildg. 6.12 Festlegen eines markierten Textes als Schnellbaustein im Schnellbaustein-Menü



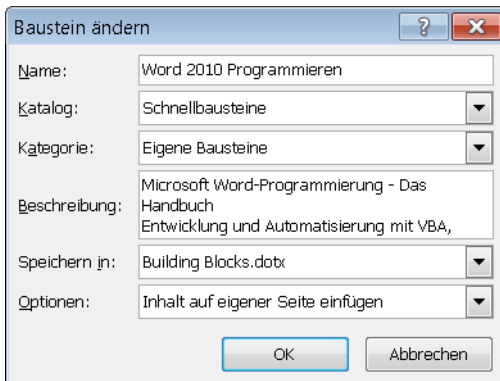
Jeder Baustein wird durch verschiedene Eigenschaften beschrieben. Sie können diese Eigenschaften anzeigen, wenn Sie zunächst mit der rechten Maustaste einen Schnellbausteineintrag anklicken und im daraufhin geöffneten Kontextmenü den Eintrag *Eigenschaften bearbeiten* wählen.

Abbildg. 6.13 Anzeigen und bearbeiten der Bausteineigenschaften



Daraufhin wird das Dialogfeld zum Bearbeiten der Bausteineigenschaften geöffnet.

Abbildg. 6.14 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 Vorlage *Building Blocks.dotx* handeln; der Datei *Built-In Building Blocks.dotx* kann kein Schnellbaustein hinzugefügt werden (siehe den Abschnitt »Die zentrale Bausteinvorlage *Built-In Building Blocks.dotx*« ab Seite 269).

Über die Parameter der `Add`-Methode wird dann der Baustein entsprechend der Abbildung 6.14 festgelegt.

Listing 6.16 Markierten Text per Makro als Schnellbaustein speichern

```
Set objTemplate = FindTemplate("Building Blocks.dotx")
Set objBB = objTemplate.BuildingBlockEntries.Add( _
    Name:="Word 2010 Programmieren", _
    Type:=wdTypeQuickParts, _
    Category:="Eigene Bausteine", _
    Range:=Selection.Range, _
    Description:="Microsoft Word-Programmierung - Das Handbuch," &
        "Entwicklung und Automatisierung mit VBA, XML und VSTO 3., " &
        "überarbeitete und erweiterte Auflage, Jetzt auch für Word 2010!", _
    InsertOptions:=wdInsertContent)
End Sub
```

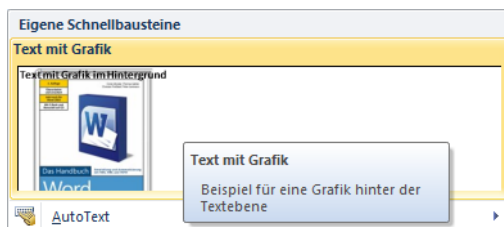
ACHTUNG

Leider lassen sich in Word 2007 nicht mehr alle Funktionsaufrufe und Funktionen, die über die neu eingeführte Multifunktionsleiste bzw. das »Menüband«, wie es in Word 2010 nun heißt, erreichbar sind, mit dem Makrorekorder aufzeichnen. Während der Makrorekorder in Word 2007 z. B. für das Erstellen eines Schnellbausteins noch den folgenden VBA-Code aufzeichnete:

```
WordBasic.CreateCommonFieldBlockFromSel Description:="Autor"
```

zeichnet Word 2010 nun nur noch das Markieren des Textes, aber keinen Befehl zum Erstellen des Schnellbausteins selbst mehr auf.

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. Sie müssen dabei nur beachten, dass der Anker der Grafik mit im markierten Bereich liegt. Der Makroaufruf bleibt in diesem Fall gleich.

Abbildg. 6.15 Festlegen eines markierten Textes mit Grafik als Schnellbaustein im Schnellbaustein-Menü


Sobald Sie einen Schnellbaustein angelegt haben, wird dieser auch im Organizer für Bausteine aufgeführt (siehe den Abschnitt »Der Organizer für Bausteine« ab Seite 280).

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 6.13) aufzurufen. Diese Einfügeoptionen sind soweit selbsterklärend, sodass darauf nicht weiter eingegangen wird.

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

Da die Schnellbausteine an verschiedenen Stellen (Vorlagen) gespeichert werden können, werden im Listing 6.17 alle geladenen Vorlagen durchlaufen und für jede Vorlage der Schnellbaustein typ wdTypeQuickParts auf Einträge (BuildingBlocks) überprüft.

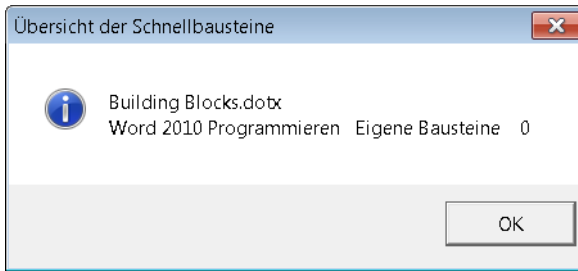
Da die Baustein-Einträge im Baustein-Katalog über die Reihenfolge

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 6.16).

Listing 6.17 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)
                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. 6.16 Anzeige verfügbarer Schnellbausteine


Für die `InsertOptions`-Eigenschaft stehen folgende `wdDocPartInsertOptions`-Konstanten zur Verfügung (siehe auch Abbildung 6.14, Feld *Optionen*).

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

Name	Wert	Beschreibung
<code>wdInsertContent</code>	0	Inlinebaustein Fügt nur den Inhalt des Schnellbausteins in den Fließtext ein
<code>wdInsertPage</code>	2	Seitenebenen-Baustein Fügt den Inhalt des Schnellbausteins auf einer eigenen Seite ein
<code>wdInsertParagraph</code>	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 6.18 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
                For intBBCount = 1 To objBBT.Categories(intCount).BuildingBlocks.Count
                    Set objBB = objBBT.Categories("Eigene Bausteine").BuildingBlocks(intCount)
                    If objBB.Name = "Word 2010 Programmieren" 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`-Objekts 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 Dokumentstellen 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`-Objekts) dar. Insgesamt stehen die 32 in Tabelle 6.7 aufgeführten Baustein-typen zur Verfügung.

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

Wie im Abschnitt »Der Schnellbaustein-Katalog« ab Seite 272 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 6.19 werden für einen Baustein aus der *Built-In Building Blocks.dotx* die verschiedenen Informationen (*Index*, *Vorlage*, *Katalog*, *Kategorie*) ausgegeben (Abbildung 6.17).

Listing 6.19 Ausgabe der Bausteininformationen

```
Sub ShowBuiltInBuildingBlockInformation()
    Const c_Titel As String = "Bausteininformationen"
    Dim objTemplate As Template
    Dim i As Variant
    Dim objBB As BuildingBlock
    Templates.LoadBuildingBlocks
    Set objTemplate = FindBuiltInBuildingBlocksDOTX
    i = InputBox("Bitte Bausteinnummer angeben (1-" & _
        objTemplate.BuildingBlockEntries.Count & ")", _
        "Bausteininformationen", "3")
    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 _
```

Listing 6.19 Ausgabe der Bausteininformationen (Fortsetzung)

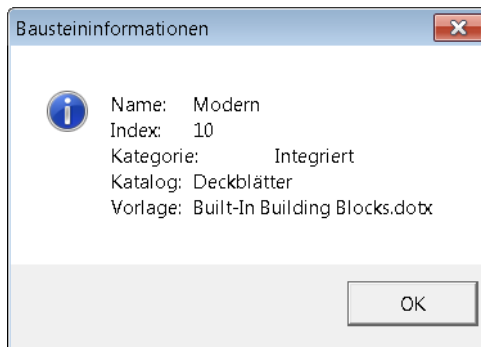
```

    & 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

```

CD-ROM Die Funktion *FindBuiltInBuildingBlocksDOTX* befindet sich im Modul *modBuildingBlocks* in der Beispieldatei *Kap06-BuildingBlocks.docm* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap06*.

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

Abbildg. 6.17 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 der Datei *Building Blocks.dotx* 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 6.20 Erstellen von Falzmarken und Speichern als benutzerdefinierter Kopfzeilen-Baustein

```

Sub Bsp_FalzmarkenBausteinErstellen()
    Dim oDoc As Document
    Set oDoc = funcBsp06_Falzmarke
    Dim objTemplate As Template
    Dim conType As WdBuildingBlockTypes
    Dim objRange As Range
    Dim intHdr As Integer
    Dim hdr As HeaderFooter
    Set objTemplate = FindBuildingBlocksDOTX
    conType = wdTypeCustomHeaders
    If oDoc.Sections(1).PageSetup.DifferentFirstPageHeaderFooter Then

```

Listing 6.20 Erstellen von Falzmarken und Speichern als benutzerdefinierten Kopfzeilen-Baustein (Fortsetzung)

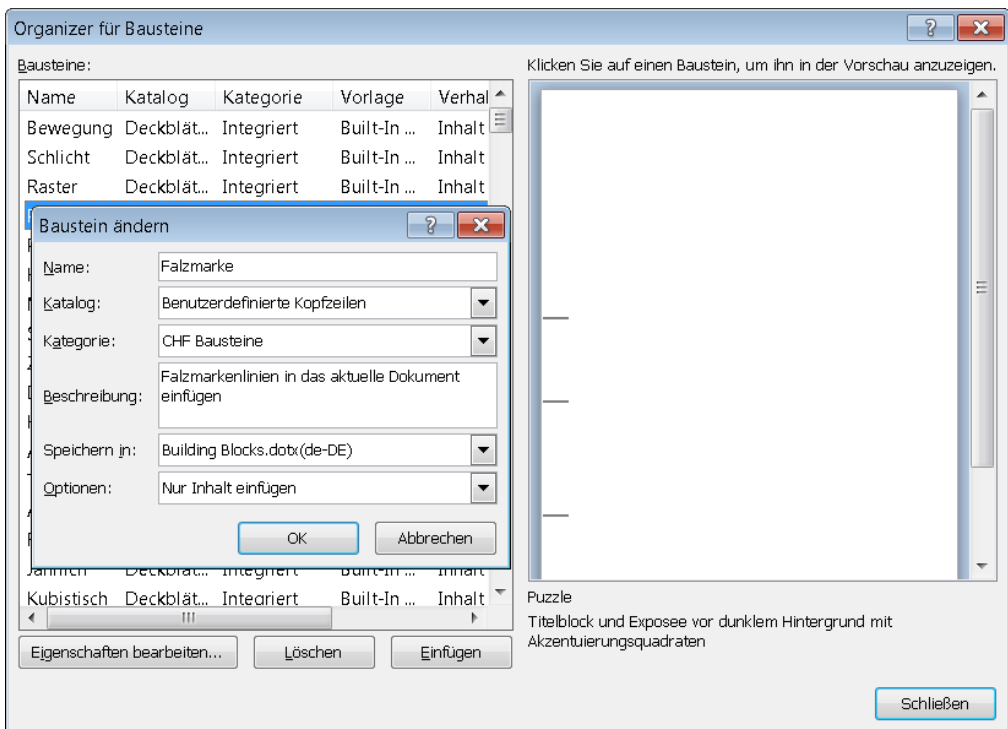
```

    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« ab Seite 280) finden Sie daraufhin den neu erstellten Baustein *Falzmarke* im Katalog *Benutzerdefinierte Kopfzeile* unter der Kategorie *CHF Bausteine*. Gespeichert ist dieser Baustein in der *Building Blocks.dotx*.

Abbildg. 6.18 Organizer mit neu erstelltem, benutzerdefinierten Kopfzeilen-Baustein



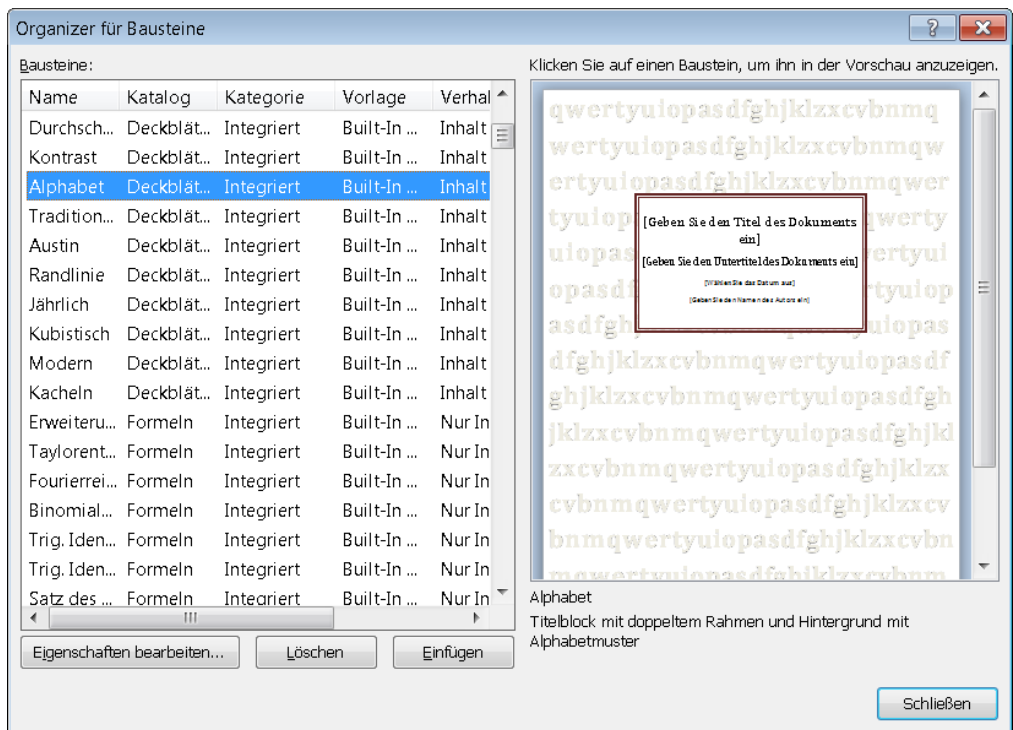
CD-ROM Das vollständige Beispiel mit allen benötigten Funktionen befindet sich im Modul *modFalzmarkeErstellen*. Sie finden das Modul in der Beispieldatei *Kap06-BuildingBlocks.docm* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap06*.

Der Organizer für Bausteine

Der Organizer für alle Bausteine, einschließlich 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 Befehl *Organizer für Bausteine* auf. Sie finden diesen im Drop-downmenü zur Schaltfläche *Schnellbausteine* auf der Registerkarte *Einfügen* in der Gruppe *Text*.

Abbildg. 6.19 Dialogfeld *Organizer für Bausteine*



Im Objektmodell erreichen Sie das entsprechende Dialogfeld über den Dialogs-Typen `wdDialogBuildingBlockOrganizer`:

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

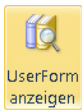
Leider lässt sich dieses Dialogfeld nicht einschränken oder per Parameter vorsortieren, da dieses Dialogs-Objekt keine Parameter besitzt.

Über die drei Schaltflächen *Eigenschaften bearbeiten*, *Löschen* und *Einfügen* rufen Sie zum einen das Eigenschaftendialogfeld auf (Abbildung 6.14), über das Sie durch Zuweisen eines anderen Speicherortes den markierten Baustein verschieben (organisieren) können, können den markierten Baustein löschen 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 diese Registerkarte öffnen, sehen Sie eine neue Gruppe mit einer Reihe von Symbolen, über die Sie die verschiedenen Beispiele zu diesem Thema aufrufen können.

Über das 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. 6.20 Übersicht über alle Bausteine mit der Möglichkeit der Anzeigeeinschränkung

Übersicht über die Bausteine (BuildingBlocks)

Vorlage:

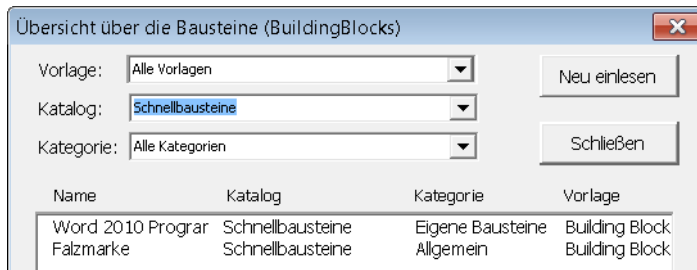
Katalog:

Kategorie:

Name	Katalog	Kategorie	Vorlage
Traditionell	Deckblätter	Integriert	Built-In Bu
Randlinie	Deckblätter	Integriert	Built-In Bu
Stapel	Deckblätter	Integriert	Built-In Bu
Schlicht	Deckblätter	Integriert	Built-In Bu
Alphabet	Deckblätter	Integriert	Built-In Bu
Jährlich	Deckblätter	Integriert	Built-In Bu
Kubistisch	Deckblätter	Integriert	Built-In Bu
Modern	Deckblätter	Integriert	Built-In Bu
Nadelstreifen	Deckblätter	Integriert	Built-In Bu
Durchscheinend	Deckblätter	Integriert	Built-In Bu
Herausgestellt	Deckblätter	Integriert	Built-In Bu
Puzzle	Deckblätter	Integriert	Built-In Bu
Bewegung	Deckblätter	Integriert	Built-In Bu
Kacheln	Deckblätter	Integriert	Built-In Bu
Kontrast	Deckblätter	Integriert	Built-In Bu
Kreisoberfläche	Formeln	Integriert	Built-In Bu
Binomischer Lehrsatz	Formeln	Integriert	Built-In Bu
Erweiterung einer St	Formeln	Integriert	Built-In Bu
Fourierreihe	Formeln	Integriert	Built-In Bu
Satz des Pythagoras	Formeln	Integriert	Built-In Bu
Quadratformel	Formeln	Integriert	Built-In Bu

Ein Baustein kann per Doppelklick an der Eingabemarke eingefügt werden

Abbildg. 6.21 Einschränkung der Anzeige auf bestimmte Bausteine (Vorlagen, Kataloge, Kategorien)



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 6.14 auf Seite 273) aufzurufen. Wenn Sie einen Baustein ändern möchten, müssen Sie dies über den Organizer vornehmen.

CD-ROM

Die in diesem Abschnitt dargestellten Beispiele finden Sie in der Beispieldatei *Kap06-BuildingBlocks.docm* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap06*.

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-Dateiressourcen von Word werden weniger strapaziert

Seit jeher unterstützt Word 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.

13

In Word 2010 gesellt sich offiziell eine neue Art Formatvorlage dazu: die verknüpfte Formatvorlage. Mit ihr können Textstellen innerhalb eines, mit einer anderen Formatvorlage formatierten Absatzes mit der gleichen Zeichenformatierung versehen werden, wie beispielsweise eine Überschrift, ohne dafür eine zusätzliche Formatvorlage erstellen zu müssen.

Eigentlich wurde die verknüpfte Formatvorlage schon in Word 2002 eingeführt mit der Absicht, die Anzahl verschiedener Formatierungen im Dokument zu reduzieren. Nur wurde sie als solche weder gekennzeichnet noch in der Dokumentation erwähnt. Sie wurde einfach automatisch im Hintergrund erstellt, sobald der Benutzer eine Auswahl Zeichen mit einer Absatzformatvorlage formatierte. Das hat für etwas Verwirrung sowie Irritation gesorgt und ist hoffentlich nun endgültig ausgestanden.

HINWEIS

Die ursprüngliche Verknüpfung von Formatvorlagen ist für die Erscheinung der berechtigten »Zchn Zchn« Einträge in Formatvorlagenlisten verantwortlich. In früheren Ausgaben dieses Buchs wurde die Problematik im Abschnitt »Wilde Formatvorlagen« beschrieben. Falls Sie mit dieser Problematik konfrontiert sind, finden Sie diese Informationen auf der CD-ROM zum Buch im Ordner `\Beilagen\Kap06_WildeFormatvorlagen`.

Ab jetzt nimmt die verknüpfte Formatvorlage ihren Platz neben den Zeichen- und Absatzformatvorlagen ein. Somit können Formatvorlagen für Zeichen, für Absätze oder für beides definiert werden. Es ist auch möglich, verknüpfte Formatvorlagen gänzlich auszuschalten. In diesem Modus können mit einer verknüpften Formatvorlage nur Absätze formatiert werden (mit anderen Worten, Word verhält sich dann wie Word 97 und frühere Versionen).

HINWEIS

In Word 2002 wurden noch 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.

Benutzerschnittstellen für Formatvorlagen

44

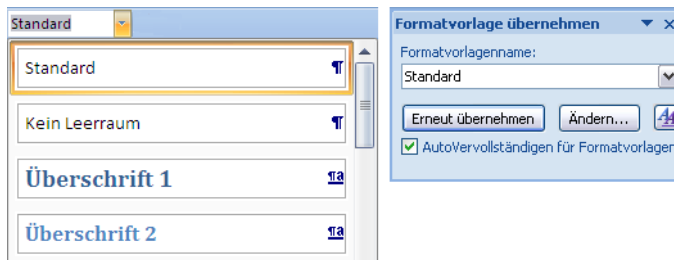
In den letzten Jahren erlebten wir einige Änderungen in den Benutzerschnittstellen für Formatvorlagen. Der alltägliche Gebrauch wurde bewusst von der Verwaltung der Formatvorlagen getrennt. Während dem professionellen Hersteller von Vorlagen mehr und wirksamere Werkzeuge zur Verfügung gestellt wurden, wollte Microsoft dem Normalbenutzer den korrekten Einsatz der Formatierung vereinfachen.

Beim Starten von Word fällt also die Liste *Schnellformatvorlage* in der Gruppe *Formatvorlagen* der Registerkarte *Start* sofort auf. Der Vorlagenverwalter kann den Inhalt dieser Liste bestimmen, sodass

der Benutzer die wichtigsten Formatierungen für das Dokument zuerst sieht und – hoffentlich – darauf zugreift.

Der Benutzer kann ferner mit dem Aufgabenbereich *Formatvorlagen* arbeiten, falls eine bessere Übersicht einer langen Liste von Formatvorlagen wünschenswert ist. Zudem steht der kleine Aufgabenbereich *Formatvorlage übernehmen* zur Verfügung, die mit der Tastaturkombination $\boxed{\uparrow} + \boxed{\text{Strg}} + \boxed{\text{S}}$ eingeblendet wird. Es ist auch möglich, die bewährte Dropdownliste aus der alten Symbolleiste *Formatieren* der Schnellzugriffsleiste hinzuzufügen. Diese letzten beiden Schnittstellen sind in Abbildung 6.22 abgebildet.

Abbildg. 6.22 Die alte Dropdownliste in der Symbolleiste für den Schnellzugriff, sowie sein Ersatz, der neue Aufgabenbereich



TIPP

Um die alte Liste der Formatvorlagen der Benutzerschnittstelle über die Word-Optionen hinzuzufügen, wählen Sie in der Liste *Befehle auswählen* den Eintrag *Befehle nicht im Menüband* aus. Suchen Sie den Eintrag *Formatvorlage* und weisen ihn der Schnellzugriffsleiste oder einer benutzerdefinierten Gruppe im Menüband zu.

Für Benutzer, die die Finger gerne auf der Tastatur behalten, besteht die Möglichkeit, jeder Formatvorlage eine Tastaturkombination zuzuweisen. Das entsprechende Dialogfeld wird erreicht über *Datei/Optionen/Menüband anpassen*, in dem die Schaltfläche *Anpassen* betätigt werden muss.

Anpassungen der Benutzerschnittstellen

Zunächst muss uns als Entwickler von Lösungen klar sein, dass der Durchschnittsanwender 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 in Word 2002, sowie Schnellformatvorlagen in Version 2007, hat zu einem gewissen Grad geholfen, 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 waren in Word 2003 und früher auch Symbolleisten mit Symbolschaltflächen für Formatvorlagen. Hierfür gibt es ab Word 2007 keinen gleichwertigen Ersatz, der ohne Programmierkenntnisse realisierbar ist. Leider können Formatvorlagen Steuerelementen im Menüband nicht direkt

zugewiesen werden. Mit dem Word-Objektmodell kann ein nicht modales UserForm zur Verfügung gestellt werden. Makros können auch für das Menüband erstellt werden, die Formatvorlagen der gegenwärtigen Markierung zuweisen. Ferner besteht für den Entwickler eines COM-Add-Ins die Möglichkeit, ein Custom Task Pane für diesen Zweck einzusetzen.

HINWEIS Eine Diskussion zur Anpassung des Aufgabenbereichs *Formatvorlagen* befindet sich im Kapitel 18. Eine kurze Einführung in VSTO und die Erstellung von Aufgabenbereichen wird in Kapitel 10 vorgestellt. Mehr über UserForms erfahren Sie in Kapitel 15. Die Anpassung des Menübands steht im Kapitel 16 beschrieben.

Abbildg. 6.23 Jede Schaltfläche des UserForm weist dem markierten Text eine Formatvorlage zu



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.

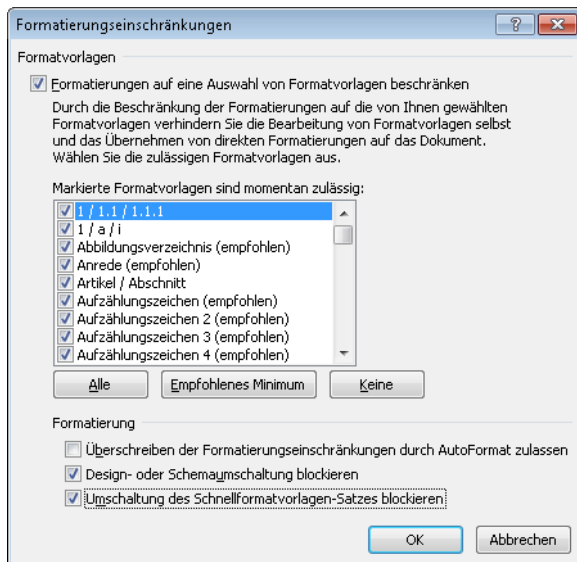
Ab Word 2003 dagegen steht eine Dokumentschutzoption zur Verfügung, um die im Dokument erlaubten Formatierungen festzulegen. In der Benutzerschnittstelle von Word 2010 befindet sie sich im Aufgabenbereich *Formatierung und Bearbeitung einschränken*, der über die Schaltfläche *Bearbeitung einschr.* der Gruppe *Schützen* in der Registerkarte *Entwicklertools* zu finden ist. (In Word 2007 ist die Schaltfläche mit *Dokument schützen* beschriftet). Durch Anklicken des Links *Einstellungen* wird das in Abbildung 6.24 dargestellte Dialogfeld *Formatierungseinschränkungen* eingeblendet. Als Folge dieser Einschränkung werden direkte Formatierungsbefehle gesperrt, und nur die aktivierten

Formatvorlageneinträge dürfen im Dokument benutzt werden. Zudem darf der Benutzer weder neue Formatvorlagen definieren, noch bestehende ändern.

Die Funktionalität ist 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. 6.24 Hier wird festgelegt, 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 Dialogfelds 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.

Auto-
Format-
Override


Das untere Kontrollkästchen steht für die Dokumentoption *AutoFormatOverride*. Ist diese Einstellung aktiviert, dürfen die Optionen *Format/AutoFormat* und *AutoFormat während der Eingabe* ausgeführt werden. (Ab Word 2007 befindet sich *AutoFormat* nicht im Menüband; der Befehl kann jedoch der Symbolleiste für den Schnellzugriff zugewiesen werden.)

Designs
& Style-
Sets

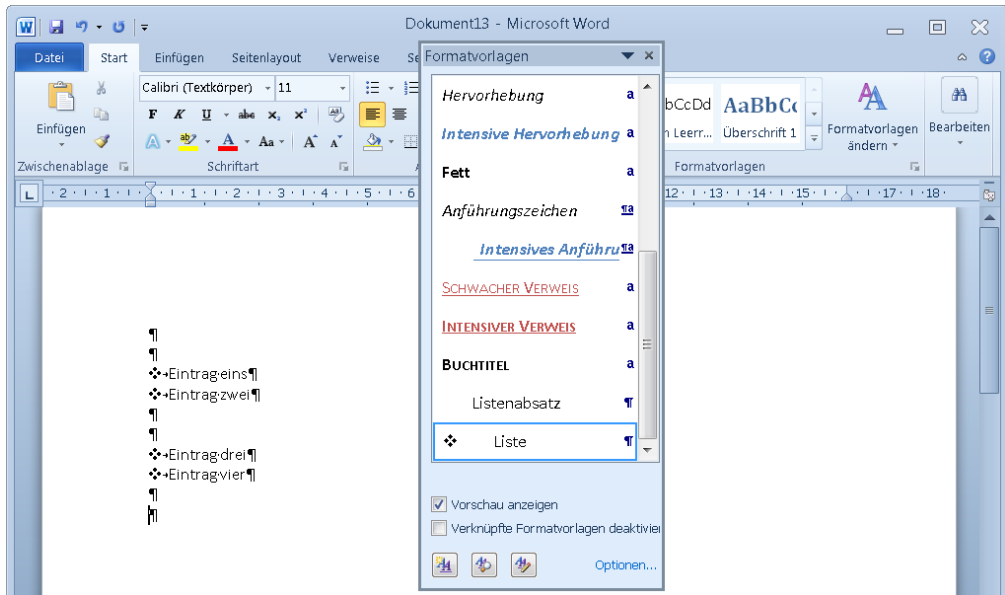
Das zweite und dritte Kontrollkästchen beziehen sich auf *Designs* (der erste Schalter in der Registerkarte *Seitenlayout*) sowie Schnellformatvorlagen-Sätze (der Schalter *Formatvorlagen ändern* in der Registerkarte *Start*). Sie entsprechen den Eigenschaften *LockTheme* sowie *LockQuickStyleSet* des *Document-Objekts*.

Bitte beachten Sie, dass diese drei Optionen eingeschaltet werden, sobald das Dialogfeld mit *OK* bestätigt wird. Ob der Dokumentschutz eingeschaltet oder das Kontrollkästchen *Formatierung auf eine Auswahl von Formatvorlagen* aktiviert ist, spielt keine Rolle. Um die Restriktion auszuschalten, müssen diese Optionen im Dialogfeld deaktiviert 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 6.25 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. 6.25 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 6.21 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 6.21 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 6.22 (.NET): Die C#-Version von Listing 6.21



```
private void NurTextZulassen_CS()
{
    object objTrue = (object) true;
    object objFalse = (object) false;
    wd.Document doc = wdApp.ActiveDocument;
    foreach (wd.Style styl in doc.Styles)
```

Listing 6.22 (.NET): Die C#-Version von Listing 6.21

```

{
    styl.Locked = true;
    if (styl.InUse)
    {
        switch (styl.NameLocal)
        {
            case "Nur Text":
                //Diese werden im Dokument beibehalten.
            case "Überschrift 1":
            case "Überschrift 2":
            case "Überschrift 3":
                //Können nicht gelöscht werden, also ersetzen.
                wd.Range rng = doc.Content;
                rng.Find.ClearFormatting();
                rng.Find.Replacement.ClearFormatting();
                rng.Find.Format = true;
                object objStyl = (object) styl.NameLocal;
                rng.Find.set_Style(ref objStyl);
                object objReplaceStyle = (object) wd.WdBuiltinStyle.wdStyleNormal;
                rng.Find.Replacement.set_Style(ref objReplaceStyle);
                object objReplaceText = (object) "";
                object objFindText = (object) "";
                object objWrapStop = (object) wd.WdFindWrap.wdFindStop;
                object objReplaceAll = (object) wd.WdReplace.wdReplaceAll;
                rng.Find.Execute(ref objFindText, ref objFalse, ref objFalse, ref objFalse,
                    ref objFalse, ref objFalse, ref objTrue, ref objWrapStop, ref objTrue,
                    ref objReplaceText, ref objReplaceAll, ref objFalse,
                    ref objFalse, ref objFalse, ref objFalse);
                break;
            case "Standard":
            case "Normale Tabelle":
            case "Absatz-Standardschriftart":
            case "Keine Liste":
                //Können nicht gelöscht werden.
                break;
            default:
                styl.Delete();
                break;
        }
    }
}
doc.AutoFormatOverride = false;
//Nur die Formatvorlage 'Nur Text' darf im Dokument benutzt werden.
object objStyleNurText = wd.WdBuiltinStyle.wdStylePlainText;
doc.Styles.get_Item(ref objStyleNurText).Locked = false;
//direkte Formatierungen entfernen
doc.Content.Font.Reset();
object objPassword = (object) "";
doc.Protect(wd.WdProtectionType.wdNoProtection, ref objFalse, ref objPassword,
    ref objFalse, ref objTrue);
}

```

WICHTIG Das Sperren einer Formatvorlage gilt nur für die Benutzerschnittstelle. Über das Objektmodell können Sie weiterhin Formatvorlagen erstellen und ändern, ohne den Dokumentschutz aufzuheben.

Word-interne Formatvorlagen

Ein von der unveränderten *Normal.dotm* erstelltes Dokument enthält mehr als 250 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 Formatvorlagenname 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. die 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.

Name-
Local

Eine bestimmte Formatvorlage wird mit einem Indexwert angesprochen. Dabei kann es sich um eine Ganzzahl (Position in der Auflistung) handeln, um den Namen, wie er in der Umgebung erscheint (*NameLocal*-Eigenschaft), sowie, für Words interne Formatvorlagen, um einen *WdBuiltinStyle*-Konstantwert. Eine Liste der *WdBuiltinStyle*-Konstantwerte finden Sie im Objektkatalog des VB-Editors; die Enumeration entspricht in ungefähr den englischen Namen. Um herauszufinden, welcher Konstantwert zu welcher Formatvorlage passt:

```
MsgBox ActiveDocument.Styles(wdStyleNormal).NameLocal
'Gibt "Standard" zurück in einer deutschen Umgebung
'Gibt "Normal" zurück in einer englischen Umgebung
```

Arbeiten Ihre Anwendung mit den 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 6.23 Alle nicht Word-internen Formatvorlagen in einem neuen Dokument auflisten

```

Sub AlleNichtWordFVAuflisten()
    Dim doc As Word.Document
    Dim docNeu As Word.Document
    Dim styl As Word.Style
    Dim rng As Word.Range

    Set doc = ActiveDocument
    Set docNeu = Documents.Add
    Set rng = docNeu.Content
    For Each styl In doc.Styles
        If Not styl.BuiltIn Then
            rng.InsertAfter styl.NameLocal & vbCrLf
        End If
    Next
End Sub

```

Listing 6.24 (.NET): Die C#-Version von Listing 6.23



```

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");
        }
    }
}

```

CD-ROM

Die in diesem Abschnitt dargestellten Beispiele finden Sie in der Beispieldatei *Bsp06_01_Style.docm* 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, stellt sich eher die Frage, wie allen Benutzern möglichst aussagekräftige 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 6.25 führt die Angaben im Code auf. Die `AutoOpen`-Prozedur wird beim Öffnen des Dokuments automatisch ausgeführt.

Listing 6.25 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
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"

```


Listing 6.25 Die Namen von Formatvorlagen der Umgebungssprache anpassen (Fortsetzung)

```

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

Private Function SprachIndexHolen(ByVal lSprache As Long) As Long
    Dim lIndex As Long

    Select Case lSprache
        Case 1031, 3079, 5127, 4103, 2055
            lIndex = INDEX_DEUTSCH
        Case 4108, 1036, 5132
            lIndex = INDEX_FRANZ
        Case 1033, 2057, 615
            lIndex = INDEX_ENGLISCH
        'Unerwartete Sprache
        Case Else
            End Select
        SprachIndexHolen = lIndex
    End Function

```

CD-ROM

Die in diesem Abschnitt dargestellten Beispiele finden Sie in der Beispieldatei *Bsp06_02_Style.docm* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap06*.

Formatvorlagen erstellen und modifizieren

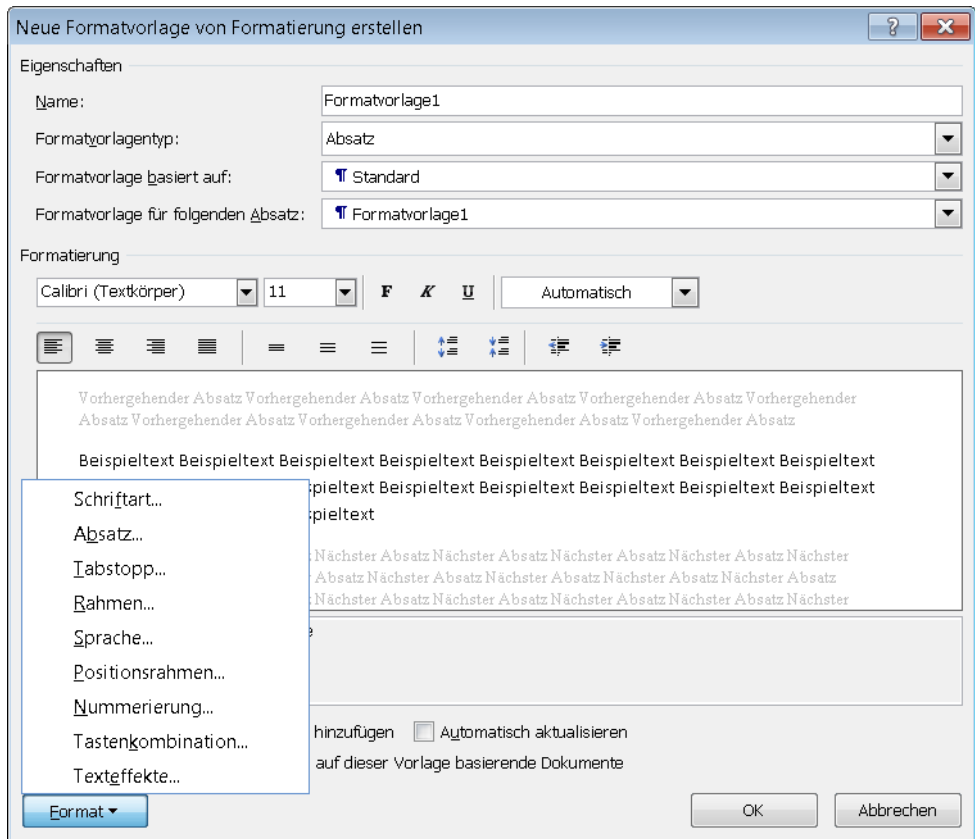
Meistens werden Formatvorlagen in Dokumentvorlagen erstellt und gespeichert. Alle von der Dokumentvorlage erstellten Dokumente erben automatisch deren Formatvorlagen. Deshalb kommt es im Entwickleralltag vergleichsweise selten vor, dass Formatvorlagen mit Code erstellt werden müssen, außer wenn die Anwendung 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`-Konstantwerte (Word 2000 erkennt nur die beiden ersten): `WdStyleTypeParagraph` (Absatz), `WdStyleTypeCharacter` (Zeichen), `WdStyleTypeList` (Nummerierung) oder `WdStyleTypeTable` (Tabelle).

In Word 2007 kamen dazu die Typen `WdStyleTypeLinked` (verknüpft = kann als Zeichen und als Absatzformatvorlage dienen) und `WdStyleTypeParagraphOnly` (nur Absatz = nicht verknüpft). Verknüpfte Formatvorlagen wurden im Abschnitt »Formatvorlagen« ab Seite 282 näher vorgestellt. Die Option, sie auszuschalten, entspricht im Objektmodell der Eigenschaft `Application.RestrictLinkedStyles`.

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 6.26) mit den Menüs unter der Schaltfläche *Format* statt. Da diese den Dialogfeldern in den Einträgen des alten 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 6.26 zeigt.

Abbildg. 6.26 Das Dialogfeld, um eine Formatvorlage zu erstellen oder neu zu definieren


Darin werden zuerst alle Word-eigenen Formatvorlagen gesperrt und anschließend zwei neue Formatvorlagen erstellt. Bitte beachten Sie, dass die erste Formatvorlage, »Bericht Fließtext«, automatisch der zweiten, »Bericht Überschrift«, folgt, wenn die -Taste gedrückt wird (NextParagraphStyle-Eigenschaft). Basiert eine Formatvorlage auf einer anderen oder folgt eine einer anderen, muss diese Formatvorlage schon vorhanden sein, sonst erfolgt eine Fehlermeldung. Deshalb werden diese Formatvorlagen in dieser Reihenfolge definiert.

Die Definitionen werden in einer getrennten Prozedur den Formatvorlagen zugewiesen. Aus Platzgründen haben wir für das Beispiel nur einige der möglichen Eigenschaften als Argumente von *NeueAbsatzFormatvorlage* aufgelistet, um eine Idee der Vorgehensweise zu liefern. Es ist erkennbar, dass die Eigenschaften auf genau die gleiche Art und Weise benutzt werden, als würden Sie mit einem Range- oder Selection-Objekt arbeiten. Sie dürfen alle Formatierungsbefehle benutzen, die unter dem Menü *Format* des Dialogfelds *Formatvorlage erstellen* in Abbildung 6.26 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 6.27 zu sehen ist.

Listing 6.26 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
    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.TaskPanes(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 6.27 (.NET): Die C#-Version von Listing 6.26


```
private void Listing6_27_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,
        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";
```

Listing 6.27 (.NET): Die C#-Version von Listing 6.26 (Fortsetzung)

```

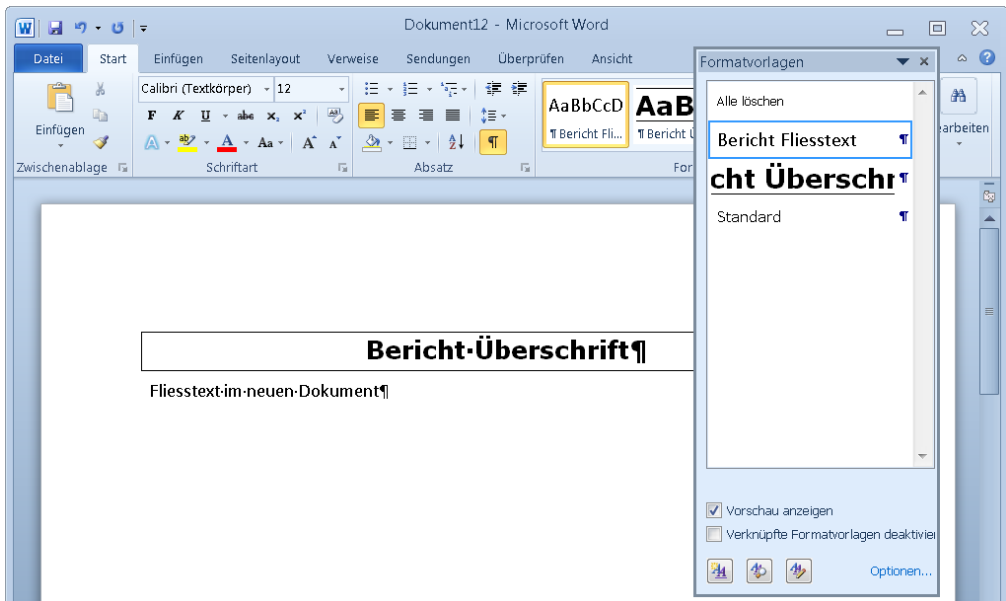
rng.Text = "Fliesstext im neuen Dokument.";
rng.set_Style(ref sty1Fliesstext);
}

```

CD-ROM

Die in diesem Abschnitt dargestellten Beispiele finden Sie in der Beispieldatei *Bsp06_03_Style.docm* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap06*.

Abbildg. 6.27 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.

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.

Während den letzten Jahren haben die Entwickler an dem grundlegenden Konzept nichts geändert, sondern versucht, das Verhalten mehr an die Erwartungen der Anwender anzupassen. Mit begrenztem Erfolg.

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 Kollaps 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 Codefragment 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 Listenebene (`ListLevel`). Diese werden in Listing 6.28 und in Abbildung 6.28 veranschaulicht.

Listing 6.28 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)." & _
```

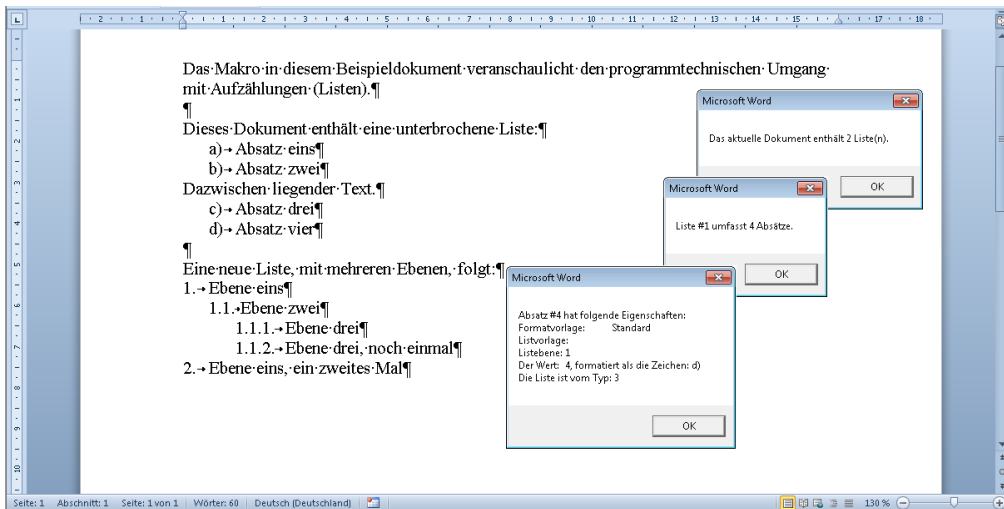
Listing 6.28 Eigenschaften von Listen in einem Dokument ermitteln (Fortsetzung)

```

vbCr & "Liste #" & CStr(lListZaehler) & " umfasst " & _
CStr(lAnzParas) & " Absätze."
For lParaZaehler = 1 To lAnzParas
    Set para = lst.ListParagraphs(lParaZaehler)
    sFormatvorlage = para.Style
    With para.Range.ListFormat
        sListvorlage = .ListTemplate.Name
        sVisuelleZiffer = .ListString
        sOrdinalZahl = CStr(.ListValue)
        sListEbene = CStr(.ListLevelNumber)
        sListTyp = CStr(.ListType)
    End With
    MsgBox "Absatz #" & CStr(lParaZaehler) & " hat folgende Eigenschaften:" & _
        & vbCr & "Formatvorlage:" & vbTab & sFormatvorlage & vbCr & _
        "Listvorlage:" & vbTab & sListvorlage & vbCr & "Listebene:" & vbTab & _
        sListEbene & vbCr & "Der Wert:" & vbTab & sOrdinalZahl & _
        ", formatiert als die Zeichen:" & sVisuelleZiffer & vbCr & _
        "Die Liste ist vom Typ " & sListTyp
Next
Next
End Sub

```

Abbildg. 6.28 Eigenschaften von Listen in einem Dokument ermitteln



CD-ROM

Die in diesem Abschnitt dargestellten Beispiele finden Sie in der Beispieldatei *Bsp06_01_Num.docm* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap06*.

Mit diesen Eigenschaften können Informationen über die Nummerierung eines Dokuments ermittelt werden. Sie sollen jedoch nicht dazu gebraucht werden, um Nummerierungen vorzunehmen oder zu ändern. Statt direkt auf den Text einzuwirken, soll Automatisierungscode direkter mit den Verwaltungsstrukturen arbeiten.

Listvorlagen (*ListTemplates*)

Im Hintergrund kennzeichnet Word jeden nummerierten Absatz mit einem Vermerk, zu welcher Liste er gehört. Die Listen werden im Dokument an einer zentralen Stelle geführt. Der Absatz schlägt an dieser Stelle nach, wie das Listenelement (also sich selbst) anzuzeigen ist.

HINWEIS Verhält sich ein Listenelement »komisch« (falsche Formatierung oder Nummerierung), wurde es wahrscheinlich intern mit einem anderen Listenobjekt verbunden.

Listen ihrerseits beziehen ihre Formatierungseinstellungen aus sogenannten *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«. Fast jedes Mal, wenn er auf eine Vorschau in einer der drei Listen der Gruppe *Absatz (Aufzählungszeichen, Nummerierung bzw. Liste mit mehreren Ebenen)* 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. Ab Word 2002 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 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« Listenvorlagen erstellt werden oder in der Dokumentvorlage vorhanden sind. Dies bedingt, dass die Dokumentvorlage von einer »sauberen« *Normal.dotm* erstellt wird, die garantiert keine Listvorlagen enthält. Wir raten also, die aktuelle *Normal.dotm* umzubenennen, sodass Word beim Starten ein sauberes Exemplar erstellt, worauf dann die neue Dokumentvorlage basiert.

HINWEIS In Kapitel 13 wird die Vorbereitung der Dokumentvorlage *Normal.dotm* besprochen, die allen Word-Dokumenten zugrunde 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 Listvorlagen schleifen, wie die Funktion in Listing 6.29 veranschaulicht. Falls eine Listvorlage mit der angegebenen Bezeichnung bereits vorhanden ist, wird diese zurückgegeben. Sonst wird eine neue erstellt und zurückgegeben.

Listing 6.29 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.

Outline-Num-bered 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 6.30 bzw. das Listing 6.31. 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 6.30 Eine Listvorlage definieren

```
Sub AufzählungslisteErstellen()
    Dim doc As Word.Document
    Dim LTAufzählung As Word.ListTemplate
    Dim sLTName As String
    Dim lAufzählungZeichencode As Long
    Dim LL As Word.ListLevel

    Set doc = ActiveDocument
    sLTName = "Pfeil-Aufzählung"
    lAufzählungZeichencode = 220 'Wingdings
    Set LTAufzählung = ListTemplateIndex(sLTName, doc)
    LTAufzählung.OutlineNumbered = False
    Set LL = LTAufzählung.ListLevels(1)
    ListLevelFormatierungFestlegen LL, wdListLevelAlignLeft, wdListNumberStyleNone, _
        ChrW$(lAufzählungZeichencode), 0, CentimetersToPoints(0.7), _
        CentimetersToPoints(0.7), wdTrailingTab, False, 1, "Wingdings", False, _
        "Meine Aufzählung"
End Sub

Private Sub ListLevelFormatierungFestlegen(LL As Word.ListLevel, _
    LL_Ausrichtung As WdListLevelAlignment, LL_NummerArt As WdListNumberStyle, _
    LL_NummerFormat As Long, LL_NummerPos As Single, LL_TabPos As Single, _
    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 6.31 (.NET): Die C#-Version von Listing 6.30



```

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.
    //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;
    }
}

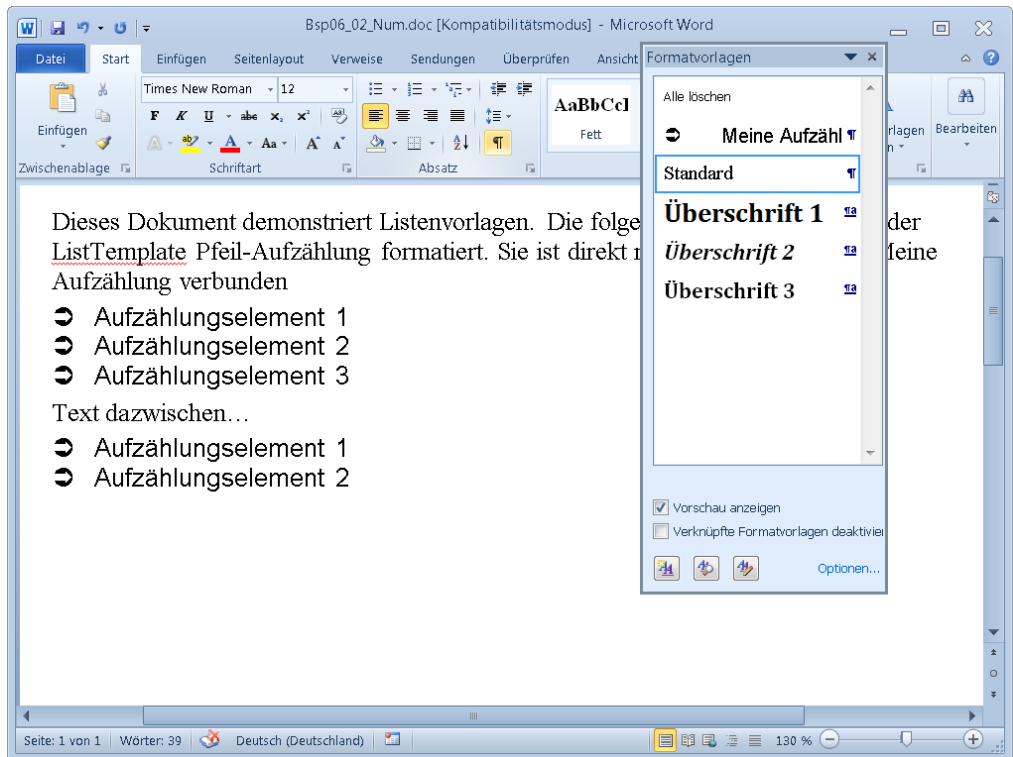
```

Listing 6.31 (.NET): Die C#-Version von Listing 6.30 (Fortsetzung)

```

        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.

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

ListLevel-Eigenschaft	Mögliche Werte/Bemerkungen
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.
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 Single , 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 Single , 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 TabPosition anzuspringen. Diese und TextPosition haben keine Wirkung, wenn TrailingCharacter nicht wdTrailingTab 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.

CD-ROM Die in diesem Abschnitt dargestellten Beispiele finden Sie in der Beispieldatei *Bsp06_02_Num.docm* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap06*.

Eine Gliederungsliste der anderen Art

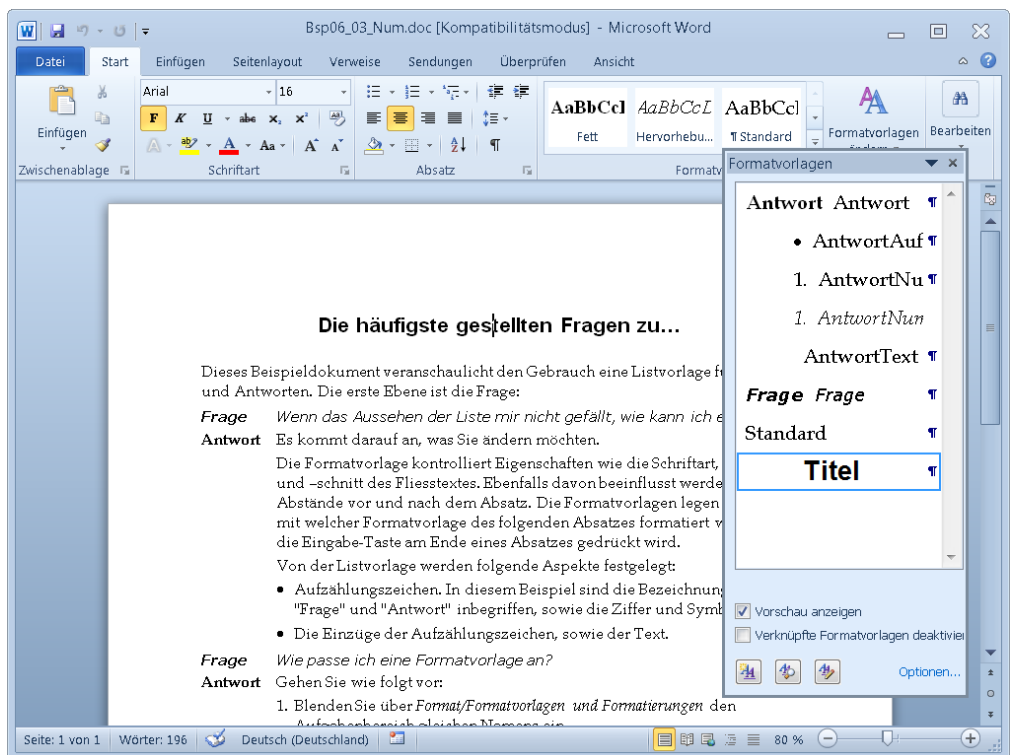
Oft wird gefragt, ob statischer Text als Teil einer Formatvorlage definiert werden könnte. Leider lautet die Antwort darauf: Nein. Einen Umweg gibt es dennoch, wenn eine Listvorlage mit der Formatvorlage verbunden wird. Aus der Erläuterung zur NumberFormat-Eigenschaft ist hervorgegangen, dass statischer 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 6.32 erstellt. Im Gegensatz zu Listing 6.30 wird eine Listvorlage des Typs Gliederung festgelegt, die neun Listebenen zur Verfügung stellt. Derer 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.

Abbildg. 6.30 Die ersten fünf Ebenen der Listvorlage definieren die Elemente eines Fragen-und-Antwort-Dokuments



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 *AntwortNum*-

meriert-Formatierung angewendet werden. Ein nochmaliges Drücken formatiert den Absatz mit *AntwortAufzählung*. Die Tastenkombination  +  +  wechselt zu einer höheren Listenebene.

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, sodass der Benutzer nur die vordefinierten Listvorlagen verwendet.

Listing 6.32 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 6.33 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 6.33 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
```

Listing 6.33 Die Fragen und erster Absatz jeder Antwort als Zusammenfassung in ein neues Dokument übernehmen (*Fortsetzung*)

```
Set docNeu = Application.Documents.Add
Set rng = docNeu.Content
For Each para In docQuelle.Lists(1).ListParagraphs
    If para.Range.ListFormat.ListLevelNumber <= 2 Then
        rng.FormattedText = para.Range.FormattedText
        'Da diese Auflistung vom Dokumentende bis zum Dokumentanfang
        'durchschleift wird, wird jeder zusätzlicher Absatz am
        'Anfang des Bereichs eingefügt.
        rng.Collapse wdCollapseStart
    End If
Next
End Sub
```

CD-ROM Die in diesem Abschnitt dargestellten Beispiele finden Sie in der Beispieldatei *Bsp06_03_Num.docm* 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.

Für Office 2007 wurde die Grafikfunktionalität 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.

Diese verwirrende Situation wurde in Version 2010 etwas gelichtet, was aber zum Verschwinden von einigen bekannten und »altverdienten« Grafikwerkzeugen geführt hat. Wir werden in diesem Abschnitt aus Platzgründen nicht allen Änderungen in der Benutzerschnittstelle nachgehen, sondern uns hauptsächlich dem Umgang mit dem Objektmodell widmen.

Inline-
Shape vs.
Shape

Es gibt zwei verschiedene Methoden, eine Grafik in ein Dokument einzubetten: in der Zeile mit dem Text oder mit einer Textflussformatierung. Steht eine Grafik in der Zeile mit dem Text, ist sie Teil der *InlineShapes*-Auflistung, Word behandelt sie grundsätzlich wie ein Textzeichen, und die VBA-Befehle gehören dem Word-Objektmodell an. Wurde sie jedoch mit Textfluss formatiert, ist sie frei platzierbar auf der Seite, sie steht in der Zeichnungsebene des Dokuments, ist Teil der *Shapes*-Auflistung, und die passenden Befehle befinden sich hauptsächlich im Office-Objektmodell, da Zeichnungsobjekte Office-übergreifend sind.

Seit Word 2002 darf der Benutzer selbst bestimmen, mit welcher Art Positionierung eine Grafik neu eingefügt wird. Die Einstellung *Bilder einfügen als* befindet sich in den *Optionen* innerhalb der Kategorie *Erweitert* im Abschnitt *Ausschneiden, Kopieren und Einfügen*. Folglich muss der Code, wollen Sie alle Grafiken eines Dokuments bearbeiten, beide Auflistungen berücksichtigen.

Grafiken einfügen

Die mit dem Makrorekorder erzielten Ergebnisse der Einfügung und Formatierung einer Grafik sind unterschiedlich. Allgemein kann gesagt werden, dass Handlungen mit `InlineShapes` aufgezeichnet werden, während (vor allem in Word 2010) der Makrorekorder für `Shapes` wenig Informationen liefert.

Während in früheren Word-Versionen ein `InlineShape` von einem `Shape` durch die Farbe der Anfasser schnell erkennbar war, ist dies ab Word 2007 nur im Kompatibilitätsformat möglich. Grafiken, die in ein OpenXML-Dokument (*.docx*) eingefügt wurden, sind visuell nur durch Prüfen der Textumbruchformatierung erkennbar, oder bei Sichtbarkeit des Anchor-Symbols neben einer Absatzmarke.

Aus diesen Gründen gestaltet sich die Erforschung des Objektmodells für Grafiken nicht einfach.

Aus Datei mit `AddPicture`

Beim Vergleich der zwei Routinen in Listing 6.34 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).

Listing 6.34 Dieser aufgezeichnete Code gibt Aufschluss über die *InlineShape*- bzw. *Shape*-Objekte

```
Sub InlineShapeImportierenUndBearbeiten()
    Selection.InlineShapes.AddPicture FileName:="C:\Beispiele\Kap06\Soap Bubbles.bmp", _
        LinkToFile:=False, SaveWithDocument:=True
    Selection.MoveLeft Unit:=wdCharacter, Count:=1, Extend:=wdExtend
    Selection.InlineShapes(1).Fill.Visible = msoFalse
    Selection.InlineShapes(1).Fill.Solid
    Selection.InlineShapes(1).Fill.Transparency = 0#
    Selection.InlineShapes(1).Line.Weight = 0.75
    Selection.InlineShapes(1).Line.Transparency = 0#
    Selection.InlineShapes(1).Line.Visible = msoFalse
    Selection.InlineShapes(1).LockAspectRatio = msoTrue
    Selection.InlineShapes(1).Height = 113.4
    Selection.InlineShapes(1).Width = 113.4
    Selection.InlineShapes(1).PictureFormat.Brightness = 0.5
    Selection.InlineShapes(1).PictureFormat.Contrast = 0.5
    Selection.InlineShapes(1).PictureFormat.ColorType = msoPictureAutomatic
    Selection.InlineShapes(1).PictureFormat.CropLeft = 0#
    Selection.InlineShapes(1).PictureFormat.CropRight = 0#
    Selection.InlineShapes(1).PictureFormat.CropTop = 0#
    Selection.InlineShapes(1).PictureFormat.CropBottom = 0#
End Sub

Sub ShapeImportierenUndBearbeiten()
    ActiveDocument.Shapes.AddPicture(Anchor:=Selection.Range, FileName:= _
        "C:\Beispiele\Kap06\Zapotek.bmp", LinkToFile:=False, SaveWithDocument:=True). _
        WrapFormat.Type = wdWrapSquare
    ActiveDocument.Shapes(1).Select
```

Listing 6.34 Dieser aufgezeichnete Code gibt Aufschluss über die *InlineShape*- bzw. *Shape*-Objekte (Fortsetzung)

```
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
```

Beide Prozeduren arbeiten mit dem *Selection*-Objekt, weil der Code ursprünglich mit dem Makrorekorder einer früheren Word-Version aufgezeichnet wurde. Wie in Kapitel 5 erwähnt, soll Code mit dem *Selection*-Objekt möglichst vermieden werden. Listing 6.35 bzw. Listing 6.36 zeigt den bereinigten Code. Da die *AddPicture*-Methode immer ein Objekt zurückgibt, ist es einfach, eine entsprechende Objektvariable 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* (Abbildung 6.34), die über den Eintrag *Weitere Layoutoptionen* der Dropdownmenüs zu den Schaltflächen *Position* und *Textumbruch* (Zeilenumbruch in Word 2010) auf der Registerkarte *Bildtools/Format* der Gruppe *Anordnen*.)

**ACHTUNG**

In Word 2010 gibt es ein etwas merkwürdiges Verhalten: Falls die Markierung sich auf der ersten Seite des Dokuments befindet, wird das Shape-Objekt oben, relativ zum ersten Absatz eingefügt. Auf allen anderen Seiten wird es korrekterweise an der markierten Stelle erscheinen. Für diesen Fall finden Sie weitere Hinweise im Abschnitt »Die Positionierung von Shape-Objekten« ab Seite 322.

Listing 6.35

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 6.36

(.NET): Die C#-Version von Listing 6.35



```
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)
{
    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);
}
```

Listing 6.36 (.NET): Die C#-Version von Listing 6.35 (Fortsetzung)

```

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);
}

```

CD-ROM Die in diesem Abschnitt dargestellten Beispiele finden Sie in der Beispieldatei *Bsp06_01_Graf.docm* 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. Standardmäßig übernimmt Word die Textflussformatierung der Ursprungsumgebung. Eine eingefügte Grafik mit Textfluss (Shape) ist markiert und kann einer Objektvariablen direkt zugewiesen werden. Bei einem *InlineShape* steht die Einfügemarke unmittelbar rechts daneben. Zuerst muss also die Art Grafikobjekt ermittelt werden, bevor das eingefügte Objekt einer Objektvariablen zugewiesen werden kann, wie Listing 6.37 veranschaulicht.

Listing 6.37 Eine eingefügte Grafik einer Objektvariablen zuweisen

```

Sub MitEingefügteGrafikAlsShapeArbeiten()
    Dim sel As Word.Selection
    Dim ils As Word.InlineShape
    Dim shp As Word.Shape

    Set sel = Selection
    sel.Paste
    Select Case sel.Type
        Case wdSelectionInlineShape
            Set ils = Selection.InlineShapes(1)
            Set shp = ils.ConvertToShape
        Case wdSelectionShape
            Set shp = Selection.ShapeRange(1)
        Case Else
            'Die Markierung enthält keine Grafik.
            'Testen, ob ein InlineShape eingefügt wurde;
            'das Zeichen links neben der Markierung erfassen.
            sel.Collapse wdCollapseStart
            sel.MoveStart wdCharacter, -1
            If sel.Type = wdSelectionInlineShape Then
                Set ils = Selection.InlineShapes(1)
                Set shp = ils.ConvertToShape
            Else
                'Es ist keine Grafik, also aussteigen
                MsgBox ("Die Markierung enthält keine Grafik.")
                Exit Sub
            End If
        End If
    End Select
End Sub

```

Listing 6.37 Eine eingefügte Grafik einer Objektvariablen zuweisen (Fortsetzung)

```

End Select
shp.Name = "Eingefügte Grafik"
End Sub

Sub MitEingefügteGrafikAlsInlineShapeArbeiten()
    Dim sel As Word.Selection
    Dim ils As Word.InlineShape
    Dim shp As Word.Shape

    Set sel = Selection
    sel.Paste
    Select Case sel.Type
        Case wdSelectionInlineShape
            Set ils = Selection.InlineShapes(1)
        Case wdSelectionShape
            Set shp = Selection.ShapeRange(1)
            Set ils = shp.ConvertToInlineShape
        Case Else
            'Die Markierung enthält keine Grafik.
            'Testen, ob ein InlineShape eingefügt wurde;
            'das Zeichen links neben der Markierung erfassen.
            sel.Collapse wdCollapseStart
            sel.MoveStart wdCharacter, -1
            If sel.Type = wdSelectionInlineShape Then
                Set ils = Selection.InlineShapes(1)
            Else
                'Es ist keine Grafik, also aussteigen
                MsgBox ("Die Markierung enthält keine Grafik.")
                Exit Sub
            End If
        End Select
        ils.Range.Bookmarks.Add "EingefügteGrafik", ils.Range
    End Sub

```

Grafikart
bestim-
men

Der Benutzer kann eine Grafik auch über *Inhalte einfügen* (im Dropdownmenü zur Schaltfläche *Einfügen* auf der Registerkarte *Start*) ins Dokument holen. In diesem Fall kommt die Einstellung der Option *Bilder einfügen als* zum Zug, um die Positionierung der Grafik zu bestimmen. Im Objektmodell entspricht *Inhalte einfügen* der *PasteSpecial*-Methode, deren *Placement*-Parameter die Möglichkeit bietet, die Grafik als *InlineShape* oder *Shape* einzufügen. Das Verhalten der Markierung ist gleich, wie bei der üblichen Einfügung. Das heißt, bei einem *InlineShape* muss der Code die neu eingefügte Grafik zuerst markieren, erst dann kann das markierte Objekt einer Objektvariablen zugewiesen werden (Listing 6.38). Die Grafikart wird mithilfe der Parameter *DataType* festgelegt, die ein Mitglied der Enumeration *WdPasteDataType* erwartet (Tabelle 6.10).

Tabelle 6.10 Die in der *PasteSpecial*-Methode zur Verfügung stehenden Dateiformate

WdPasteDataType-Konstantwerte	Wert	Beschreibung
wdPasteBitmap	4	Bitmap
wdPasteDeviceIndependentBitmap	5	Geräteunabhängiges Bitmap
wdPasteEnhancedMetafile	9	Enhanced-Metafile

Tabelle 6.10 Die in der PasteSpecial-Methode zur Verfügung stehenden Dateiformate (Fortsetzung)

WdPasteDataType-Konstantwerte	Wert	Beschreibung
wdPasteHTML	10	HTML
wdPasteHyperlink	7	Hyperlink
wdPasteMetafilePicture	3	Metafile-Picture
wdPasteOLEObject	0	OLE-Objekt
wdPasteRTF	1	Rich-Text-Format (RTF)
wdPasteShape	8	Shape
wdPasteText	2	Text

Listing 6.38 Grafiken über die Methode *PasteSpecial* einfügen

```

Sub InhalteEingefuegtesInlineShapeErfassen()
    Dim rng As Word.Range
    Dim ils As Word.InlineShape

    On Error GoTo Fehlerbehandlung
    Set rng = Selection.Range
    rng.PasteSpecial Placement:=wdInLine, DataType:=wdPasteMetafilePicture
    'Testen, ob ein InlineShape eingefügt wurde;
    'das Zeichen links neben dem Bereich erfassen.
    rng.Collapse wdCollapseStart
    rng.MoveStart wdCharacter, -1
    Set ils = rng.InlineShapes(1)
    ils.Range.Bookmarks.Add "EingefuegteGrafik", ils.Range

Fertigstellen:
    Exit Sub

Fehlerbehandlung:
    Select Case Err.Number
        Case 5342 'Unbekannter Datentyp beim Einfügen
            MsgBox "Keine Grafik wurde in der Zwischenablage gefunden.", _
                vbOKOnly + vbCritical, "Kapitel 6 - Grafiken"
        Case Else
            MsgBox "Es gab den folgenden unerwarteten Fehler in der Prozedur " & _
                "InhalteEingefuegtesInlineShapeErfassen:" & vbCr & vbCr & Err.Number & _
                vbCr & vbCr & Err.Description, vbCritical + vbOKOnly, "Kapitel 6 - Grafiken"
    End Select
End Sub

Sub InhalteEingefuegtesShapeErfassen()
    Dim shp As Word.Shape
    Dim rng As Word.Range

    On Error GoTo Fehlerbehandlung
    Set rng = Selection.Range
    rng.PasteSpecial Placement:=wdFloatOverText, DataType:=wdPasteMetafilePicture
    Set shp = rng.ShapeRange(1)
    shp.Name = "Eingefuegte Grafik"

```

Listing 6.38 Grafiken über die Methode *PasteSpecial* einfügen (Fortsetzung)

```

Fertigstellen:
Exit Sub

Fehlerbehandlung:
Select Case Err.Number
Case 5342 'Unbekannter Datentyp beim Einfügen
    MsgBox "Keine Grafik wurde in der Zwischenablage gefunden.", _
        vbOKOnly + vbCritical, "Kapitel 6 - Grafiken"
Case Else
    MsgBox "Es gab den folgenden unerwarteten Fehler in der Prozedur " & _
        "InhalteEingefuegtesShapeErfassen:" & vbCrLf & vbCrLf & Err.Number & vbCrLf
        & vbCrLf & Err.Description, vbCritical + vbOKOnly, "Kapitel 6 - Grafiken"
End Select
End Sub

```

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.



Beim Einfügen einer Grafik in Word 2010 wird dem Benutzer die Möglichkeit geboten, die Grafikpositionierung zu beeinflussen. Die »Schaltfläche für Einfügeoptionen« in Abbildung 6.31 listet die Einträge *Ursprüngliche Formatierung beibehalten* sowie *Grafik* auf. Letztere formatiert die Grafik mit der Textflussformatierung *Quadrat* und positioniert sie weiter unten und im rechten Drittel der Seite.

Abbildg. 6.31 Optionen für die Textflussformatierung einer Grafik beim Einfügen in der Benutzerschnittstelle



Im Objektmodell entsprechen diese Optionen den Parametern `wdFormatOriginalFormatting` sowie `wdChartPicture` der Methode `PasteAndFormat`.

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 Listing 6.40.

Es ist auch möglich, 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.

CD-ROM Die in diesem Abschnitt dargestellten Beispiele finden Sie in der Beispieldatei *Bsp06_01_Graf.docm* 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 6.35 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 6.11 aufgelistet.

Tabelle 6.11 Eigenschaften für die Verwaltung von Verknüpfungen

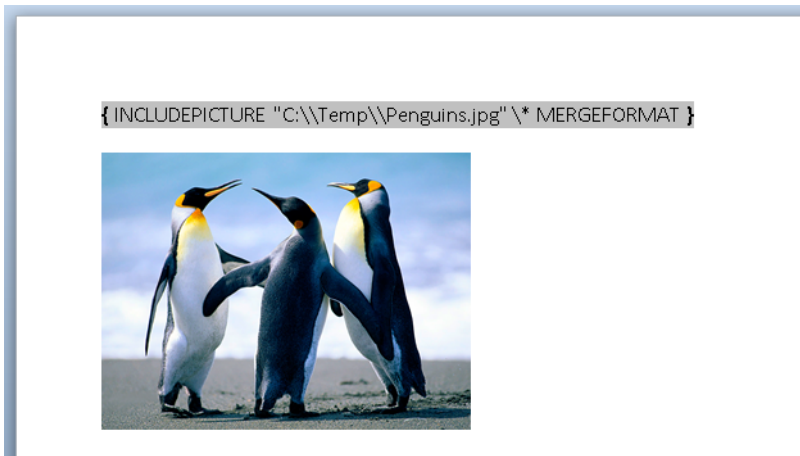
LinkFormat-Eigenschaft oder -Methode	Datentyp	Beschreibung
<code>BreakLink</code>		Löst die Verknüpfung auf
<code>SavePictureWithDocument</code>	Boolean	Wenn <code>False</code> , wird nur die Verknüpfung zur Grafik im Dokument gespeichert. (*)
<code>SourceFullName</code> <code>SourceName</code> <code>SourcePath</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 <code>True</code> , kann die Grafik nicht aktualisiert werden
(*) Fügt ab Word 2002 automatisch den Schalter <code>* MERGEFORMAT</code> hinzu, auch wenn er bereits vorhanden ist, was zu Formatierungsverlusten führen kann.		

Im Hintergrund verwaltet Word verknüpfte Grafiken über die Feldfunktion `IncludePicture` (in Abbildung 6.32 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 diejenige für `InlineShape`-Objekte sehen, da Shapes außerhalb des Textes in der Zeichnungsebene stehen.

Abbildg. 6.32 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, muss lediglich der Dateiname angegeben werden. Um auf die nächst höhere Ordnebene zu weisen, wird `..\\` vor den Pfadnamen gesetzt; mehrere `..\\` dürfen aufeinanderfolgen. 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 Ordner 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:

- Sie erlaubt die globale Bearbeitung der Verknüpfung direkt im Dokument. Eine Pfadangabe kann beispielsweise mit *Suchen und Ersetzen* geändert werden
- Nur so ist es möglich, den Schalter `* MERGEFORMAT` im Dokument vorhandenen 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 auch durch das Objektmodell direkt im Feldcode bearbeitet werden, um in Word 2000 ein 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 wiederhergestellt werden. Am Ende der Prozedur *GrafikAlsFeldEinfuegen*

wird die Verknüpfung aufgelöst, sodass 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:\\Beispiele\\Tulips.jpg"
Private Const strGRAFIKPFAD2 As String = "C:\\Beispiele\\Chrysanthemum.jpg"

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
```



Word 2010 benutzt diesen Feldcode nicht. Wird ein Dokument im alten Word97-2003-Dateiformat geöffnet und in das neue Dokumentformat gespeichert, werden die Feldcodes entfernt. Sie dürfen weiterhin die Feldfunktion einfügen und sie funktioniert – auch in .docx-Dokumenten. Allerdings

werden beim Öffnen eines OpenXML-Dokuments die Feldcodes und nicht deren Ergebnisse (die Grafiken) angezeigt, was wohl eine Sicherheitsmaßnahme ist.

Die Missbilligung (»deprecation«) des *IncludePicture*-Feldcodes hat für den Benutzer einige Nachteile, was die globale Verwaltung verknüpfter Grafiken angeht.

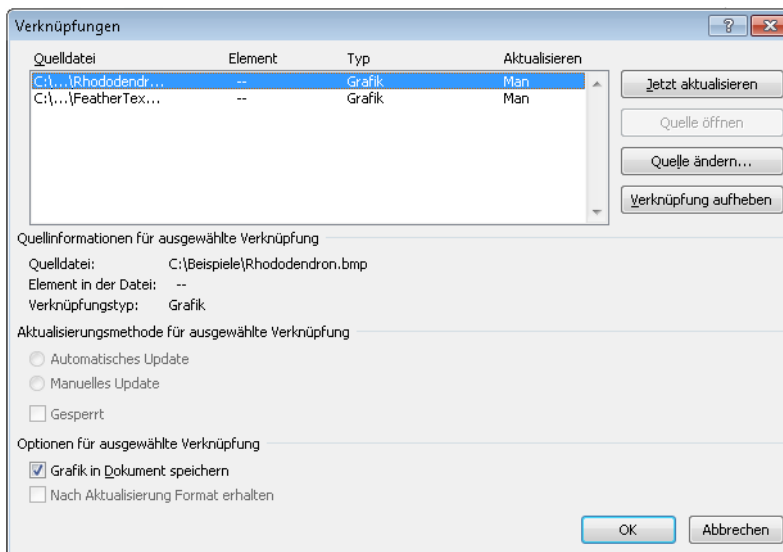
Das Dialogfeld für die Verwaltung verknüpfter Objekte befindet sich in Word 2007 unter der *Office*-Schaltfläche unter dem Befehl *Vorbereiten*, ganz unten im Untermenü: *Verknüpfungen mit Dateien bearbeiten* anwählen. In Word 2010 ist er ebenfalls nicht einfach zu finden: ganz unten, rechts in der Ansicht *Datei/Informationen*.

Obwohl damit Verknüpfungen verwaltet werden können, sieht der Benutzer nicht, auf welche Grafik ein Eintrag weist. Zudem sind die Informationsspalten schmal, nicht erweiterbar, und keine Laufleiste ist vorhanden (Abbildung 6.33). Enthält das Dokument viele verknüpfte Grafiken, oder sind die Dateinamen nicht beschreibend genug oder zu lange, gestaltet sich die Verwaltung der Verknüpfungen äußerst mühsam.

Der Benutzer kann zwar aus dem Kontextmenü den Befehl *Bild ändern* anwählen und eine andere Grafik anstelle der bestehenden einfügen. Er muss aber ausdrücklich einen Befehl wählen, der die Verknüpfung vornimmt, sonst wird die Grafikdatei ohne sie eingefügt – Word ersetzt eine verknüpfte Datei nicht automatisch mit einer erneuten Verknüpfung.

Leider steht die Information und Funktionalität für die Verwaltung von Verknüpfungen im neuen Aufgabenbereich *Auswahl und Sichtbarkeit* nicht zur Verfügung. Das wäre wohl eine ideale Lösung.

Abbildg. 6.33 Das Dialogfeld in der Benutzerschnittstelle für die Verwaltung von Grafiken



Es liegt dem Entwickler also nahe, seinen Benutzern ein Werkzeug für die Verwaltung von Verknüpfungen anzubieten. Das Listing 6.40 veranschaulicht eine Möglichkeit. Enthält die Markierung ein *InlineShape* oder ein *Shape* wird das Word-eigene Dialogfeld für die Grafikdateiauswahl eingeblendet (wird jedoch nicht ausgeführt). Diese Pfadangabe ersetzt den in der Eigenschaft *SourceFullName* gespeicherten Wert, alle anderen Eigenschaften der Grafik bleiben erhalten.

HINWEIS Mehr über die Fernsteuerung von Word-eigenen Dialogfeldern finden Sie im Kapitel 15 beschrieben.

Listing 6.40 Änderung der Verknüpfung einer markierten Grafik

```
Sub VerknüpfteGrafikErsetzen()
    Dim shp As Word.Shape
    Dim ils As Word.InlineShape
    Dim sel As Word.Selection

    Set sel = Selection
    Select Case sel.Type
        Case wdSelectionInlineShape
            Set ils = sel.InlineShapes(1)
            If ils.Type = wdInlineShapeLinkedPicture Then
                Debug.Print "Neuer Grafikpfad: " & GrafikErsetzen(ils)
            Else
                Debug.Print "Die Grafik ist nicht verknüpft."
            End If
        Case wdSelectionShape
            Set shp = sel.ShapeRange(1)
            If shp.Type = msoLinkedPicture Then
                Debug.Print "Neuer Grafikpfad: " & GrafikErsetzen(shp)
            Else
                Debug.Print "Die Grafik ist nicht verknüpft."
            End If
        Case Else
            MsgBox ("Markieren Sie bitte zuerst eine Grafik")
    End Select
End Sub

Private Function GrafikErsetzen(grafik As Object) As String
    Dim strPfadAngabe As String

    strPfadAngabe = GrafikdateiWählen
    'Die Verknüpfung nur ändern, wenn eine Datei ausgewählt wurde.
    If Len(strPfadAngabe) > 0 Then
        grafik.LinkFormat.SourceFullName = strPfadAngabe
    End If
    GrafikErsetzen = strPfadAngabe
End Function

Private Function GrafikdateiWählen() As String
    Dim dlg As Word.Dialog
    Dim strPfadAngabe As String

    Set dlg = Dialogs(wdDialogInsertPicture)
    With dlg
        .Display
        strPfadAngabe = .Name
    End With
    GrafikdateiWählen = strPfadAngabe
End Function
```

CD-ROM Die in diesem Abschnitt dargestellten Beispiele finden Sie in der Beispieldatei *Bsp06_02_Graf.doc* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap06*.

Layout-Optionen

Historisch gesehen sind InlineShape-Objekte Shape-Objekten vorzuziehen. Sie waren allgemein stabiler, in allen Ansichten sichtbar sowie ansprechbar und einfacher zu verwalten. Mit Einführung der neuen Office-Grafikschnittstelle fallen die meisten Nachteile des Shape-Objekts weg:

- Im neuen Dateiformat sind weder Shapes noch InlineShapes in der Entwurfs- oder Gliederungsansicht sichtbar
- Über den Aufgabenbereich *Auswahl und Sichtbarkeit* kann der Benutzer Shapes ausblenden (InlineShapes müssen weiterhin mit der Zeichenformatierung »Ausgeblendet« versehen sein, um sie unsichtbar zu machen.)
- Seit Word 2007 berücksichtigt Word in Abbildungsverzeichnissen und Querverweisen die in AutoFormen eingefügten Beschriftungen
- Seit Einführung des neuen Dokumentformats sind Word-Dokumente allgemein stabiler und weniger korruptionsanfällig

HINWEIS Falls Sie mit dem alten Dateiformat arbeiten, finden Sie die Informationen zum Grafik-Layout auf der CD-ROM zum Buch im Ordner *\Beilagen\Kap06_Grafiken*.

Es gibt jedoch noch Aufgaben, wofür ein InlineShape besser geeignet ist.

Beschriftungen

Beschriftungen in Word werden automatisch entweder über oder unter der Grafik eingefügt. Wissenschaftliche Arbeiten verlangen manchmal deren Positionierung rechts, neben dem Objekt. In diesem Fall ist es vorteilhaft, die Grafik in die linke Zelle einer einzeiligen Tabelle einzufügen und die Beschriftung in die rechte.

Eine Tabelle ist auch sonst als Behälter für Grafik und Abbildung erwägenswert, da beide zusammen verschoben werden können. (Tabellen können auch mit Textflussformatierung versehen werden.)

Eine weitere, oft benutzte Möglichkeit, um eine Grafik mit ihrer Beschriftung zu verbinden, ist, die Grafik als InlineShape in einen Positionsrahmen einzufügen.

HINWEIS Positionsrahmen sind seit längerem »missbilligt« und nicht mehr einfach zu verwalten. Deshalb soll deren Einsatz immer eingehend geprüft und Alternativen in Erwägung gezogen werden. Positionsrahmen haben jedoch noch bestimmte Vorteile. Beispielsweise können sie als fester Bestandteil einer Formatvorlage definiert werden, was die automatische, freie Positionierung von Text auf der Seite ermöglicht (als Marginalien).

Ein weiterer Vorteil des Einfügens einer Grafik als InlineShape in eine Tabelle oder einen Positionsrahmen ist, wenn diese mit fester Breite oder Höhe formatiert sind, passt sich die Grafik dieser Dimension proportional an und die Größe muss nicht weiter bearbeitet werden. Beispielcode hierfür finden Sie in Listing 6.41.

Listing 6.41 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:\Beispiele\Kap06\Tulips.jpg", _
        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

```

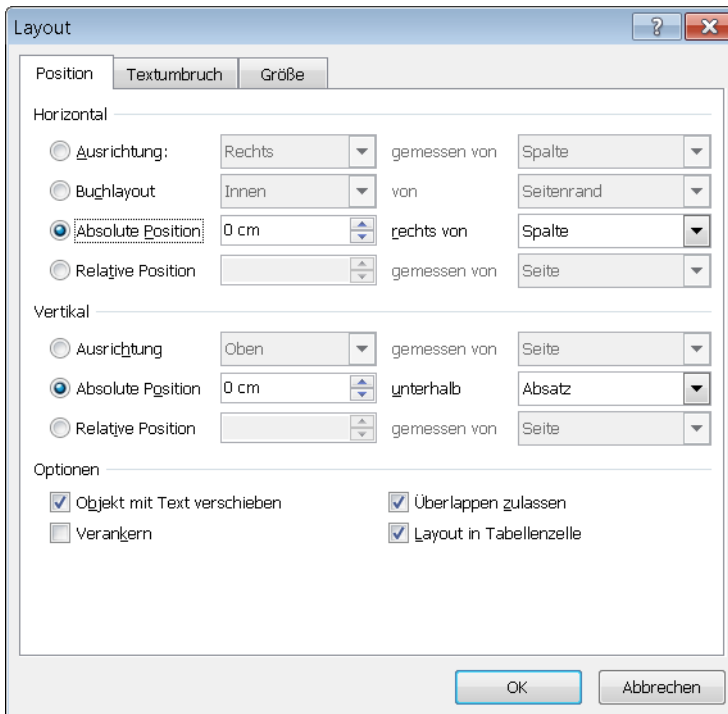
CD-ROM Die in diesem Abschnitt dargestellten Beispiele finden Sie in der Beispieldatei *Bsp06_03_Graf.doc* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap06*.

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 auf der Registerkarte *Bildtools/Format*. Im Dropdownmenü der Schaltflächen *Position* oder *Textumbruch* wählen Sie den Eintrag *Weitere Layoutoptionen*. Im Dialogfeld wechseln Sie anschließend zur Registerkarte *Position* (Abbildung 6.34). (Für Positionsrahmen geht's über die Befehlsfolge unter *Positionsrahmen formatieren* des Kontextmenüs. 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. 6.34 Die Optionen, die die Positionierung von *Shape*-Objekten regeln



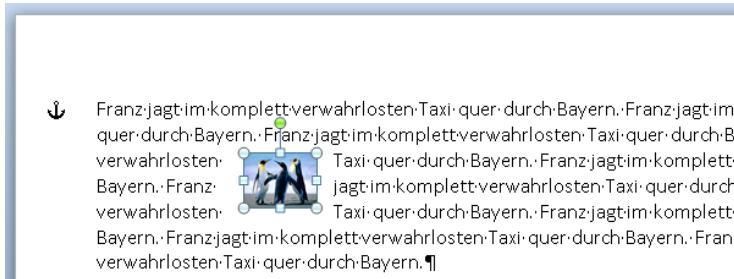
Die erste und wichtigste Regel für die Positionierung ist, dass ein grafisches Objekt (Shape oder Positionsrahmen (Frame)) *immer* mit einem Absatz verankert ist, und sich *immer* auf der gleichen Seite befindet wie dieser Absatz. In Abbildung 6.35 sehen Sie einen Objektanker. Falls das entsprechende Kontrollkästchen in der Kategorie *Anzeigen* der Word-Optionen aktiviert ist, soll er sichtbar sein, wenn sein grafisches Objekt markiert ist.

Der Anker kann mit der Maus verschoben werden, eine programmtechnische Schnittstelle dafür gibt es jedoch nicht. Der verankernde Bereich kann lediglich beim Einfügen eines Shape-Objekts über die *AddPicture*-Methode bestimmt werden (mit Ausnahme des Fehlverhaltens in Word 2010, wie im Abschnitt »Aus Datei mit *AddPicture*« ab Seite 309 erwähnt). Zwar hat das Shape-Objekt eine *Anchor*-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. 6.35 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, sodass 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, sodass 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 *Position*. 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) im Zusammenspiel mit den Top- (oben) bzw. Left- (links) Eigenschaften für die Festlegung des Layouts. Die möglichen Werte der ersten beiden Eigenschaften sind in Tabelle 6.12 sowie in Tabelle 6.13 zusammengefasst.

Tabelle 6.12 Werte für die Eigenschaft *RelativeVerticalPosition*

Enumeration	Wert	Beschreibung
wdRelativeVerticalPositionLine	3	Der Abstand wird relativ zur verankernde Zeile gemessen (mit Text verschieben)
wdRelativeVerticalPositionMargin	0	Der Abstand wird vom oberen Textrand gemessen
wdRelativeVerticalPositionPage	1	Der Abstand wird vom oberen Seitenrand gemessen
wdRelativeVerticalPositionParagraph	2	Der Abstand wird relativ zum verankernden Absatz gemessen (mit Text verschieben)

Tabelle 6.13 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 6.14.

Tabelle 6.14 Werte für die 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 Buchs ausgerichtet (<i>Gerade/ungerade anders</i> in der Menüfolge <i>Datei/Seiteneinrichten/Layout</i>)
wdShapeLeft	–999998	links
wdShapeOutside	–999993	Die Grafik wird am Außenrand eines Buchs ausgerichtet (<i>Gerade/ungerade anders</i> in der Menüfolge <i>Datei/Seiteneinrichten/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 6.42 vorgegangen.

Listing 6.42 Ein grafisches Objekt relativ zur Seite positionieren

```
Sub GrafikGenauAufDerSeitePositionieren()
    Dim shp As Word.Shape

    'Zweites InlineShape im Dokument in ein Shape konvertieren
```

Listing 6.42 Ein grafisches Objekt relativ zur Seite positionieren (*Fortsetzung*)

```
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
```

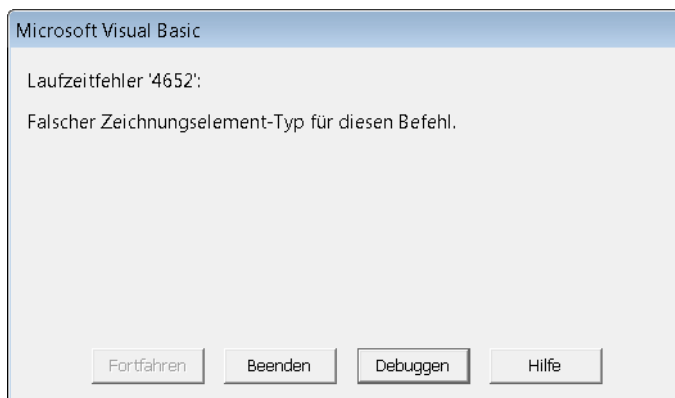
CD-ROM Die in diesem Abschnitt dargestellten Beispiele finden Sie in der Beispieldatei *Bsp06_03_Graf.doc* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap06*.

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 dem *Application*-Objekt an und sind ausschließlich im Word- und keinem anderen Office-Objektmodell enthalten.

Neue Positionierungs- und Vergrößerungsmöglichkeiten

Zusätzlich zu den oben beschriebenen Positionierungsmöglichkeiten wurden für die neuen Office 2007-Grafiken weitere eingeführt. Diese stehen in Word 2007 hauptsächlich für AutoFormen zur Verfügung. In Word 2010 können sie allen Grafiken eines OpenXML-Dokuments zugewiesen werden. Für Dokumente im alten Dateiformat bleiben sie gesperrt. Der Laufzeitfehler in Abbildung 6.36 erscheint, wenn die neuen Positionierungsbefehle mit einem Shape-Objekt des falschen Typs verwendet werden.

Abbildg. 6.36 Fehlermeldung, wenn ein *Shape*-Objekt mit den neuen Grafikeigenschaften nicht kompatibel ist


Die Tabelle 6.15 bis Tabelle 6.17 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 6.43.

Tabelle 6.15 Zusätzliche Werte für die Eigenschaft *RelativeVerticalPosition* ab 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 6.16 Zusätzliche Werte für die Eigenschaft *RelativeHorizontalPosition* ab 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 6.17 Werte für die Eigenschaften *RelativeHorizontalSize* sowie *RelativeVerticalSize* ab 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
<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

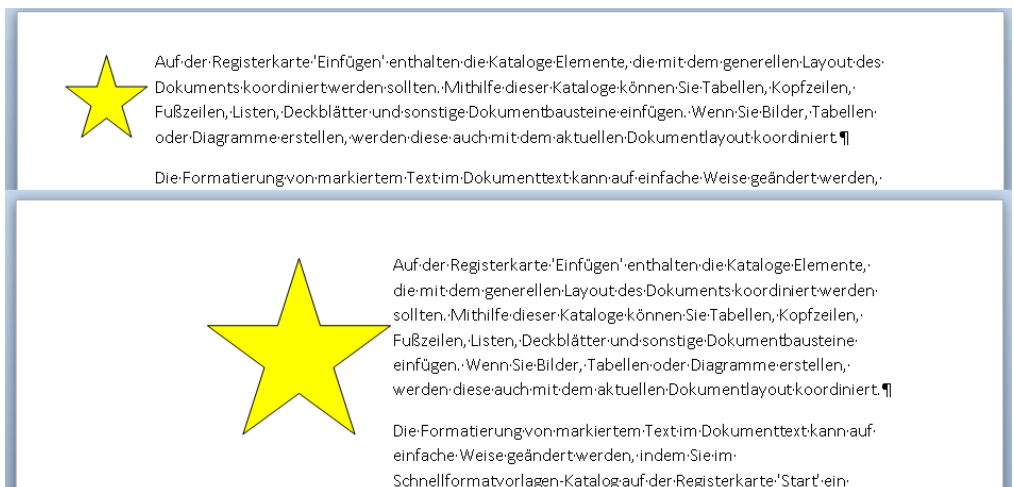
Tabelle 6.17 Werte für die Eigenschaften *RelativeHorizontalSize* sowie *RelativeVerticalSize* ab Office 2007 (Fortsetzung)

Enumeration	Wert	Beschreibung
<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 6.37 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 auf 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 6.37 zu erzielen, wurde die Prozedur nochmals durchgeführt.

Abbildg. 6.37 Die relative Positionierung sowie Größenänderung einer AutoForm in Word 2007/2010



Listing 6.43 Eine AutoForm mit den neuen Möglichkeiten in Word 2007/2010 formatieren

```

Sub FormRelativ()
    Dim shp As Word.Shape

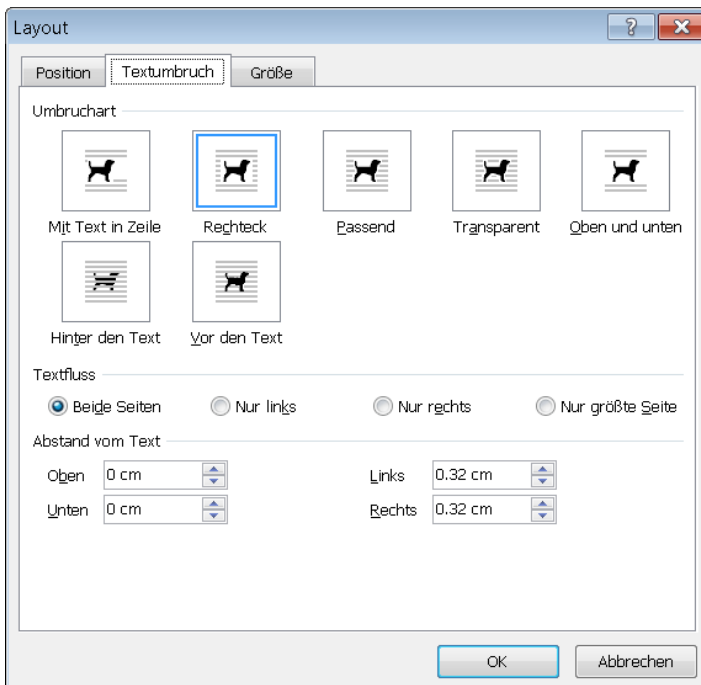
    'Die AutoForm wurde vorgängig explizit benannt
    Set shp = ActiveDocument.Shapes("Stern")
    shp.LockAspectRatio = msoFalse
    shp.RelativeHorizontalSize = wdRelativeHorizontalSizeLeftMarginArea
    shp.WidthRelative = 50
    shp.Height = shp.Width
    shp.RelativeHorizontalPosition = wdRelativeHorizontalPositionLeftMarginArea
    shp.LeftRelative = 50
    shp.RelativeVerticalPosition = wdRelativeVerticalPositionMargin
    shp.Top = 0
    shp.LockAspectRatio = msoTrue
End Sub

```

CD-ROM Die in diesem Abschnitt dargestellten Beispiele finden Sie in der Beispieldatei *Bsp06_06_Graf.docm* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap06*.

Textfluss-Formatierung

Neben der Registerkarte *Bildposition* enthält das Dialogfeld *Erweitertes Layout* die Registerkarte *Textfluss* (Abbildung 6.38), die einige zusätzliche Optionen zur Registerkarte *Layout* im Dialogfeld *Grafik formatieren* anbietet.

Abbildg. 6.38 Der genaue Textfluss wird im Dialogfeld *Layout* festgelegt

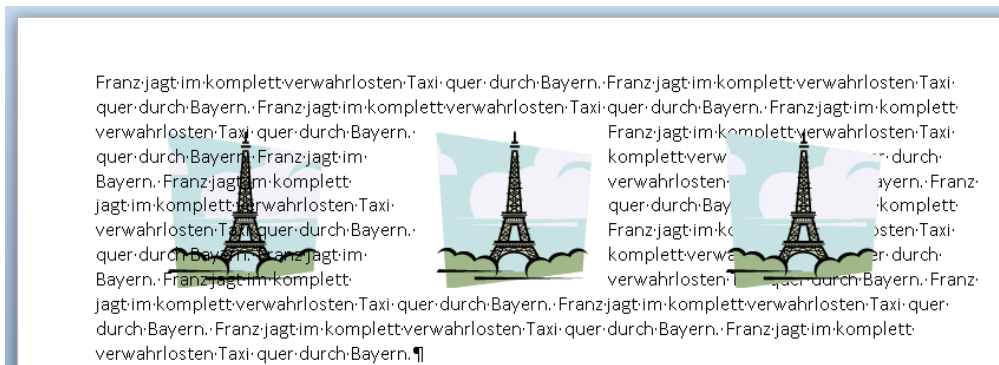
Textfluss Im Objektmodell umfasst die Eigenschaft `WrapFormat` die Optionen in den Abschnitten *Textfluss* und *Abstand vom Text*. Auch die meisten Umbrucharten sind dieser Eigenschaft zugeteilt, außer *Hinter den Text* und *Vor den Text*, die unter die Methode `ZOrder` fallen. Die `WrapFormat`-Eigenschaften mit ihren Werten und Wirkungen sind in Tabelle 6.18 aufgelistet.

Tabelle 6.18 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 und gehört weiterhin der <code>Shapes</code> -Auflistung an.
	<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> , alleinstehend in einem Absatz, dieser Einstellung vorzuziehen.
Side	<code>wdWrapBoth</code>	0	<i>Textfluss/Beide Seiten</i>
	<code>wdWrapLargest</code>	3	<i>Textfluss/Nur größte Seite</i>
	<code>wdWrapLeft</code>	1	<i>Textfluss/Nur links</i>
	<code>wdWrapRight</code>	2	<i>Textfluss/Nur rechts</i>
<code>DistanceBottom</code>			<i>Abstand vom Text/Unten</i>
<code>DistanceLeft</code>			<i>Abstand vom Text/Links</i>
<code>DistanceRight</code>			<i>Abstand vom Text/Rechts</i>
<code>DistanceTop</code>			<i>Abstand vom Text/Oben</i>

Wenn Sie das grafische Layout eines Word-Dokuments betrachten, sehen Sie, dass es aus drei dimensional »Ebenen« besteht: aus der Textebene, der Ebene dahinter und der Ebene davor. Diese sind in der Abbildung 6.39 klar ersichtlich. Von links nach rechts: hinter dem Text, die Textebene und vor dem Text.

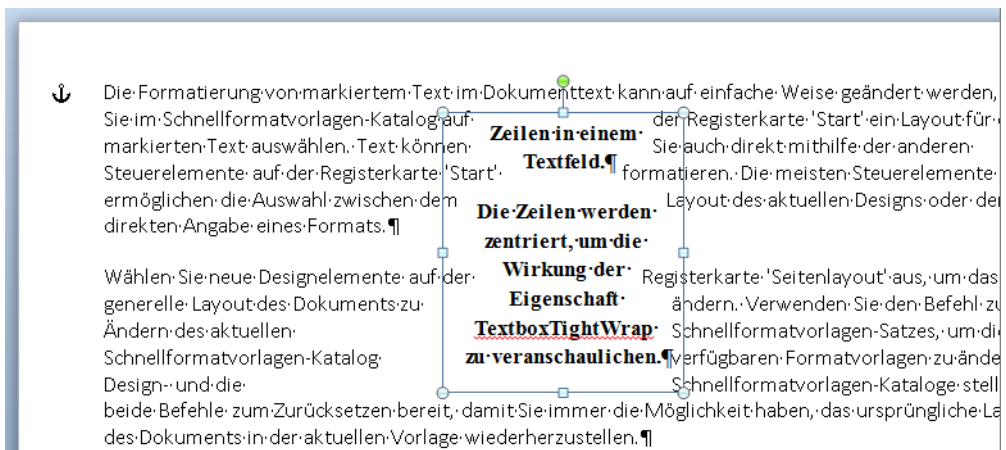
Abbildg. 6.39 Die drei grafischen Ebenen eines Word-Dokuments

**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.

Eine neue Art von Umbruch wurde in Word 2007 eingeführt: *TextboxTightWrap*. Es ermöglicht den Umbruch von Text *um den Text* in einem Textfeld, wie in Abbildung 6.40.

Abbildg. 6.40 Der Text im Dokument fließt um den Text im Textfeld, statt um den (unsichtbaren) Rahmen

Tabelle 6.19 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 6.44 zeigt, wie das Textfeld in Abbildung 6.40 erstellt wurde.

Listing 6.44 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

    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
```

CD-ROM Die in diesem Abschnitt dargestellten Beispiele finden Sie in der Beispieldatei *Bsp06_07_Graf.docm* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap06*.

Reihen-
folge

Zudem können in allen drei Ebenen Grafiken übereinander liegen. Diese Reihenfolge wird in der Benutzerschnittstelle über die Schaltflächen *Ebene nach vorne* bzw. *Ebene nach hinten* der Gruppe *Anordnen* der Registerkarte *Zeichentools/Format* festgelegt. Im Objektmodell entspricht diesem Befehl die Methode *ZOrder*, deren Werte in Tabelle 6.20 aufgelistet sind.

Tabelle 6.20 Die Werte der Methode *ZOrder*

Enumeration	Menübefehl
<code>msoBringForward</code>	<i>Ebene nach vorne</i>
<code>msoBringInFrontOfText</code>	<i>Vor dem Text platzieren</i>
<code>msoBringToFront</code>	<i>In den Vordergrund</i>
<code>msoSendBackward</code>	<i>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 zu einander 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.

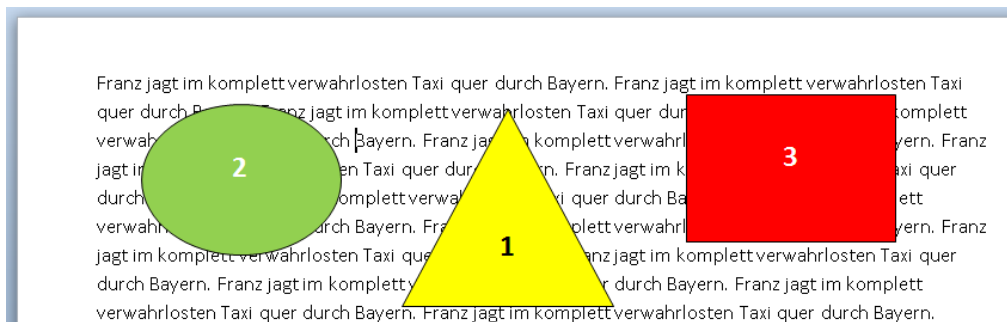
In Abbildung 6.41 sehen Sie drei AutoFormen (alle in der gleichen Ebene vor dem Text stehend), die in der Reihenfolge nummeriert sind, wie sie in das Dokument eingefügt wurden. Die Prozedur *GrafikenAusrichtenUndEinordnen* in Listing 6.45 richtet sie zuerst waagrecht sowie senkrecht zentriert auf der Seite (also übereinander) aus. Wie in Abbildung 6.42 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, sodass am Schluss die Nummer 1 wie in Abbildung 6.43 vorn erscheint.

Wenn anschließend das gelbe 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 die ZOrderPosition angeht, rein optisch.

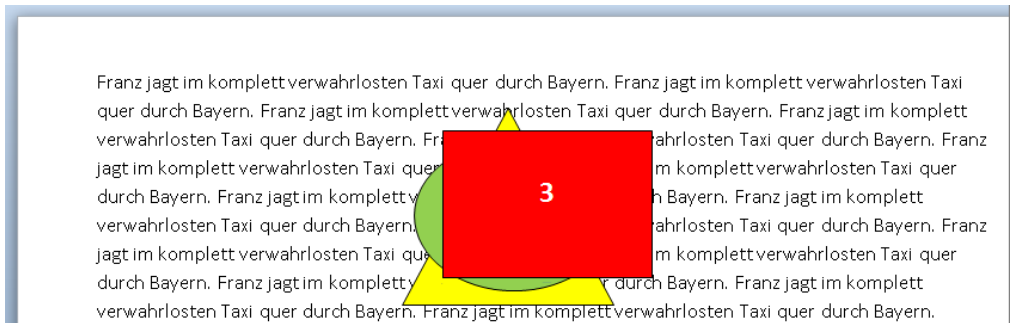
HINWEIS

Bitte beachten Sie, wenn Sie mit einer For Each...Next-Schleife alle Shape-Objekte in einem Dokument oder Bereich durchlaufen, dass diese in der Reihenfolge ihrer Verankerung geschieht. Mit dieser Reihenfolge hat die ZOrderPosition nichts zu tun.

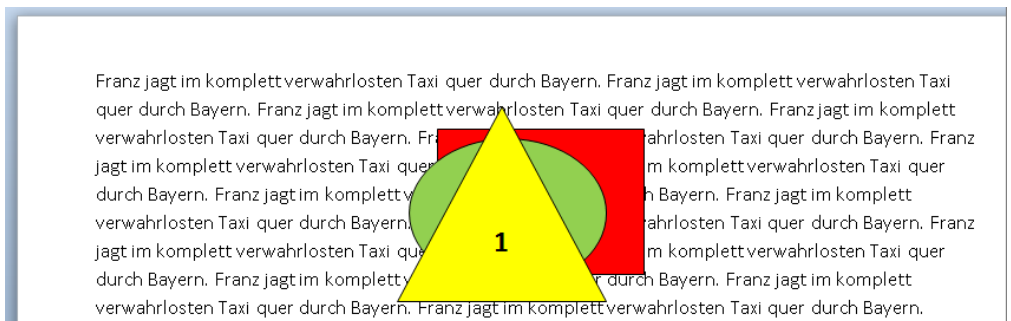
Abbildg. 6.41 Die Grafiken sind in der Reihenfolge nummeriert, in welcher sie eingefügt wurden



Abbildg. 6.42 Zieht man die Grafiken übereinander, liegt die zuletzt eingefügte zuoberst



Abbildg. 6.43 Die Prozedur in Listing 6.45 hat die *ZOrder* umgekehrt



Listing 6.45 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.  
    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
```

Listing 6.45 AutoFormen zentriert ausrichten und hintereinander anordnen (Fortsetzung)

```

'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

```

CD-ROM

Die in diesem Abschnitt dargestellten Beispiele finden Sie in der Beispieldatei *Bsp06_04_Graf.doc* 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 Ihnen eine allgemeine Vorstellung davon vermitteln sollen, wie mit VBA AutoFormen in einem Word-Dokument erstellt werden.

HINWEIS

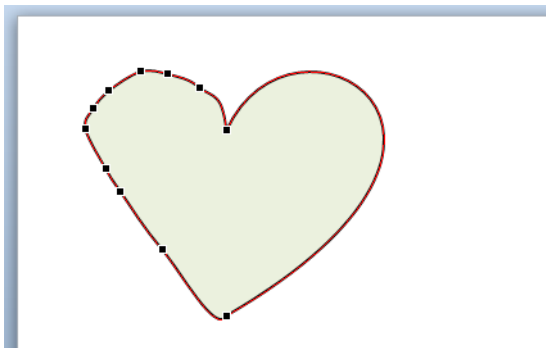
Angaben zu WordArt, das ein Objektmodell besitzt, finden Sie in Kapitel 12.

Freihandformen bearbeiten



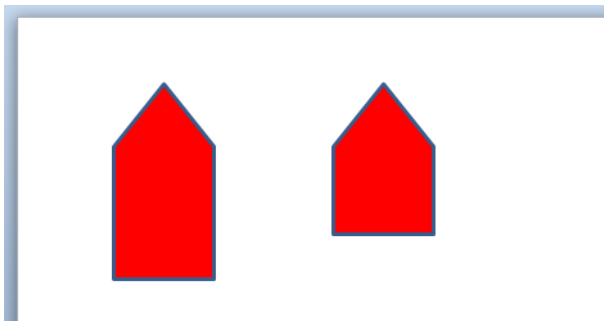
Etwas, das immer wieder für Unklarheit sorgt, ist die *Points*-Eigenschaft der *ShapeNodes*-Auflistung. Ein *ShapeNode* ist entweder ein Eckpunkt oder bei Kurven ein Punkt, der die Rundung bestimmt. Durch Änderung der Koordinaten (*Points*) wird das Aussehen einer *von Hand gezeichneten* Auto-Form geändert. In der Benutzerschnittstelle werden sie über die Schaltfläche *Form bearbeiten* der Gruppe *Formen einfügen* der Registerkarte *Zeichentools/Format* aktiviert und sehen wie in Abbildung 6.44 aus.

Abbildg. 6.44 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 Hilfeangaben zur Eigenschaft. Obwohl die Objekthierarchie *Shape.Nodes.Item.Points* lautet, erscheint eine Fehlermeldung (»die Typen sind unverträglich«), wenn eine als *Shape* deklarierte Objektvariable eingesetzt wird. Um das Problem zu umgehen, muss die Objektvariable als allgemeiner Typ *Object* deklariert werden. Das Listing 6.46 veranschaulicht das Problem und die Lösung. Das Makro erstellt ein Fünfeck als Vieleck (*Polygon*), dann werden die zwei senkrechten Seiten verkürzt, wie in Abbildung 6.45 ersichtlich.

Abbildg. 6.45 Ein Fünfeck mit VBA erstellen und bearbeiten



Zuerst wird eine Objektvariable – shp – für die AutoForm als Shape deklariert. In der Datenfeldvariable 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 Objektvariable 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 gleichgesetzt.

Noch ein Datenfeld wird gebraucht, um die Koordinaten des Punktes festzuhalten, den wir verschieben möchten. Auch hier muss eine neue Objektvariable her, da Points nur einer Objektvariablen 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 Punkt (typografisches Maß) verkürzt (höher gestellt).

Listing 6.46

Ein Polygon erstellen und anschließend die Eckpunkte ändern

```
Sub EinFünfeckZeichnen()
    Dim shp As Word.Shape
    Dim aEckPunkte(1 To 6, 1 To 2) As Single
    Dim vw As Word.View
    Dim bCurVw As Boolean

    Set vw = ActiveDocument.ActiveWindow.View
    'In Word 2002 können Grafiken falsch positioniert werden,
    'wenn die oberen und unteren Seitenränder ausgeblendet sind.
    'Die Benutzereinstellung festhalten und am Schluss wieder herstellen
    'WORD 2000: die beiden folgenden Zeilen entfernen!
    bCurVw = vw.DisplayPageBoundaries
    If bCurVw = False Then vw.DisplayPageBoundaries = True

    aEckPunkte(1, 1) = 25
    aEckPunkte(1, 2) = 25
    aEckPunkte(2, 1) = 50
    aEckPunkte(2, 2) = 50
    aEckPunkte(3, 1) = 50
    aEckPunkte(3, 2) = 100
    aEckPunkte(4, 1) = 0
    aEckPunkte(4, 2) = 100
    aEckPunkte(5, 1) = 0
    aEckPunkte(5, 2) = 50
    aEckPunkte(6, 1) = 25
    aEckPunkte(6, 2) = 25
```

Listing 6.46 Ein Polygon erstellen und anschließend die Eckpunkte ändern (Fortsetzung)

```

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 Formatierungen zuzuweisen. Neue AutoFormen werden immer mit der Formatvorlage *Standard* formatiert. Während es möglich ist, AutoFormen zu kopieren, stellen wir zunehmend Probleme in gewissen Word-Versionen 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 6.47 finden Sie Beispielcode, der alle Textfelder (und ausschließlich diese) mit Textinhalt im Dokumenttext gleich formatiert.

Listing 6.47 Den Text in allen Textfeldern des Dokumentkörpers mit Arial 10, fett, formatieren

```

Sub AlleTextBereicheFormatieren()
    Dim shp As Word.Shape

    For Each shp In ActiveDocument.Shapes
        If shp.Type = msoTextBox Then
            With shp.TextFrame
                If .HasText Then
                    .TextRange.Font.Name = "Arial"
                    .TextRange.Font.Size = 10
                End If
            End With
        End If
    Next shp
End Sub

```

Listing 6.47 Den Text in allen Textfeldern des Dokumentkörpers mit Arial 10, fett, formatieren (Fortsetzung)

```
.TextRange.Bold = True
End If
End With
End If
Next shp
End Sub
```

CD-ROM Die in diesem Abschnitt dargestellten Beispiele finden Sie 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 basierend 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 266 ff.)
- In den weiteren Abschnitten standen Formatvorlagen (Seite 282), die Nummerierungen (Seite 297 ff.) und die Grafik-Objekte (Seite 308 ff.) im Vordergrund des Interesses

Basierend auf der Erfahrung der Autoren in den Newsgruppen und Foren 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	342
Inhalt mit Tabellen strukturiert darstellen	351
Feldfunktionen	380
Formulare: das <i>FormField</i> -Objekt	393
Die Alternative zu Formularfeldern: das <i>ContentControls</i> -Objekt	402
Der Seriendruck: das <i>MailMerge</i> -Objekt	433
Zusammenfassung	452


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 die Ausgabe von Daten einsetzen. Dieses Kapitel widmet sich den Word-eigenen Werkzeugen, um dieser Aufgabe gerecht zu werden.

Zielscheibe Textmarke: das *Bookmark*-Objekt

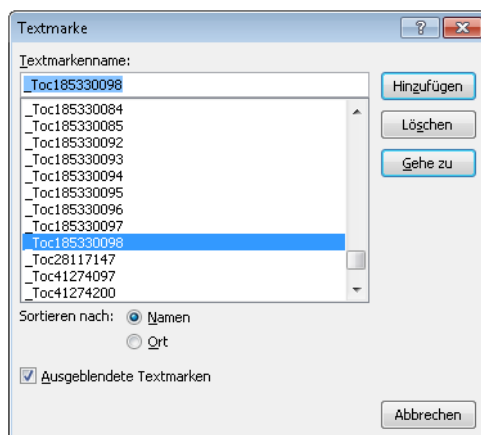
Textmarken in Word-Dokumenten erfüllen zwei wichtige Aufgaben. Dem Anwender dienen sie hauptsächlich als Quellenbezeichner für Verweise. Der Entwickler benutzt sie, um Zielbereiche zu bezeichnen.

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« ab Seite 380). 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 (die Schaltfläche *Textmarke* befindet sich 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, sodass 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.

Abbildg. 7.2 Die zwei Arten von Textmarken

*Falls die entsprechende Option im Abschnitt *Dokumentinhalt anzeigen* von Datei/Optionen/Erweitert aktiviert ist, sehen Sie anfangs dieses Absatzes eine Textmarke, die eine Position markiert.¶*
In diesem Absatz ist das dritte Wort mit einer Textmarke versehen.¶

TIPP

Um die Textmarken im Word-Dokument zu sehen, muss die entsprechende Option aktiviert sein. Sie befindet sich in der Kategorie *Erweitert* der (*Word*-) *Optionen* im Abschnitt *Dokumentinhalt anzeigen*.

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 Schaltfläche in der Schnellzugriffsleiste 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#.NET): Die C#-Version von Listing 7.1



```

private void TextmarkeEinfuegen_CS()
{
    string text = "";
    string markierungTyp = "";
    object missing = System.Type.Missing;
    if (wdApp.Documents.Count < 1)
    {
        wdApp.Documents.Add(ref missing, ref missing, ref missing, ref missing);
    }
    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="";
}

```

CD-ROM Die Beispieldatei *Bsp07_01_Bookmark.docm* 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.

[TM1-mit-Inhalt].[TM2-als-Position].[TM3-mit-Inhalt]
 [NEUER-TEXT][TM1-mit-Inhalt].[NEUER-TEXT][TM2-als-Position].
 [NEUER-TEXT].

Listing 7.3 Eine Textmarke nach Einfügen von Daten wieder erstellen, sodass 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 (.NET): Die C#-Version von Listing 7.3



```
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.docm");
    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.
```

Listing 7.4 (.NET): Die C#-Version von Listing 7.3 (Fortsetzung)

```

object objBookmarkName = textmarkenName;
wd.Bookmark bkm = doc.Bookmarks.get_Item(ref objBookmarkName);
bkm.Range.InsertBefore(text);

textmarkenName = "TM2";
objBookmarkName = textmarkenName;
bkm = doc.Bookmarks.get_Item(ref objBookmarkName);
//Neuen Text an die Position der Textmarke einfügen.
bkm.Range.Text = text;

bkm=null;
textmarkenName = "TM3";
objBookmarkName = textmarkenName;
//Den gegenwärtigen Inhalt mit neuem Text ersetzen und Textmarke beibehalten.
//Prüfen, ob die Textmarke vorhanden ist
if (doc.Bookmarks.Exists(textmarkenName))
{
    bkm = doc.Bookmarks.get_Item(ref objBookmarkName);
    wd.Range rng = bkm.Range;
    rng.Text = text;
    //Die Textmarke ist weg und muss ersetzt werden
    object objRange = rng;
    bkm = doc.Bookmarks.Add(textmarkenName, ref objRange);
}
}

```

CD-ROM Die Beispieldatei *Bsp07_01_Bookmark.docm* 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 durchgeschleift 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
\Sel	Gegenwärtige Markierung	Selection.Range
\PrevSel1	Die zuletzt bearbeitete Stelle	Application.GoBack
\PrevSel2	Die vorletzte bearbeitete Stelle	Application.GoBack zweimal
\StartOfSel	Startpunkt der gegenwärtigen Markierung	Range.Start
\EndOfSel	Endpunkt der gegenwärtigen Markierung	Range.End
\Line	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)
\Char	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	Range.Characters(1)
\Para	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 \Para die Absatzmarke nicht ein.	Range.Paragraphs(1)
\Section	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 \Section der erste Abschnitt in der Markierung.	Range.Sections(1)
\Doc	Gesamter Inhalt des aktiven Dokuments, mit Ausnahme der letzten Absatzmarke	ActiveDocument
\Page	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 \Page die erste Seite der Markierung. Befindet sich die Einfügemarke oder Markierung auf der letzten Seite des Dokuments, enthält die Textmarke \Page nicht die letzte Absatzmarke.	(keine gleichwertige Anweisung)
\StartOfDoc	Der Anfang des Dokuments	ActiveDocument.Content.Start

Tabelle 7.1 Liste der vordefinierten Textmarken, mit gleichwertiger Anweisung des Objektmodells, wo vorhanden (*Fortsetzung*)

Vordefinierte Textmarke	Beschreibung	Objektmodell-Anweisung
\EndOfDoc	Das Ende des Dokuments	ActiveDocument.Content.End
\Cell	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 \Cell die erste Zelle in der Markierung.	Range.Cells(1)
\Table	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 \Table die gesamte erste Tabelle der Markierung. Dies ist auch der Fall, wenn nicht die gesamte Tabelle markiert ist.	Range.Tables(1)
\HeadingLevel	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 \HeadingLevel 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*- anstatt 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
```

Listing 7.5 Da mit vordefinierten Textmarken gearbeitet wird, wird ausnahmsweise mit dem *Selection*- anstatt dem *Range*-Objekt gesucht (*Fortsetzung*)

```
Application.ScreenUpdating = false
With Application.Selection.Find
    .ClearAllFuzzyOptions
    .ClearFormatting
    .MatchAllWordForms = False
    .MatchCase = False
    .MatchSoundsLike = False
    .MatchWholeWord = False
    .MatchWildcards = False
    .Format = True
    .Forward = True
    .Text = ""
    .Wrap = wdFindStop
    .Style = wdStyleHeading3
End With
Do
    bFound = Selection.Find.Execute
    If bFound Then
        rngZielBereich.FormattedText = _
            Selection.Bookmarks("\HeadingLevel").Range.FormattedText
        rngZielBereich.Collapse Direction:=wdCollapseEnd
        Selection.Collapse Direction:=wdCollapseEnd
    End If
    Loop While bFound
End Sub
```

CD-ROM Die Beispieldatei *Bsp07_01_Bookmark.docm* 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, sodass 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, sodass Dokumente, die in einer früheren Version geöffnet werden, anders aussehen könnten. Bei Automatisierungscode kommt es gar zu Kompilierungsfehlern, die die Fehlerbehandlung nicht abfangen kann.

Es ist also sehr wichtig, Anwendungen, die für mehrere Word-Versionen vorgesehen sind, in der ältesten dieser Versionen zu entwickeln und gründlich zu testen.

PROFITIPP

Für den Entwickler, der VBA-Code in einem Word-Projekt schreibt, bietet sich die Möglichkeit, versionspezifischen Code in ein eigenes Modul zu schreiben. Da VBA den Code eines Moduls erst kompiliert, wenn erstmals eine darin stehende Prozedur aufgerufen wird, ist dies ein probates Mittel, um Unterschiede im Objektmodell in einer Anwendung zu handhaben.

Entwickler, die Word von einer anderen Umgebung aus automatisieren, müssen mit »Late Binding« arbeiten.

In beiden Fällen wird die Eigenschaft `Application.Version` abgefragt, um herauszufinden, welche Word-Version vorliegt.

Tabellen erstellen

Add

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.

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.

Convert-
TextTo-
Table

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
```

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)

```

Dim lAnzSpalten As Long
Dim lZaehlerZeile As Long
Dim lZaehlerSpalte As Long

Set doc = Documents.Add
Set rng = doc.Content
lAnzZeilen = lDIM_1 + 1
lAnzSpalten = lDIM_2 + 1

Dim aDaten(lDIM_1, lDIM_2) As String
DatenZuweisen aDaten()

'Zeilen nach Bedarf erstellen und Daten einfügen
Set tbl = doc.Tables.Add(rng, 1, lAnzSpalten, wdWord8TableBehavior)
Set row = tbl.Rows(1)
For lZaehlerZeile = LBound(aDaten, 1) To UBound(aDaten, 1)
    'Daten in die letzte Zeile eingeben.
    For lZaehlerSpalte = LBound(aDaten, 2) To UBound(aDaten, 2)
        row.Cells(lZaehlerSpalte + 1).Range.Text = _
            aDaten(lZaehlerZeile, lZaehlerSpalte)
    Next
    'Neue Zeile nach Bedarf einfügen.
    If row.Index <> lAnzZeilen Then
        Set row = Nothing
        Set row = tbl.Rows.Add
    End If
Next

Set rng = BereichNachTabelleFestlegen(tbl)

'Alle Zeilen erstellen, dann die Daten einfügen.
Set tbl = Nothing
Set tbl = doc.Tables.Add(rng, lAnzZeilen, lAnzSpalten, wdWord8TableBehavior)
For lZaehlerZeile = LBound(aDaten, 1) To UBound(aDaten, 1)
    For lZaehlerSpalte = LBound(aDaten, 2) To UBound(aDaten, 2)
        tbl.Cell(lZaehlerZeile + 1, lZaehlerSpalte + 1).Range.Text = _
            aDaten(lZaehlerZeile, lZaehlerSpalte)
    Next
Next

Set rng = BereichNachTabelleFestlegen(tbl)

'Aus den Daten eine zeichengetrennte Zeichenkette erstellen
'Diese einfügen und in eine Tabelle umwandeln.
Dim strDaten As String
For lZaehlerZeile = LBound(aDaten, 1) To UBound(aDaten, 1)
    For lZaehlerSpalte = LBound(aDaten, 2) To UBound(aDaten, 2)
        strDaten = strDaten & aDaten(lZaehlerZeile, lZaehlerSpalte)
        If lZaehlerSpalte <> UBound(aDaten, 2) Then
            strDaten = strDaten & vbTab
        End If
    Next
    If lZaehlerZeile <> UBound(aDaten, 1) Then
        strDaten = strDaten & vbCr
    End If
Next

```

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)

```

    End If
    Next
    rng.Text = strDaten
    Set tbl = rng.ConvertToTable(vbTab)
End Sub

Private Sub DatenZuweisen(ByRef aDaten() As String)
    Dim lZaehlerZeilen As Long
    Dim lZaehlerSpalten

    For lZaehlerZeilen = LBound(aDaten, 1) To UBound(aDaten, 1)
        For lZaehlerSpalten = LBound(aDaten, 2) To UBound(aDaten, 2)
            aDaten(lZaehlerZeilen, lZaehlerSpalten) = _
                "Zeile " & CStr(lZaehlerZeilen + 1) & _
                ", Spalte " & CStr(lZaehlerSpalten + 1)
        Next
    Next
End Sub

Public Function BereichNachTabelleFestlegen(tbl As Word.Table) As Word.Range
    'Den Bereich nach der Tabelle und eine neue Absatzmarke festlegen
    Dim rng As Word.Range

    Set rng = tbl.Range
    rng.Collapse Direction:=wdCollapseEnd
    rng.InsertParagraphAfter
    rng.Collapse Direction:=wdCollapseEnd

    Set BereichNachTabelleFestlegen = rng
End Function

```



Da sich bei der C#-Automatisierung die Word-Anwendung ohne Dokument öffnet, wird in Listing 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 .NET Framework Datenfelder (Arrays) anders handhabt.

Listing 7.7 (.NET): Die C#-Version von Listing 7.6



```

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;
}

```

Listing 7.7 (.NET): Die C#-Version von Listing 7.6 (Fortsetzung)

```

int zaehlerZeile;
int zaehlerSpalte;
string tab = "\t";
string[,] aDaten = new string[DIM_1, DIM_2];
object objMissing = System.Reflection.Missing.Value;

DatenZuweisen(ref aDaten);
//Zeilen nach Bedarf erstellen und Daten einfügen
object objTableBehavior8 = wd.WdDefaultTableBehavior.wdWord8TableBehavior;
object objTableAutoFit = wd.WdAutoFitBehavior.wdAutoFitWindow;
wd.Table tbl = doc.Tables.Add(rng, 1, AnzSpalten, ref objTableBehavior8,
    ref objTableAutoFit);
wd.Row row = tbl.Rows[1];
for(zaehlerZeile = aDaten.GetLowerBound(0);
    zaehlerZeile<= aDaten.GetUpperBound(0); zaehlerZeile++)
{
    //Daten in die letzte Zeile eingeben.
    for (zaehlerSpalte = aDaten.GetLowerBound(1);
        zaehlerSpalte<= aDaten.GetUpperBound(1); zaehlerSpalte++)
    {
        row.Cells[zaehlerSpalte + 1].Range.Text = aDaten[zaehlerZeile, zaehlerSpalte];
    }
    //Neue Zeile nach Bedarf einfügen.
    if (row.Index != AnzZeilen)
    {
        row = null;
        row = tbl.Rows.Add(ref objMissing);
    }
    rng=null;
}
rng = BereichNachTabelleFestlegen(tbl);

//Alle Zeilen erstellen, dann die Daten einfügen.
tbl = null;
tbl = doc.Tables.Add(rng, AnzZeilen, AnzSpalten, ref objTableBehavior8,
    ref objMissing);
for (zaehlerZeile = aDaten.GetLowerBound(0);
    zaehlerZeile <= aDaten.GetUpperBound(0);zaehlerZeile++)
{
    for (zaehlerSpalte = aDaten.GetLowerBound(1);
        zaehlerSpalte <= aDaten.GetUpperBound(1); zaehlerSpalte++)
    {
        tbl.Cell(zaehlerZeile + 1, zaehlerSpalte + 1).Range.Text =
            aDaten[zaehlerZeile, zaehlerSpalte];
    }
}

rng = null;
rng = BereichNachTabelleFestlegen(tbl);
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++)
{

```


Listing 7.7 (NET): Die C#-Version von Listing 7.6 (Fortsetzung)

```

        for (zaehlerSpalte = aDaten.GetLowerBound(1);
            zaehlerSpalte <= aDaten.GetUpperBound(1); zaehlerSpalte++)
        {
            daten += aDaten[zaehlerZeile, zaehlerSpalte];
            if (zaehlerSpalte != aDaten.GetUpperBound(1))
            {
                daten += tab;
            }
        }
        if (zaehlerZeile != aDaten.GetUpperBound(1))
        {
            daten += "\n";
        }
    }
    rng.Text = daten;
    object objTab = (object) tab;
    tbl = rng.ConvertToTable(ref objTab, ref objMissing, ref objMissing, ref objMissing,
        ref objMissing, ref objMissing, ref objMissing, ref objMissing, ref objMissing,
        ref objMissing, ref objMissing, ref objMissing, ref objMissing, ref objMissing,
        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 (ab Word 2003) 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-Dateiformat finden Sie in Teil V dieses Buches. Weitere Informationen zu allen Dateiformaten befinden sich auf der MSDN-Webseite bei Microsoft.com.

CD-ROM Die Beispieldatei *Bsp07_01_Table.docm* 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 Absatzzeigenschaften sowie den Textfluss innerhalb von Tabellenzellen als auch Rahmen und Schattierungen. Diese werden mit den gleichen Befehlen ausgeführt, wie sonst für die Formatierung von Text in der Word-Umgebung und dürfen in einer Tabellenformatvorlage festgehalten und damit einer Tabelle zugewiesen werden.

Als zum grafischen Layout gehörend betrachten wir alles, was mit der Größe der Tabelle und ihrer Positionierung auf der Seite zu tun hat. Hierzu gehören Spaltenbreite, Zeilenhöhe, Spalten- und Zeilenanzahl, Zellenverbindungen und die Textflussformatierung um die Tabelle. Diese Eigenschaften können nicht als Teil einer Tabellenformatvorlage definiert werden.

Die nicht-grafische Formatierung

Die Formatierung von Text innerhalb einer Tabelle wird genau gleich vorgenommen wie außerhalb der Tabelle. Die Formatierung wird einer Markierung (Selection) oder einem Bereich (Range) zugewiesen. Hier einige Beispiele:

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 Rows-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
```

ACHTUNG Eigentlich enthält jede Tabellenzelle zwei »verborgene« Zeichen, wie im Abschnitt »Informationen aus Tabellen holen« ab Seite 372 beschrieben. Ab Word 2007 ist das letzte anscheinend nicht mehr gleich »sichtbar«, wie bisher. Somit müssen alle Prozeduren, die auf diesem Prinzip aufbauen, für Word 2007 und 2010 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 *Tabelleneigenschaften*, das über *Tabellentools/Layout* in der Gruppe *Tabelle* erreicht wird. Die Abbildung 7.5 veranschaulicht die Wirkung der vorgestellten Anweisungen.

Links sehen Sie die Tabelle, bevor die Textflussformatierung geändert wurde; rechts die Folgen dieser Formatierungen:

- Die ersten zwei Zeilen werden auf den folgenden Seiten wiederholt
- Der Text innerhalb einer Zelle darf nicht über die Seite umbrechen
- Der erste Absatz der vierten Zelle in Spalte 1 darf nicht in die nächste Zeile umbrechen
- Der Tabellenrand (statt dem Text in der Tabelle) steht linksbündig mit dem Dokumenttext. Dies fällt weniger auf, weil der Abstand zwischen Text und Zellrand auf Null gesetzt wurde.

Abbildg. 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	

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 die Schaltfläche *Rasterlinien anzeigen* in *Tabellentools/Layout*, in der Gruppe *Tabelle* gesteuert. 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 Beispielcode:

```
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 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 Werts 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 Gruppe *Ausrichtung* der Registerkarte *Tabellentools/Layout* als die Schaltfläche *Textrichtung* wieder. 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)
```

Zeilenumbruch/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, sodass der erste Absatz der Zelle in die Zeilenbreite 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;
```

CD-ROM

Die Beispieldatei *Bsp07_02_Table.docm*, woraus die obigen Codefragmente 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

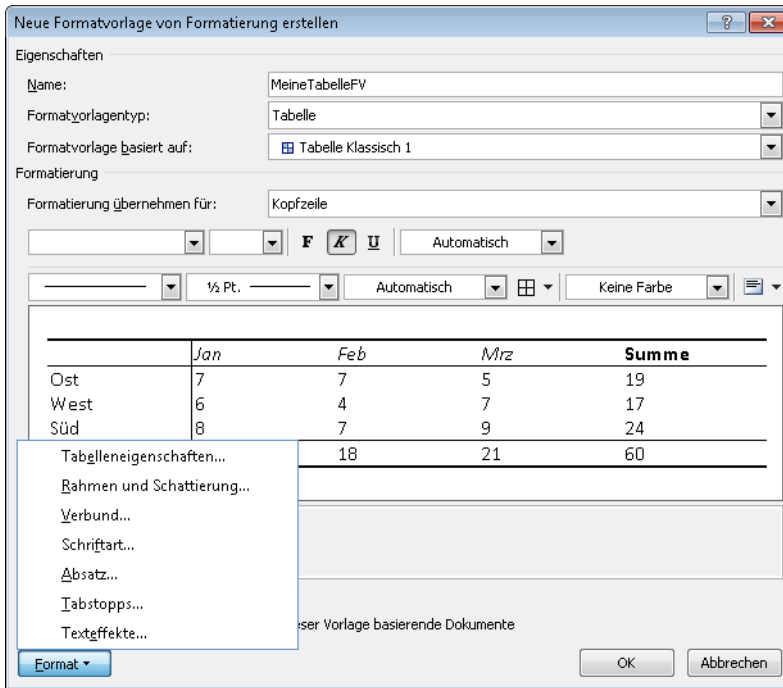
Tabellenformatvorlagen wurden in Word 2002 eingeführt. Falls ein Dokument, worin Tabellenformatvorlagen für die Formatierung benutzt wurden, in einer früheren Word-Version geöffnet wird, gehen die Tabellenformatvorlagen verloren. Die Konvertierung behält alle Tabellenformatierungen wie Rahmenlinien und 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* ein. 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 »Kalenderblatt erstellen« auf der CD-ROM zum Buch im Ordner \Bonus\Kapl.

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, vor der Version 2010 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. Das Problem ist in Word 2010 behoben worden.

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 darauffolgende (also die zweite der gesamten Tabelle). Die Anzahl der Zeilen und/oder Spalten in einem Verbund bestimmen Sie über den gleichnamigen Eintrag.

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 eventuell andere Rahmenformatierungen ausschalten

	2000	2001	2002	Durchschnitt
Orangen	1980	2300	2000	1990
Bananen	1650	1800	1900	1775
Ananas	850	900	740	795
Summen	6480	7001	6642	

Eckzellen

Falls Sie einer Eckzelle eine besondere Formatierung zuweisen, werden Sie unter Umständen feststellen, dass die Zelle sich scheinbar weigert, diese anzunehmen. Und zwar kommen Eckzellenformatierungen erst zum Vorschein, wenn der Zeile und der Spalte, die sich an diesem Punkt kreuzen, auch eine Formatierung zugeteilt wurde. Diese Einschränkung können Sie in der Benutzeroberfläche umgehen, indem Sie der Zeile und der Spalte den Schriftschnitt *Fett* zuweisen und umgehend wieder ausschalten. Somit haben diese Elemente die Formatierung »nicht Fett« und die Eckzelle erscheint mit ihrer Formatierung.

Tabelleneigenschaften

Obwohl unter der *Format*-Schaltfläche der Eintrag *Tabelleneigenschaften* erscheint, stehen viele der darin enthaltenen Attribute den Tabellenformatvorlagen nicht zur Verfügung. Es ist unmöglich, einen Textfluss für die Tabelle oder eine bevorzugte Spaltenbreite oder Zeilenhöhe zu bestimmen. Die Tabellen- und 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 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 neue 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.dotm* 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:

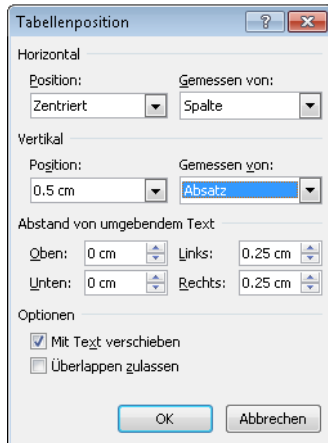
```
ActiveDocument.Tables(1).Rows.Alignment = wdAlignRowCenter
```



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 der Mauszeiger über 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 `RelativeVerticalPosition`, 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 InlineShape- 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

<code>WdTablePosition-Enum</code>	Wert	Beschreibung
<code>wdTableBottom</code>	–999997	Die Tabelle wird bündig zum unteren Text- oder Seitenrand positioniert
<code>wdTableCenter</code>	–999995	Die Tabelle wird senkrecht oder waagrecht zwischen Text- oder Seitenrändern positioniert
<code>wdTableInside</code>	–999994	Die Tabelle wird am inneren Text- oder Seitenrand positioniert, wenn <i>Gerade/ungerade anders</i> in <i>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</i> in <i>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 einer einzigen Seite beschränkt. In späteren Versionen ist dieses Verhalten nicht nur erlaubt, sondern auch standardmäßig aktiviert und kann in der Kategorie *Erweitert* der *Optionen* im Abschnitt *Kompatibilitätsoptionen für* unterbunden werden. Im Objektmodell ist 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 Word 2007 oder Word 2010 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*

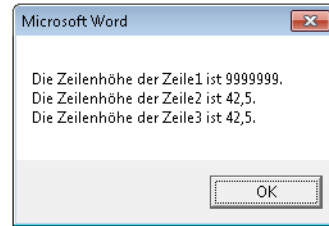
```
Sub TabellenZeilenHöhen()
    Dim strZeilenhöhen As String
    Dim tbl As Word.Table
    Dim row As Word.Row

    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 Mindesthöhe. Sie wird aber wachsen, wenn sich mehr Text darin befindet, 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.

CD-ROM

Die Beispieldatei *Bsp07_03_Table.docm* 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, 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 Tabellenzeile eine verborgene Absatzmarke (Zeichencode 13) sowie ein »Ende-der-Zelle-Zeichen« (Zeichencode 7) befinden. Meistens sind diese nicht erwünscht und müssen abgetrennt werden, wofür die Funktion *TrimZellenInhalt* in Listing 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
    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 Elemente 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)
```

Listing 7.10 Daten aus einer Tabelle Zelle für Zelle auslesen und in ein Array aufnehmen (Fortsetzung)

```
str = Left(str, Len(str) - 2)
TrimZellenInhalt = str
End Function
```

CD-ROM

Die Beispieldatei *Bsp07_04_Table.docm* finden Sie auf der CD-ROM im Ordner *\Beispiele\Kap07*.

Convert-
ToText

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 (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 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, sodass der Originaltext aus den Tabellenzellen gespeichert wird.

Am Schluss der Hauptprozedur werden die drei im Dokument ausgeführten Handlungen rückgängig gemacht, sodass 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
```

Listing 7.11 Eine Tabelle in eine zeichengetrennte Zeichenkette umwandeln und diese in ein Array schreiben (Fortsetzung)

```

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()
'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

```

Listing 7.11 Eine Tabelle in eine zeichengetrennte Zeichenkette umwandeln und diese in ein Array schreiben (*Fortsetzung*)

```
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
```

CD-ROM Die Beispieldatei *Bsp07_04_Table.docm* 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 Es bietet sich die Alternative, das XML-Format der Tabelle zu bearbeiten, die in eine Zeichenkette gelesen werden kann. 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 Zeile der zweiten Zeile eingefügt, sondern in die zweite Zeile der ersten Zeile, und dies ohne eine Fehlermeldung oder sonstigen Hinweis, dass die Zelle nicht vorhanden wäre. Äußerste Vorsicht ist also im programmtechnischen 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 Zeile geschrieben.
    tblInnere2.Cell(2, 2).Range.Text = "Hallo"
    Debug.Print "Anzahl Zeilen: " & tblInnere2.Rows.Count
End Sub

Function TabelleEinfuegen(rng As Word.Range, lAnzZeilen As Long, _
    lAnzSpalten As Long) As Word.Table
    Dim tbl As Word.Table
    Dim lZaehler As Long

    Set tbl = rng.Tables.Add(rng, lAnzZeilen, lAnzSpalten)
    'Bei verschachtelten Tabellen wird nur eine Zeile erstellt!
    For lZaehler = tbl.Rows.Count To lAnzZeilen - 1
        tbl.Rows.Add
    Next

    Set TabelleEinfuegen = tbl
End Function
```

Abbildg. 7.11 Das Ergebnis von Listing 7.12

	Spalten Überschrift		
		Hallo	

CD-ROM Die Beispieldatei *Bsp07_05_Table.docm* 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
```

Listing 7.13 Alle Zellen einer Tabelle sowie alle Zellen von eventuell darin verschachtelten Tabellen durchschleifen (*Fortsetzung*)

```
Next tbl
End If
Next cel
End Sub
```

CD-ROM Die Beispieldatei *Bsp07_05_Table.docm* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap07*.

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 aufweisen.«

Die beste Methode, um der Struktur einer Tabelle auf den Grund zu gehen, ist, die Tabelle in ein neues Dokument zu kopieren, dieses als gefiltertes HTML zu speichern und die HTML-Datei zu bearbeiten. Dort stehen, wie in Abbildung 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 gespeicherten Word-Tabelle, im Texteditor geöffnet. Der grau hinterlegte Text hebt die nützlichen Informationen hervor.

```
<td width=224 colspan=2 valign=top style='width:167.7pt;border-top:none;
border-left:none;border-bottom:solid windowtext 1.0pt;border-right:solid
windowtext 1.0pt; padding:0cm 5.4pt 0cm 5.4pt'> <p class=HsoNormal>&nbsp;&nbsp;&nbsp;</p>
</td>
</tr>
<tr>
```

Die zweitbeste Methode ist, die XML-Eigenschaft der Tabellenbereich abzufragen und deren Resultat zu analysieren (mehr über das XML-Format von Word erfahren Sie in Kapitel 21).

Muss es wirklich mit den Bordmitteln des Objektmodells gehen, bereiten Sie sich auf etwas Kopfzerbrechen vor. Das Beispieldokument *Bsp07_04_Table.docm* enthält eine Prozedur *VerbundeneTabellenZellen*, die senkrecht sowie waagrecht verbundene Zellen aufspürt. 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.

CD-ROM

Die Beispieldatei *Bsp07_04_Table.docm* 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« ab Seite 380 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.

In diesem Abschnitt werden Feldfunktionen vom Entwicklerstandpunkt aus betrachtet, mit einigen kurzen Beispielen für deren Einsatz.

HINWEIS

In der Benutzerschnittstelle öffnen Sie über den Befehl *Feld* im Dropdownmenü zur Schallfläche *Schnellbausteine* in der Gruppe *Text* der Registerkarte *Einfügen* 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 auf der Buch-CD im Anhang B für Sie bereitsteht. Außerdem finden Sie dort weitere allgemeine Informationen zum Thema Feldfunktionen. Früher enthielt die Anwendungshilfedatei detaillierte Angaben über Zweck und Verwendung einzelner Feldfunktionen. Seit Word 2007 wurde diese Information arg dezimiert. Die beste Microsoft-Quelle für diese Informationen ist die Webseite <http://office.microsoft.com/en-us/word/CH061047231033.aspx>, die kein deutschsprachiges Gegenstück kennt. Lediglich die Word 2003-Hilfe zum Thema ist online vorhanden: <http://office.microsoft.com/de-de/results.aspx?qu=Feldfunktion+Word>.

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</code> -Enum	Wert	Beschreibung
<code>wdFieldKindCold</code>	3	Ein Feld, das kein Ergebnis hat (z.B. XE-Felder (Indexeintrag), TC-Felder (Verzeichniseintrag) oder private Felder)
<code>wdFieldKindHot</code>	1	Ein Feld, das automatisch jedes Mal, wenn es angezeigt wird oder wenn die Seite neu formatiert wird, aktualisiert wird, das aber auch manuell aktualisiert werden kann (z.B. <code>INCLUDEPICTURE</code> oder <code>FORMDROPDOWN</code>)
<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. <code>DATE</code> oder <code>INCLUDETEXT</code>)

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 Codefragment ver-

anschaulicht, 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 ein Backslash, gefolgt von einem Buchstaben signalisiert. Jeder Buchstabe steht für eine bestimmte, für die jeweilige Feldfunktion spezifische Option, die in der Feldfunktion-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*, mit dem Befehl *Eintrag festlegen* in der Gruppe *Index* der Registerkarte *Verweise* eingefügt. Die Schalter \e, \c, und \z legen die Trennzeichen zwischen einem Indexeintrag und der zugehörigen Seitenzahl, die Anzahl der Spalten sowie die Sprache für die Sortierreihenfolge fest. Jede Feldfunktion hat einen eigenen Satz von Schaltern; der gleiche Buchstabe kann für mehrere Feldfunktionen eine ganz andere Bedeutung haben.

Abbildg. 7.14 Eine Feldfunktion, um einen Index zu generieren. Die graue Schattierung wird über *Optionen/Erweitert/Dokumentinhalt anzeigen* 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 auf der Buch-CD. 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 Feldegebnis vorgenommene Zeichen- und Tabellenformatierungen, sodass 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 Abschnitt über Grafiken in Kapitel 6), um die Grafikgröße festzuhalten, und in Feldern, deren Ergebnisse 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:=wdFieldIndex, Text:="\e " & strQuote & _
    vbTab & strQuote & " \c " & strQuote & "2" & strQuote & " \z " & strQuote & "1031" _
    & strQuote, PreserveFormatting:=False)

'Ohne Angabe des Typs
Set fld = rng.Fields.Add(Range:=rng, Text:"Index \e " & strQuote & _
    vbTab & strQuote & " \c " & strQuote & "2" & strQuote & " \z " & strQuote & "1031" _
    & strQuote, PreserveFormatting:=False)
```



In C#:

```
string quote = "\""; //Anführungszeichen;
Wd.Range rng = WdApp.Selection.Range;
//Mit Angabe des Typs
object objFieldType = (object) Wd.WdFieldType.wdFieldIndex;
object objFieldText = (object) ("\\e " + quote + "\\t" + quote
    + " \\c " + quote + "2" + quote + " \\z " + quote + "1031" + quote);
object objFalse = false;
Wd.Field fld = rng.Fields.Add(rng, ref objFieldType, ref objFieldText, ref objFalse);

fld = null;
//Ohne Angabe des Typs
objFieldText = (object) ("Index \\e " + quote + "\\t" + quote
    + " \\c " + quote + "2" + quote + " \\z " + quote + "1031" + quote);
fld = rng.Fields.Add(rng, ref objMissing, ref objFieldText, ref objFalse);
```

HINWEIS

Nicht alle Feldfunktionen können überall in ein Dokument eingefügt werden, wo Text stehen kann. Insbesondere mahnen wir bei AutoFormen (Textfeldern) zur Vorsicht. Auch wenn Word es zulässt, eine Feldfunktion in ein Textfeld einzufügen, ist es nicht gewährleistet, dass sie korrekt aktualisiert oder von Word »gesehen« wird. Das Paradebeispiel hierfür stellen Querverweise und Einträge für Inhaltsverzeichnisse dar, welche 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 verschiedenen 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
```

Listing 7.14 Die Pfadangabe einer jeden Grafik, die in der Zeile mit dem Text steht, ändern (Fortsetzung)

```

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

```

CD-ROM Die Beispieldatei *Bsp07_01_Field.docm* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap07*. Die Datei wurde im Word 2010-Kompatibilitätsmodus gespeichert, um die *IncludePicture*-Feldfunktionen zu erhalten.

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. Mithilfe 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

```

Listing 7.15 Ermitteln, ob sich die Markierung in einer Feldfunktion befindet (Fortsetzung)

```

Set rngDok = rngMarkierung.Duplicate
'Der Bereich erstreckt sich vom Dokumentanfang ...
rngDok.Start = ActiveDocument.Range.Start
'... bis zu einem Zeichen nach der Markierung, falls die Markierung
'am Anfang eines Feldes steht (das Feld ist grau hinterlegt).
rngDok.End = rngMarkierung.End + 1
'Auch der markierte Bereich wird um ein Zeichen erweitert
rngMarkierung.End = rngMarkierung.End + 1
'Die Anzahl der Felder bis zur Markierung ermitteln
lAnzFelder = rngDok.Fields.Count
Set fld = ActiveDocument.Fields(lAnzFelder)

'Falls ein verborgenes Feld wie XE, RD oder TC vorliegt ...
If fld.Kind = wdFieldKindCold Then
    '... wird der Feldcodebereich mit dem Markierungsbereich verglichen.
    If rngMarkierung.InRange(fld.Code) Then
        Debug.Print "Ja, im FeldCode"
        bInFeld = True
    End If
Else
    'Sonst wird der Feldresultatbereich mit dem Markierungsbereich verglichen.
    If rngMarkierung.InRange(fld.Result) Then
        Debug.Print "Ja, im Feldresultat."
        bInFeld = True
    End If
End If
'Liegt die Markierung am Feldanfang, wird der Startpunkt des Feldcodebereichs
'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

```

CD-ROM Die Beispieldatei *Bsp07_01_Field.docm* 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 *Macro-button*-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.

CD-ROM

Die Prozeduren *ButtonTestAusführen* und *ButtonTest*, die die beschriebene Funktionalität veranschaulichen, finden Sie in der Beispieldatei *Bsp07_01_Field.docm* 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, sei 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>.

Verknüpfungen

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 abzuplagen.

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, sodass 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 2010 gilt, sowie *Excel.Sheet.12* in Office 2010), 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$(34) &
"C:\\Beispiele\\Kap07\\Bsp07_02_Field.xls" & Chr$(34) & " " & Chr$(34) & _
"Tabelle1!wdFieldType" & Chr$(34) & " \\a \\f 5 \\h \\* MERGEFORMAT"
```

Und für ein Arbeitsblatt aus einer Excel 2010-Arbeitsmappe, mit Zellenreferenz statt Bereichsnamen:

```
"LINK Excel.Sheet.12 " & Chr$(34) &
"C:\\Beispiele\\Bsp07_02_Field.xlsx" & Chr$(34) & " " & Chr$(34) & _
"Tabelle1!Z1S1:Z3S3" & Chr$(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)
wdFieldEditTimes	(neu in Word 2002)	EditTimes	Advance	Versetzen
wdFieldExpression	=(Ausdruck)	=(Formula)	Ask	Frage
wdFieldSection	Abschnitt	Section	Author	Author
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	Author	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}
```


CD-ROM

Die Beispieldateien *Bsp07_02_Field.docm*, *Bsp07_02_Field.xls* und *Bsp07_02_Field.xlsx* 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\\Kap07\\Hydrangeas.jpg, 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 im Word-Dokument 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 ab Version 2003 weder in der Hilfe noch online. Am besten gehen Sie über den Befehl *Feld*, wählen dort den Eintrag *= (Formel)*, und danach die Schaltfläche *Formel betätigen*. Im Dialogfeld geben Sie die mathematische Formel ein. Die Formel kann mit Einträgen aus den Dropdownlisten *Zahlenformat*, *Funktionen einfügen* und *Textmarke einfügen* ergänzt werden. Das ins Dokument eingefügte Resultat liefert den Text für die *Add*-Methode. Wenn Sie die korrekte Syntax noch nicht kennen, kann dies ohne helfende Beispiele frustrierend sein. Wir befassen uns deshalb hier mit zwei Aspekten.

Auf Tabelleninhalt verweisen

Früher stand im Hilfethema, wie Tabellenbezüge zu formulieren sind. Darunter befand 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) }`.

CD-ROM

Die Beispieldatei *Bsp07_02_Field.docm* 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 erledigt. Programmtechnisch gestaltet sich die Aufgabe als äußerst komplex, weil das Verschachteln von Feldfunktionen im Objektmodell nicht ausdrücklich vorgehen 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. Größere Zahlen verursachen eine Fehlermeldung als Feldfunktionsergebnis. In der heutigen Zeit sind Millionenbeträge alltäglich geworden; eine Anpassung der Funktionalität drängt sich auf.

Oben in Abbildung 7.16 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 } " } }
```

zehnmillionendreihundertsechsfünfzigtausendzweihundertachtunddreißig

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 der Feldklammern eingefügt, und der 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
    bEscapeCharFound = False
    sBuffer = ""
    Set rngStart = rngZielBereich.Duplicate

    rngZielBereich.TextRetrievalMode.IncludeFieldCodes = True
    For lIndex = 1 To Len(strFeldCode)
        sChar = Mid(strFeldCode, lIndex, 1)
        Select Case sChar
            Case EscapeChar:
                If bEscapeCharFound Then
                    sBuffer = sBuffer + sChar
                    bEscapeCharFound = False
                Else
                    bEscapeCharFound = True
                End If
            Case "{"
                If bEscapeCharFound Then
                    sBuffer = sBuffer + sChar
                    bEscapeCharFound = False
                Else
                    ' Wir sind am Anfang einer Feldfunktion angelangt.
                    rngZielBereich.Text = sBuffer
                    sBuffer = ""
                    rngZielBereich.Collapse direction:=wdCollapseEnd
                    rngZielBereich.Fields.Add Range:=rngZielBereich, Type:=wdFieldEmpty, _
                        Text:=sBuffer, PreserveFormatting:=False
                    rngZielBereich.SetRange Start:=rngZielBereich.Start + 1, _
                        End:=rngZielBereich.Start + 1
                    rngZielBereich.Delete unit:=wdCharacter, Count:=2
                End If
            Case "}"
                ' ... (rest of the function code)
        End Select
    Next lIndex
    rngZielBereich.Text = sBuffer
    rngZielBereich.Collapse direction:=wdCollapseEnd
    rngZielBereich.Fields.Add Range:=rngZielBereich, Type:=wdFieldEmpty, _
        Text:=sBuffer, PreserveFormatting:=False
    rngZielBereich.SetRange Start:=rngZielBereich.Start + 1, _
        End:=rngZielBereich.Start + 1
    rngZielBereich.Delete unit:=wdCharacter, Count:=2

    rngZielBereich
End Function
```

Listing 7.16 Eine Zeichenkette in eine Feldfunktion umwandeln (Fortsetzung)

```

        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

    Set rngTarget = Selection.Range
    strTeststrFeldCode = rngTarget.Text
    rngTarget.InsertAfter vbCr
    rngTarget.Collapse wdCollapseEnd
    Set rngTarget = FeldCodeEinfuegen(strTeststrFeldCode, rngTarget)
    rngTarget.Fields.Update
End Sub

```

Feldfunktion in Text umwandeln

Das Gegenstück zur Funktion, die Text in eine Feldfunktion umwandelt, ist eine Prozedur, die eine Feldfunktion in Text konvertiert, wie in Listing 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 und kopiert ihn in die Zwischenablage

```

Sub FeldCodeInText()
    Dim rng As Word.Range
    Dim strFeldCode As String
    Dim strNeu As String

```

Listing 7.17 Diese Prozedur wandelt die markierte Feldfunktion in einfachen Text um und kopiert ihn in die Zwischenablage (*Fortsetzung*)

```

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
'es einfügen kann, wo er will.
Set oData = New DataObject
oData.SetText strNeu
oData.PutInClipboard
End Sub

```

CD-ROM Die Beispieldatei *Bsp07_02_Field.docm* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap07*.

Formulare: das *FormField*-Objekt

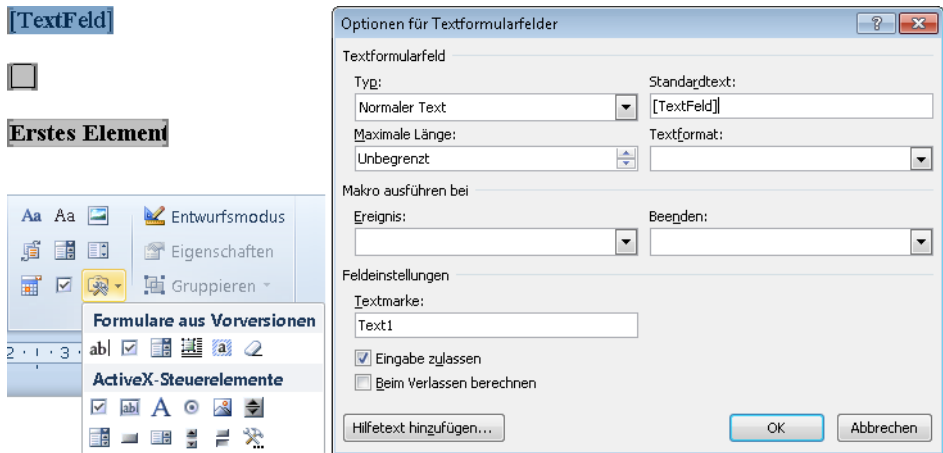
Viele Dokumentarten wie Memos und Faxnachrichten muss der Anwender wiederholt erstellen. Um den Arbeitsvorgang zu beschleunigen und zu vereinfachen, werden hierfür Dokumentvorlagen bereitgestellt. Diese können sogar mit vorhandenen Daten über den Seriendruck oder eine automatisierte Lösung bestückt werden, 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 Registerkarte *Entwickler-tools*. Die Formularfelder sind in der Liste *Vorversionstools* der Gruppe *Steuerelemente* aufgelistet. Der

Aufgabenbereich *Formatierung und Bearbeitung einschränken*, wird über die Schaltfläche *Bearbeitung einschränken* der Gruppe *Schützen* eingeblendet.

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 Felds. 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 Liste der Formular-Werkzeuge sowie ein *Optionen*-Dialogfeld

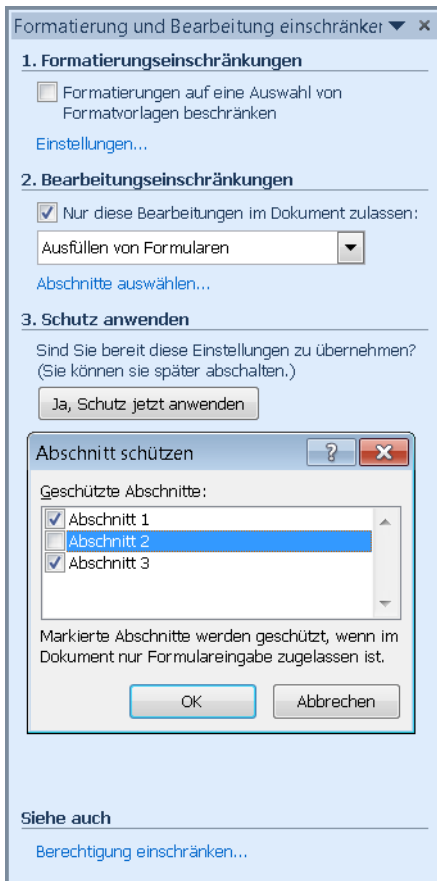


Formularfelder können nur als solche benutzt werden, wenn das Dokument als ein Formular geschützt ist. Nur dann kann durch Drücken der [F1]-Taste oder mit der Maus zwischen den Formularfeldern navigiert werden. Um Teile eines Dokuments für die normale Bearbeitung freizustellen, können Abschnittswechsel 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 Codeablaufs zuweisen müssen, muss das Dokument im ungeschützten Zustand sein.

Formularfeldwerte

Um den Textinhalt eines Textfelds 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
```

Besser ist, ein Kontrollkästchen konsequent über die Eigenschaft `Value` festzulegen:

```
doc.FormFields("Kontrollkästchen1").CheckBox.Value = True
```

ACHTUNG

Der Zustand eines Kontrollkästchens kann ebenfalls über die `Result`-Eigenschaft abgefragt werden. In Word 2007 kann es auch so deaktiviert, aber nicht aktiviert werden. Das Gegenteil gilt für Word 2010: Zuweisung eines Werts (egal welcher) aktiviert ein Kontrollkästchen; es kann über die `Result`-Eigenschaft aber nicht deaktiviert werden:

```
lKontrollkästchenWert = doc.FormFields("Kontrollkästchen1").Result
doc.FormFields("Kontrollkästchen1").Result = 0 'Nicht aktiviert. Aktiviert wäre 1
```

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 (.NET): Die C#-Version von Listing 7.18. 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;
}

```

CD-ROM Die Beispieldatei *Bsp07_01_Form.docm* 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 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 *Formatierung und Bearbeitung einschränken* (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, sodass 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 (.NET): Die C#-Version von Listing 7.20



```
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.

CD-ROM Die Beispieldatei *Bsp07_01_Form.docm* 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), da es keine Optionsfelder als Formularfelder gibt. 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 Ereignismakros

☒ Option 1

☐ Option 2

☐ Option 3

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 Formularfelds 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-Makropaar. Beim Eintreten wird der Formularfeldname in einer Dokumentvariablen gespeichert. Beim Verlassen wird die Gültigkeitsprüfung durchgeführt und deren Ergebnis in einer weiteren Dokumentvariablen gespeichert. Beim Eintreten in das nächste Feld wird im Falle eines ungültigen Ergebnisses zurück zum ersten Feld gesprungen, ansonsten wird der Name des nächsten Felds in die Dokumentvariable geschrieben.

Listing 7.22 Kontrollkästchen wie Optionsfelder 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.CheckBox.Value = False
            End If
        Next
    End If
End Function

```

CD-ROM Die Beispieldatei *Bsp07_01_Form.docm* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap07*.

HINWEIS

In Kapitel VII des Bonusteils auf der CD-ROM zum Buch 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: das *ContentControls*-Objekt

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« ab Seite 393) und ActiveX-Steuerelemente (Kapitel 12) an. Der große Nachteil der Formularfelder 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 21) 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 ab Version 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 im Word-Objektmodell üblich

In den folgenden Abschnitten werden Inhaltssteuerelemente und ihre programmtechnischen Schnittstellen eingehender vorgestellt.

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 in Word 2007 und Optionsfeldern in allen Versionen. 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* in Word 2010

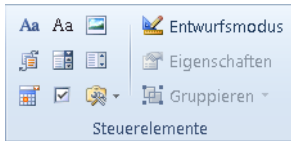





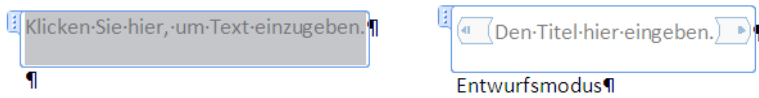
Tabelle 7.4 Die einzelnen Inhaltssteuerelemente mit *WdContentControlType*-Enumeration

Symbol	Name	Enum	Konstantwert
	Rich-Text (formatierter Text)	<code>wdContentControlRichText</code>	5
	Nur-Text	<code>wdContentControlText</code>	3
	Bild	<code>wdContentControlPicture</code>	6
	Bausteinkatalog	<code>wdContentControlBuildingBlockGallery</code>	1
	Kombinationsfeld	<code>wdContentControlComboBox</code>	4
	Dropdownliste	<code>wdContentControlDropDownList</code>	2

Tabelle 7.4 Die einzelnen Inhaltssteuerelemente mit *WdContentControlType*-Enumeration (Fortsetzung)

Symbol	Name	Enum	Konstantwert
	Datumsauswahl	wdContentControlDate	0
	Kontrollkästchen*	wdContentControlCheckBox	8
	Gruppieren	wdContentControlGroup	7
	*Nur in Word 2010		

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 (rechts im Bild), klicken Sie auf die Schaltfläche *Entwurfsmodus* und geben Sie den gewünschten Text ein.


Abbildg. 7.21 Nur-Text-Inhaltssteuerelemente im normalen sowie Entwurfsmodus

HINWEIS

Um die Anzeige des Platzhaltertexts zu unterdrücken, wird die Eigenschaft *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. Die Tabelle 7.5 bietet eine Übersicht aller Inhaltssteuerelement-Eigenschaften. Tiefer gehende Informationen dazu finden Sie in der Hilfe zum Word-Objektmodell.

Abbildg. 7.22 Eigenschaftenfenster eines Grafik-Inhaltssteuerelements, die alle Inhaltssteuerelemente gemeinsam haben

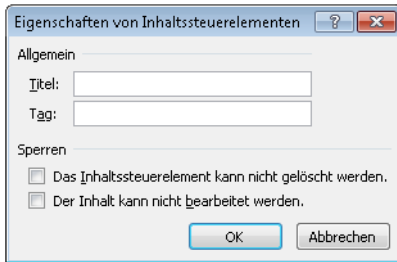


Tabelle 7.5 Eigenschaften der Inhaltssteuerelemente




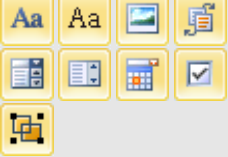
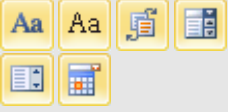
Eigenschaft	Beschreibung	Steht den Inhaltssteuerelementen zur Verfügung
<i>Titel</i>	Eine Beschriftung, die über dem Inhaltssteuerelement erscheint, wenn dieses sichtbar ist. Dieser Wert muss <i>nicht</i> zwingend eindeutig sein. Steht dem Entwickler für die Identifikation zur Verfügung über die Methode SelectContentControlsByTitle .	Alle 
<i>Tag</i>	Stellt dem Entwickler einen Datenbehälter für jedes Inhaltssteuerelement bereit. Steht dem Entwickler für die Identifikation zur Verfügung über die Methode SelectContentControlsByTag .	Alle 
<i>Das Inhaltssteuerelement kann nicht gelöscht werden</i>	Der Benutzer kann das Inhaltssteuerelement nicht aus dem Dokument löschen. Wird im Objektmodell durch die Eigenschaft LockContentControl dargestellt.	Alle 
<i>Der Inhalt kann nicht bearbeitet werden</i>	Das Inhaltssteuerelement ist gesperrt. Wird im Objektmodell durch die Eigenschaft LockContents dargestellt.	Alle 
<i>Formatvorlage zum Formatieren von Inhalt verwenden</i>	Der Inhalt wird zwingend mit der angegebenen Zeichen-Formatvorlage formatiert. Wird im Objektmodell durch die Eigenschaft DefaultTextStyle dargestellt.	Nur-Text, Rich-Text, Dropdownliste, Kombinationsfeld, Datumsauswahl, Bausteinkatalog 

Tabelle 7.5 Eigenschaften der Inhaltssteuerelemente (Fortsetzung)

















Eigenschaft	Beschreibung	Steht den Inhaltssteuer- elementen zur Verfügung
<i>Wagenrückläufe zulassen</i>	Ermöglicht die Eingabe mehrerer Absätze in ein Nur-Text-Inhaltssteuerelement. Wird im Objektmodell durch die Eigenschaft MultiLine dargestellt.	Nur-Text 
<i>Inhaltssteuerelement beim Bearbeiten des Inhalts löschen</i>	Das Inhaltssteuerelement wird bei der Texteingabe aus dem Dokument entfernt. Wird im Objektmodell durch die Eigenschaft Temporary dargestellt.	Nur-Text, Rich-Text  
<i>Anzeigenname</i>	Die Beschriftung eines Listeneintrags. Wird im Objektmodell durch die Eigenschaft Text eines DropDownListEntries.Item dargestellt.	Dropdownliste, Kombinationsfeld  
<i>Wert</i>	Unsichtbarer Inhalt eines Listeneintrags; hat den Datentyp »String«. Wird im Objektmodell durch die Eigenschaft Value eines DropDownListEntries.Item dargestellt.	Dropdownliste, Kombinationsfeld  
<i>Die Schaltflächen Hinzufügen sowie Nach oben und Nach unten</i>	Fügt der Auflistung einen neuen Eintrag zu. Wird im Objektmodell durch die Methode Add dargestellt. Ändert die Position eines Eintrags in der Liste. Werden im Objektmodell durch die Methoden MoveUp und MoveDown dargestellt. Die gegenwärtige Position gibt die Eigenschaft Index zurück.	Dropdownliste, Kombinationsfeld  
<i>Datum wie folgt anzeigen</i>	Legt das Erscheinungsbild des Inhalts fest. Wird im Objektmodell durch die Eigenschaft DateDisplayFormat dargestellt.	Datumsauswahl 
<i>Gebietsschema</i>	Legt das übergeordnete regionale Datumsformat fest. Wird im Objektmodell durch die Eigenschaft DateDisplayLocale dargestellt, die einen WdLangaugeID -Konstantwert erwartet..	Datumsauswahl 
<i>Kalendertyp</i>	Legt die Art des Kalenders (beispielsweise chinesisch, hebräisch, arabisch usw.) fest. Wird im Objektmodell durch die Eigenschaft DateCalendarType dargestellt, die einen WdCalendarType -Konstantwert erwartet.	Datumsauswahl 
<i>XML-Inhalt im folgenden Format speichern</i>	Legt das Format (Datum, Datum & Zeit oder Zeichenkette) fest, in welchem der Inhalt gespeichert werden soll. Wird im Objektmodell durch die Eigenschaft DateStorageFormat dargestellt, die einen WdContentControlDateStorageFormat -Konstantwert erwartet.	Datumsauswahl 
<i>Katalog</i>	Legt den Katalog fest, aus welchem die aufzulistenden Bausteine zu holen sind. Wird im Objektmodell durch die Eigenschaft BuildingBlockCategory dargestellt, die einen WdBuildingBlockTypes -Konstantwert erwartet.	Bausteinkatalog (Mehr zum Thema Bausteine lesen Sie in Kapitel 6) 

Tabelle 7.5 Eigenschaften der Inhaltssteuerelemente (Fortsetzung)

Eigenschaft	Beschreibung	Steht den Inhaltssteuer- elementen zur Verfügung
<i>Kategorie</i>	Legt die Kategorie fest, aus welchem die aufzulistenden Bausteine zu holen sind. Wird im Objektmodell durch die Eigenschaft BuildingBlockType dargestellt.	Bausteinatalog 
<i>Aktiviert-Symbol</i> sowie <i>Deaktiviert-Symbol</i>	Legt das gewünschte Symbol für ein aktiviertes bzw. unaktiviertes Kontrollkästchen fest. (Nur in Word 2010.)	Kontrollkästchen 

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. Nachfolgend 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 Inhaltssteuerelemente eingegeben, die sich in einer frei auf der Seite positionierten Tabelle befinden (Abbildung 7.23).

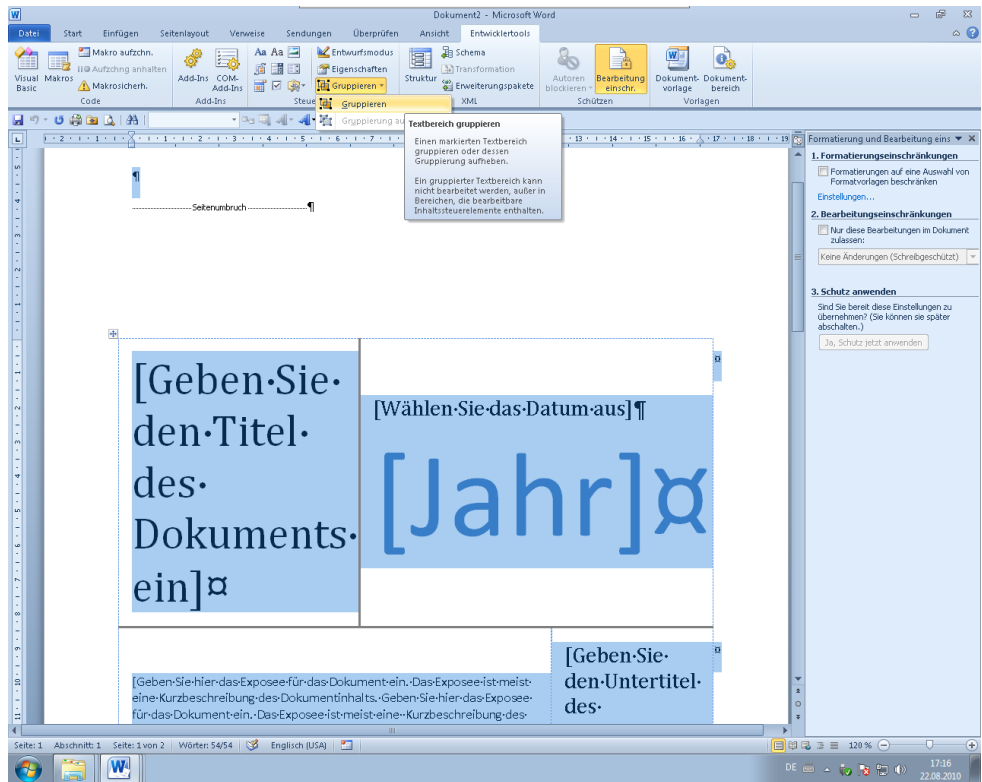
Abbildg. 7.23 Deckblatt mit Inhaltssteuerelementen in einer Tabelle

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, gehen Sie folgendermaßen vor:

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 markieren Sie mit der Maus die Absatzmarke sowie den Seitenumbruch zuoberst auf der Seite.
3. Klicken Sie auf die Schaltfläche *Gruppieren* und wählen Sie im Dropdownmenü den gleichnamigen Eintrag aus (Abbildung 7.24).

Abbildg. 7.24

Die zu schützenden Bereiche markieren und gruppieren



4. Klicken Sie auf die Schaltfläche *Entwurfsmodus* und positionieren Sie anschließend die Einfüge-
marke neben dem Element *Gruppe*, wie in Abbildung 7.25 ersichtlich.

Abbildg. 7.25

Das *Gruppe*-Inhaltssteuerelement hat den Fokus



5. Klicken Sie auf die Schaltfläche *Eigenschaften* und aktivieren Sie das Kontrollkästchen *Das
Inhaltssteuerelement kann nicht gelöscht werden*. (Dieses soll das einzige Steuerelement im Dia-
logfeld sein. Wenn Sie etwas anderes sehen, ist der Fokus nicht in dem *Gruppe*-Element.)
6. Schalten Sie den Entwurfsmodus aus, indem 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 Dokumentenebene, 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 geben Sie, sofern gewünscht, ein Kennwort ein.

HINWEIS

Die Beispielvorgabe *Bsp07_01_CC.dotm* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap07*. Ein Beispieldokument *Bsp07_01_CC_Fertig.docx* mit dem beschriebenen Schutz befindet sich im gleichen Ordner.

Falls Sie schon das Kapitel 22 gelesen haben oder bereits mit XML in Word vertraut sind, sind Ihnen wahrscheinlich die 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:sdtContent* eröffnet. Die Verschachtelung des Rich-Text-Inhaltssteuerelements »Titel« in der 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 Inhaltssteuerelement kann nicht gelöscht werden* entspricht. Das Attribut *id* wird von Word zugewiesen und identifiziert ein Inhaltssteuerelement eindeutig.

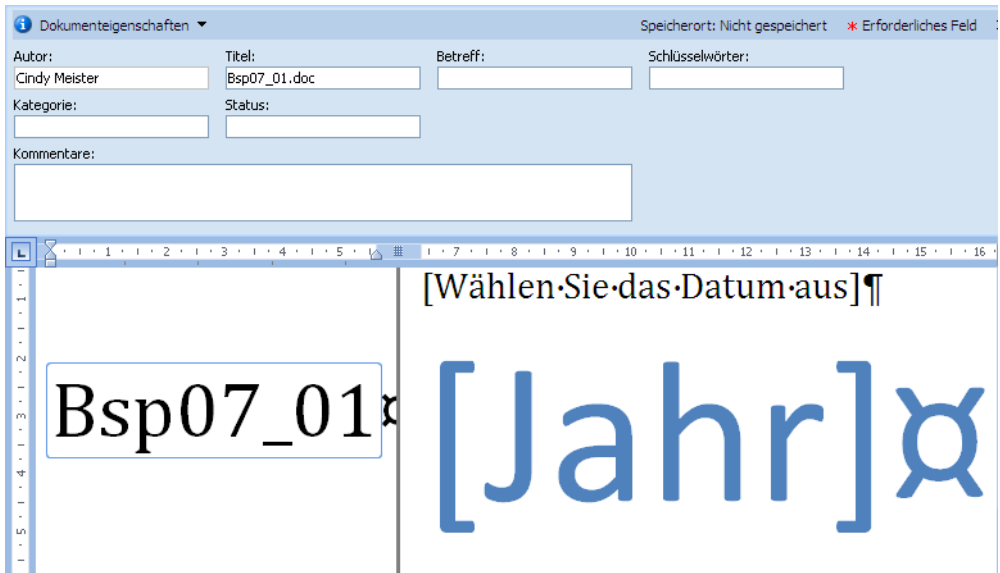
Abbildg. 7.26 Das WordProcessingML einiger Inhaltssteuerelemente der Beispielvorlage

```
- <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:databindings`. 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 selbst testen. Geben Sie in das Inhaltssteuerelement für den Titel einen Text ein. Gehen Sie zu *Datei/Informationen*; in der rechten Spalte auf die Schaltfläche *Eigenschaften* klicken und aus der Liste *Dokumentbereich anzeigen* wählen (in Word 2007 über *Office/Vorbereiten/Eigenschaften*). 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« ab Seite 415 beschrieben.

Die manuelle Methode ist ähnlich der in Kapitel 16 für das Menüband 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« (das auch in Word 2010 anstandslos funktioniert). 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/modonovan/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>

- Building Document Generation Systems from Templates with Word 2010 and Word 2007 [http://msdn.microsoft.com/en-us/library/ff433638\(offic.14\).aspx](http://msdn.microsoft.com/en-us/library/ff433638(offic.14).aspx)
- Word 2007 Content Control Toolkit (.NET Framework 2.0 muss auf dem Rechner installiert sein) <http://www.codeplex.com/Wiki/View.aspx?ProjectName=dbe>

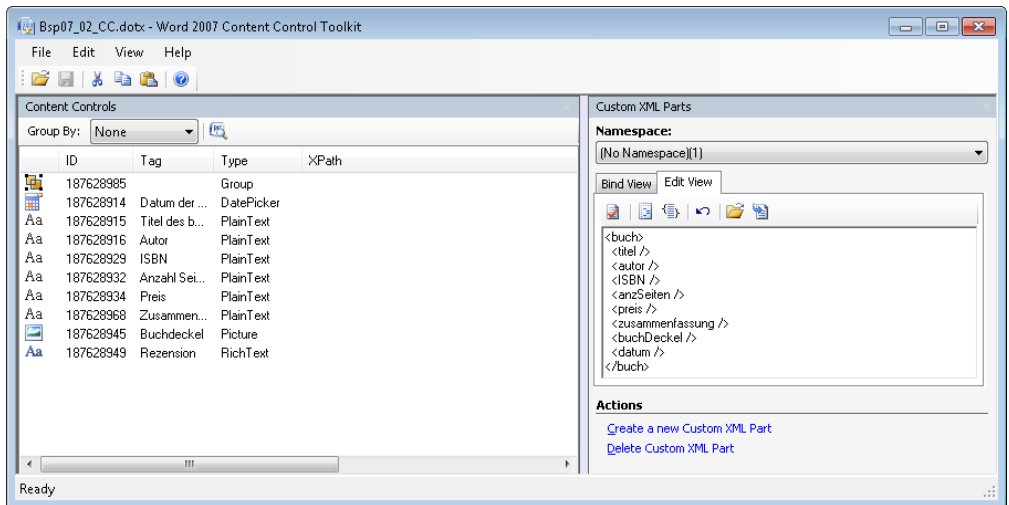
Abbildg. 7.28 Dokumentinhalt in Inhaltssteuerelementen mit einem benutzerdefinierten XML-Teil verbinden



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).

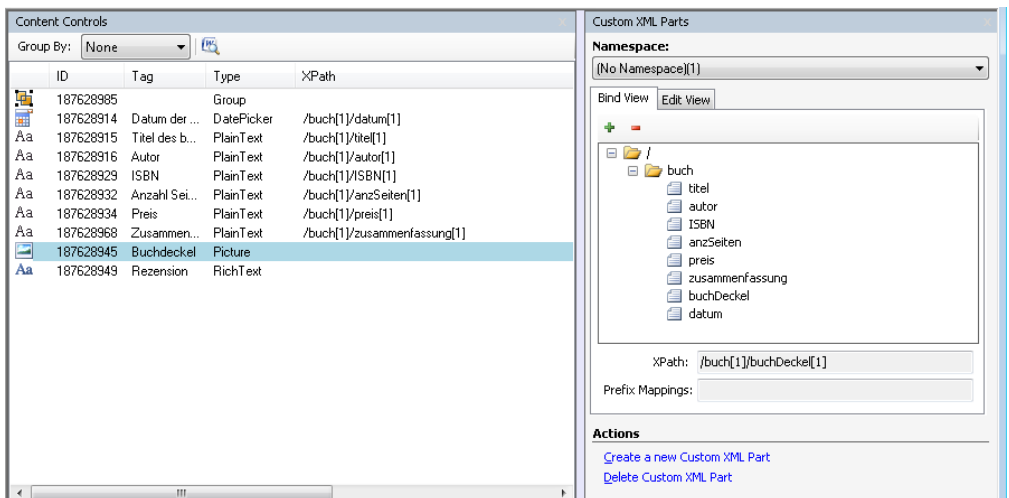
Da das Dokument noch keinen benutzerdefinierten XML-Teil enthält, klicken Sie auf den Link *Click here to create new one* im rechten Fenster. Ein XML-Teil namens *CustomXML* wird im Dokument mit allen benötigten Verweisen erstellt. Klicken Sie nun auf den Registerreiter *Edit View* (Abbildung 7.29). 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 ist Ihnen vielleicht aufgefallen, 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 einiger der Elemente. Speichern, schließen und öffnen Sie das Dokument in Word erneut, um das Ergebnis zu sehen.

CD-ROM

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, sodass der Benutzer die Daten nur einmal eingeben muss.

Die *Ref*-Feldfunktion funktioniert mit Inhaltssteuerelementen nicht zufriedenstellend, 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. Kombinationsfelds 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 und
- die Ereignisse für Inhaltssteuerelemente einsetzen.

Weiter werden in diesem Abschnitt die folgenden Themen vorgestellt:

- Das Navigieren zwischen Inhaltssteuerelementen
- Der Umgang mit Namensräumen in XML-Teilen

Die Briefvorlage kurz vorgestellt



Die Firma Northwind stellt eine Vorlage mit Briefkopf zur Verfügung. Für Word 2007/2010 wurde diese auf Vordermann gebracht. Sie wurde mit Inhaltssteuerelementen ausgestattet und zwar für: den Absendeort, 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 ein Angebot, die ihrerseits Inhaltssteuerelemente für die Produktinformationen enthält.

Abbildg. 7.31 Ein Firmenbrief mit zu einem XML-Teil verbundenen Inhaltssteuerelementen

Northwind-GmbH
Abteilung: Verkauf

→

Kirkland, den 23. August 2010

Alfreds Futterkiste
Maria Anders
Obere Str. 57
12209 Berlin
Deutschland

Brief an Kundschaft erstellen

Absender: Nancy Davolio

Kunde: Alfreds Futterkiste, Maria Ander

Brieftyp wählen
☐ Allgemein
☒ Offerte
☐ Bestellung

☐ deutsch
☒ englisch

OK

Abbrechen

Produkt

Lagerbestand

☒ Chai 39
☒ Chang 17
☒ Aniseed Syrup 13
☐ Chef Anton's Cajun Seasoning 53
☐ Chef Anton's Gumbo Mix 0

Betreff: Ihre Bestellung vom 19. August 2010

Dear Ms. Maria Anders

Art-Nr	Beschreibung	Anzahl	Stückpreis	Preis
1	Chai	1	9,00	9,00
2	Chang	1	9,50	9,50
3	Aniseed-Syrup	1	5,00	5,00
				23,50

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 ursprünglichen Angaben für die Inhaltssteuerelemente werden über ein Formular (UserForm) vorgenommen. 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. Der Benutzer kann die Art des Briefes wählen. Falls er die Option *Offerte* anklickt, wird rechts ein Katalog eingeblendet, aus dem die Produkte auszuwählen sind.

Die Produktliste wird erst beim Anklicken des Optionsfelds erstellt. Mithilfe der ADO-Methode *Save* und seiner Option *PersistFormat:=adPersistXML* werden diese Informationen lokal in *xml*-Dateien für den späteren Zugriff gespeichert. Dies reduziert den Verkehr mit der Datenbank und über das Netzwerk und beschleunigt dadurch auch die Erstellung des Briefs.

Bei Bestätigung des Formulars werden die Einstellungen ausgewertet. In den Speicher wird eine *xml*-Datei mit der passenden Datenstruktur für die Art Brief (allgemein oder Offerte) mittels MSXML-DOM geladen. Die Knotenpunkte werden mit den benötigten Formular-Daten gefüllt, die teilweise aus den vorher gespeicherten *xml*-Dateien gelesen werden. Anschließend wird diese 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 für die Anrede und Grußformel gefüllt und dann anschließend alle Inhaltssteuerelemente im Dokument gruppiert sowie geschützt. Die Listen für die Kombinationsfelder 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 und 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 Beispieldvorlage.

CD-ROM

Die Beispieldvorlage *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. Oder öffnen Sie das Dokument im »Word 2007 Content Control Toolkit« und wechseln in die *Edit*-Ansicht.

Listing 7.23 Der XML-Code des im Dokument gespeicherten XML-Teils

```
<?xml version="1.0"?>
<bestellung>
  <absender ort="Kirkland" 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>Ihre Bestellung vom 19. August 2010</betreff>
  <datum>23. August 2010</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>
```

Der Anfangspunkt stellt die Prozedur *Brief_Schreiben*, die die Umgebung vorbereitet und das Formular einblendet. Nach Bestätigung des Formulars werden die nächsten wichtigen Handlungen durch die Prozedur *AllgemeineBriefInfo* in Listing 7.24 ausgeführt. Diese ruft ihrerseits die Prozedur *OfferteXMLErstellen* auf, die mithilfe des MSXML-DOM den XML-Code des Listing 7.23 zusammenstellt und als Zeichenkette zurückgibt. Damit wird der XML-Teil (CustomXMLPart) im Dokument erstellt:

```
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
```

Inhaltssteuerelemente im Dokument ansprechen

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. 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. Dies macht das Ansprechen nur bestimmter Inhaltssteuerelemente etwas kompliziert.

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.

Inhaltssteuerelement über den Indexwert ansprechen

Folgendes Beispiel zeigt, wie das Inhaltssteuerelement mit dem Indexwert identifiziert wird. Um Inhalt in ein beliebiges, nicht geschütztes Inhaltssteuerelement zu schreiben, wird ganz einfach die Range-Eigenschaft des Inhaltssteuerelements angesprochen.

```
Dim cc As Word.ContentControl
Set cc = ActiveDocument.ContentControls(1)
cc.Range.Text = "Text in einem Inhaltssteuerelement."
```

HINWEIS Ein Beispiel mit der Eigenschaft ID als Indexwert finden Sie im Abschnitt »Namensräume« ab Seite 428.

Inhaltssteuerelement mit einer Textmarke ansprechen

Im vorliegenden Beispiel wird in der Prozedur *OfferteSchreiben* ein Baustein mit der Produkttabelle in das Rich-Text-Inhaltssteuerelement eingefügt. Um das neu eingefügte Steuerelement direkter ansprechen zu können, wurde ihm vor Erstellung des Bausteins 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)
```

Inhaltssteuerelement über die Titel-Eigenschaft ansprechen

Eigentlich würden wir erwarten, ein Inhaltssteuerelement durch seine Eigenschaft `Titel` ansprechen zu können. Es wurde jedoch entschieden, dass diese Eigenschaft ein Inhaltssteuerelement nicht eindeutig identifizieren würde. 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.

Liste eines Kombinationsfelds bzw. einer Dropdownliste füllen

In der Beispielvorlage kann der Benutzer »Anrede« und »Grußformel« eingeben oder Einträge aus den Kombinationsfeldlisten wählen. Da Northwind 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 im UserForm ausgewählte Sprache.

```
Set domNodes = domDoc.SelectNodes("//" & sprache & "/" & element)
```

Anschließend werden alle Inhaltssteuerelemente mit der gleichen `Titel`-Eigenschaft (*Anrede*, beispielsweise) mittels der Methode `SelectContentControlsByTitle` durchschleift und der Inhalt der Knotenpunkte aus der *xml*-Datei der Kombinationsfeldliste zugefügt.

Listing 7.25 Die Kombinationsfeldlisten 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
```


Listing 7.25 Die Kombinationsfeldlisten mit Einträgen aus *xml*-Dateien bestücken (Fortsetzung)

```

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
Exit For
Next
Set domNode = Nothing
Set domNodes = Nothing
Set domDoc = Nothing
End Sub

```

Im Beispiel sind diese Steuerelemente nicht mit einem CustomXMLPart verbunden, weshalb jedes einzelne angesprochen werden muss. Wären sie mit einem Knotenpunkt im CustomXMLPart verbunden, müsste die Information nur in den verbundenen Knotenpunkt geschrieben werden, und sie würde dann im Dokument erscheinen.

Eine Liste wiederholter Inhaltssteuerelemente erstellen und sie mit einem XML-Teil verbinden (1:n)

In diesem Abschnitt wird das für Word berüchtigte – 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).SelectSingleNode("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 zu summieren.

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

    '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)

    'Gleichzeitig durch die Produkt-Knotenpunkte im CustomXMLPart
    'und die Tabellenzeilen, (die fortlaufend kopiert an das Ende der Tabelle gesetzt werden)
    'mit den dazu gehörenden Inhaltssteuerelementen schleifen
    'und die beiden verknüpfen.
    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, _
```

Listing 7.26 Alle ausgewählten Produkte in der Produkttabelle auflisten

```
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). Anschließend muss dieser Bereich zuerst markiert werden, obwohl das Objektmodell den Anschein erweckt, dass das Range-Argument ein Objekt des Datentyps Word.Range akzeptieren würde.

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. 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. 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 das Argument `InUndoRedo`, das festhält, ob das Ereignis durch den Befehl *Rückgängig machen* oder *Wiederholen* ausgelöst wurde. Sie können bei Bedarf das Argument prüfen und entsprechend handeln.

HINWEIS

Inhaltssteuerelemente werden auch durch Kopieren und Verschieben gelöscht und erneut erstellt, was ebenfalls dieses Ereignis auslöst. Eigentlich ist das der Fall für alles in einem Word-Dokument, nur ist uns das nicht bewusst, weil andere Objekte kein ähnliches Ereignis haben.

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 Inhaltssteuerelements 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.

HINWEIS

Das Ereignis `ContentControlBeforeContentUpdate` wird nur ausgelöst, wenn die Information im Inhaltssteuerelement sich ändert und nicht beim Schreiben der Information in den `CustomXMLPart`.

Das Ereignis `ContentControlBeforeStoreUpdate` wird ausgelöst für jedes Zeichen, das in das Inhaltssteuerelement eingegeben wird.

Die letzten zwei Ereignisse werden beim Eintreten bzw. beim Verlassen des Inhaltssteuerelements ausgelöst. Das Argument `Cancel` des Ereignisses `ContentControlOnExit` ermöglicht den Abbruch der Handlung. Damit kann der Benutzer dazu gezwungen werden, irgendeine Bedingung zu erfüllen, bevor er weiter gehen darf. Das Listing 7.27 veranschaulicht dieses Ereignis. `ContentControlBeforeStoreUpdate` findet logischerweise vor `ContentControlOnExit` statt.

HINWEIS

Während dieses Ereignisses kann das Inhaltssteuerelement nicht geändert werden.

In der Beispielvorlage für den Geschäftsbrief interessieren nur die Inhaltssteuerelemente in der Produkttabellenspalte »Anzahl«, die alle »Anzahl« als `Title`-Eigenschaft haben. Der Inhalt eines solchen Inhaltssteuerelements 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 Spalte aktualisiert und der Fokus verlässt das Inhaltssteuerelement.

Listing 7.27 Die Eingabe beim Verlassen 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 'Die Spalte mit den Berechnungen
            Dim cel As Word.Cell 'Die jeweilige Zelle in der Spalte
            Dim grpCC As Word.ContentControl 'Das gruppierende Inhaltssteuerelement
            Dim doc As Word.Document 'Das Word-Dokument
            Dim rngGroup As Word.Range 'Der Bereich der Gruppierung
            Dim rngSel As Word.Range

            'Der Bereich, der nach Verlassung des Inhaltssteuerelements
            'markiert wird, muss festgehalten werden,
            'sodass die Markierung wieder hergestellt werden kann.
            Set rngSel = Selection.Range
            Set doc = ContentControl.Range.Document
            'Um die Feldfunktionen aktualisieren zu können,
            'muss der Schutz durch das gruppierende Inhaltssteuerelement
            'aufgehoben werden.
            Set grpCC = ContentControl.ParentContentControl
            'Diesen Bereich wird aber zuerst festgehalten.
            Set rngGroup = grpCC.Range
            grpCC.LockContentControl = False
            grpCC.Ungroup
            Set col = ContentControl.Range.Tables(1).Columns(5)
            For Each cel In col.Cells
                If cel.Range.Fields.Count > 0 Then
                    cel.Range.Fields.Update
                End If
            Next
            'Die Gruppierung wieder herstellen.
            rngGroup.MoveEnd wdCharacter, 2
            rngGroup.Select
            Set grpCC = doc.ContentControls.Add(Type:=wdContentControlGroup, Range:=rngGroup)
            grpCC.LockContentControl = True
            'Die vom Benutzer ursprüngliche Markierung wieder herstellen.
            rngSel.Select
        End If
    Case Else
    End Select
End Sub

```

ACHTUNG Solange der Entwurfsmodus eingeschaltet ist, werden die Inhaltssteuerelementereignisse nicht ausgelöst.

Ereignisse von CustomXMLPart-Objekten

Die Ereignisse für Inhaltssteuerelemente werden durch Handlungen auf der Dokumentoberfläche ausgelöst. Gelegentlich würde der Entwickler lieber auf Aktionen reagieren, die in einem CustomXMLPart stattfinden bzw. den Status eines CustomXMLParts im Dokument beeinflussen.

Dafür gibt es je drei Ereignisse für das Objekt CustomXMLPart sowie die Auflistung CustomXMLParts:

- NodeAfterDelete (ByVal OldNode As Office.CustomXMLNode, ByVal OldParentNode As Office.CustomXMLNode, ByVal OldNextSibling As Office.CustomXMLNode, ByVal InUndoRedo As Boolean)
- NodeAfterInsert (ByVal NewNode As Office.CustomXMLNode, ByVal InUndoRedo As Boolean)
- NodeAfterReplace (ByVal OldNode As Office.CustomXMLNode, ByVal NewNode As Office.CustomXMLNode, ByVal InUndoRedo As Boolean)
- PartAfterAdd (ByVal NewPart As Office.CustomXMLPart)
- PartAfterLoad (ByVal Part As Office.CustomXMLPart)
- PartBeforeDelete (ByVal OldPart As Office.CustomXMLPart)

Die drei Ereignisse für das CustomXMLPart-Objekt werden durch Änderung eines Knotenpunkts ausgelöst: wenn ein Knotenpunkt gelöscht, hinzugefügt bzw. dessen Inhalt geändert wurde.

HINWEIS

Der Datentyp des Knotenpunkts in einem CustomXMLPart ist Office.CustomXMLNode. Wie das Objekt CustomXMLPart sowie die Auflistung CustomXMLParts ist das Objekt Teil des Objektmodells von Office. Auch Dokumente anderer Office-Anwendungen, wie Excel oder PowerPoint, können CustomXMLParts enthalten. Nur können diese nicht mit der Dokumentoberfläche verknüpft werden, sodass der Inhalt automatisch im Dokument erscheint.

Die drei Ereignisse, die sich auf CustomXMLPart-Objekte beziehen, werden durch Hinzufügen, Laden bzw. Löschen des Objekts ausgelöst. Das Hinzufügen eines CustomXMLParts löst auch das PartAfterLoad-Ereignis aus.

Im Gegensatz zu den Ereignissen der Inhaltssteuerelemente, die Teil des Dokument-Objekts sind und sich im ThisDocument-Modul befinden, werden Ereignisse für CustomXMLParts auf der Anwendungsebene definiert. Wer sie im VBA einsetzen möchte, muss seinem Projekt ein Klassenmodul hinzufügen. Am Anfang des Moduls werden die Objekte deklariert, deren Ereignisse abgefangen werden sollen. Neben den Ereignisprozeduren braucht es noch eine Prozedur, die den Objektvariablen die zu überwachenden Objekte zuweist.

In Listing 7.30 werden die CustomXMLParts-Auflistung des gesamten, aktiven Dokuments der Variable cxps zugewiesen. Bevor der bestimmte CustomXMLPart der Variable cxp zugewiesen werden kann, muss sichergestellt werden, dass er vorhanden ist. In diesem vereinfachten Beispiel ist bekannt, dass der CustomXMLPart der vierte ist. Falls der CustomXMLPart in der Auflistung nicht vorhanden ist, tritt eine Fehlermeldung auf.

Listing 7.28 Inhalt eines Klassenmoduls für CustomXMLPart-Ereignisse namens CXP_Ereignisse

```
Private WithEvents cxp As CustomXMLPart
Private WithEvents cxps As CustomXMLParts

Sub EreignisObjekteZuweisen()
    Set cxps = ActiveDocument.CustomXMLParts
    If ActiveDocument.CustomXMLParts.Count > 3 Then
        Set cxp = ActiveDocument.CustomXMLParts(4)
```

Listing 7.28 Inhalt eines Klassenmoduls für *CustomXMLPart*-Ereignisse namens *CXP_Ereignisse* (Fortsetzung)

```
End If
End Sub

Private Sub cxp_NodeAfterReplace(ByVal OldNode As Office.CustomXMLNode, ByVal NewNode As
Office.CustomXMLNode, ByVal InUndoRedo As Boolean)
    MsgBox "Der Inhalt des Knotenpunktes " & OldNode.BaseName &
        " wurde geändert von " & OldNode.Text & " auf " & NewNode.Text
End Sub

Private Sub cxps_PartAfterAdd(ByVal NewPart As Office.CustomXMLPart)
    MsgBox "Ein neuer CustomXMLPart mit folgendem XML wurde in dem Dokument hinzugefügt: " _
        & NewPart.XML
End Sub

Private Sub cxps_PartAfterLoad(ByVal Part As Office.CustomXMLPart)
    MsgBox "Ein CustomXMLPart mit folgendem XML wurde in dem Dokument geladen: " _
        & Part.XML
End Sub
```

Damit ist es aber noch nicht getan. Es ist nicht möglich, eine Prozedur in einem Klassenmodell direkt auszuführen. Die Klasse muss, wie ausführlich im Kapitel 8 beschrieben, zuerst instanziiert werden, wie Listing 7.29 veranschaulicht. Zuerst wird auf Modulebene eine Variable für die Klasse deklariert. In der Prozedur *StartCXPEreignisse* wird diese instanziiert, dann die Prozedur *EreignisObjekteZuweisen* im Klassenmodul aufgerufen.

Listing 7.29 Die Klasse instanzieren und die Ereignisse initiieren





```
Public CXP4 As CXP_Ereignisse

'Ereignisse für CustomXMLPart(s) starten
Sub StartCXPEreignisse()
    Set CXP4 = New CXP_Ereignisse
    CXP4.EreignisObjekteZuweisen
End Sub
```

CD-ROM

Das Beispieldokument *Bsp07_04_CC.docm* enthält Ereignisstrukturen sowie einen *CustomXMLPart*, um das Verhalten der hier vorgestellten Ereignisse 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 Dokumente mit Inhaltssteuerelementen, solange sie des Basistyps »Nur-Text« sind, die ein Tab-Zeichen standardmäßig nicht akzeptieren. Ein 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, **Strg** + **↵**.

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 der Tastatur zu lassen, drängt sich auf. Folgend in Listing 7.30 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 Tastenkombination **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 7.30 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
```

CD-ROM

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-Knotenpunkts. Die Knotenpunkte mehrerer XML-Dateien oder -Teile können die gleiche Bezeichnung haben. Es ist wichtig, dazwischen unterschei-

den 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 alle vorgeschlagenen Personen werden vom Benutzer in Inhaltssteuerelemente eingegeben. Diese sind mit einem CustomXMLPart verbunden, wie derjenige in Abbildung 7.33. Die gleichen Elementennamen »name« und »telefon« sind für »kunde« sowie »vermittelt« definiert. Es gilt, klar dazwischen zu unterscheiden, sodass beispielsweise mit wenig Aufwand alle vorgeschlagenen Namen gelesen werden können.

Die Beispielvorgabe enthält die zwei Tabellen, mit je zwei Zeilen, bestückt mit je zwei Inhaltssteuerelementen: eines für den Namen, das andere für die Telefonnummer. Während der Vorbereitung wurden die Inhaltssteuerelemente über die Prozedur in Listing 7.31 mit dem CustomXMLPart verbunden. Der Benutzer erstellt ein auf dieser Vorlage basierendes Dokument und füllt die Listen aus.

Abbildung. 7.32 Die Inhaltssteuerelemente in den Tabellen sind mit verschiedenen Knotenpunkten gleichen Namens verbunden

In diesem Beispiel werden XML-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 zu zufügen. ¶

Hier klicken, um eine Liste der vorgeschlagenen Personen einzublenden. ¶

Microsoft Word

Folgende Personen wurden vorgeschlagen:

Max Müller

Doris Schmid

Georg Werner

OK

Abbildung. 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>
```

Im XML-Code der Abbildung 7.33 sind drei Namensräume (xmlns) definiert. Der erste gilt für das gesamte Dokument und ist allen Knotenpunkten zugeteilt, die mit keinem anderen Namensraum verbunden sind (in diesem Fall nur *vorschlag*). Der zweite Namensraum ist im Knotenpunkt *kunden* angegeben und gilt für alle darin verschachtelten Knotenpunkte (*name* und *telefon*). Der dritte ist im Knotenpunkt *vermittelte* angegeben und gilt für seine Unterknotenpunkte (*person*, *name* und *telefon*). Das vorliegende Beispiel veranschaulicht, wie diese Knotenpunkte mithilfe des Namensraums direkt angesprochen werden.

Es wäre umständlich, immer den gesamten Namensraum angeben zu müssen, wenn ein Knotenpunkt angesprochen wird. Deshalb erstellt Office-XML automatisch für jeden Namensraum ein sogenanntes Namensraumpräfix – eine Art Abkürzung oder ID. 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)
XPath="//" & pfxKunde & ":name"
```

HINWEIS

Bereits im XML-Code vorhandene Namensraumpräfixe werden von Office *nicht* übernommen.

Die letzten vier Befehlszeilen in Listing 7.31 veranschaulichen, wie mithilfe 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.

Listing 7.31 Inhaltssteuerelemente mit Knotenpunkten verbinden, die einem Namensraum zugeordnet sind

```
Private Const nsVorschlag As String = _
    "http://www.Kap07.com/Beispiel07_06_cc/Vorschlag"
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"

'Muss nur einmal, in der Vorlage, ausgeführt werden, um den CustomXMLPart zu erstellen.
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)
```

Listing 7.31 Inhaltssteuerelemente mit Knotenpunkten verbinden, die einem Namensraum zugeordnet sind (Fortsetzung)

```

pfxKunde = cxpVorschlag.NamespaceManager.LookupPrefix(nsKunde)
pfxVermittelte = cxpVorschlag.NamespaceManager.LookupPrefix(nsVermittelte)

doc.ContentControls("58468977").XMLMapping.SetMapping _
    XPath="//" & pfxKunde & ":name", Source=cpvVorschlag      'Kundenname
doc.ContentControls("58468980").XMLMapping.SetMapping _
    XPath="//" & pfxKunde & ":telefon", Source=cpvVorschlag  'KundenTelefon
doc.ContentControls("58468981").XMLMapping.SetMapping XPath="//" &
    pfxVermittelte & ":name", Source=cpvVorschlag      'Name des 1. Vermittelten
doc.ContentControls("58468982").XMLMapping.SetMapping XPath="//" &
    pfxVermittelte & ":telefon", Source=cpvVorschlag    'Telefon des ersten Vermittelten
End Sub

```

HINWEIS Das Listing 7.31 veranschaulicht zudem den Gebrauch der ID-Eigenschaft als Indexwert für die Identifikation eines Inhaltssteuerelements. Bitte beachten Sie, dass diese als Datentyp String anzugeben ist.

Bei seiner Arbeit im Dokument muss der Benutzer der zweiten Tabelle neue Zeilen, samt Inhaltssteuerelemente, hinzuzufügen. Dies wird ihm durch eine *MacroButton*-Feldfunktion unterhalb der Tabelle erleichtert. Beim darauf Klicken wird die Prozedur *NeuesPersonal* in Listing 7.32 angestoßen. Nachdem die neue Tabellenzeile mit Inhaltssteuerelementen eingefügt wurde, wird die Prozedur *XmlPersonalNodeErstellen* aufgerufen, die dem CustomXMLPart einen Satz *personal*-Knotenpunkte unter dem Element *vermittelte* zufügt.

Es ist wichtig, diese Handlung im korrekten CustomXMLPart vorzunehmen. Ein CustomXMLPart akzeptiert neben einer Ganzzahl, die der Reihenfolge in der Auflistung entspricht, auch einen Namensraum als gültigen Indexwert.

```
Set cpx = doc.CustomXMLParts(nsVorschlag)
```

Danach 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 7.32 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

    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 cxp As Office.CustomXMLPart
    Dim nodePerson As Office.CustomXMLNode
    Dim rootNode As Office.CustomXMLNode
    Dim ns As String
    Dim anzPersonNodes As Long

    Set cxp = doc.CustomXMLParts(nsVorschlag)
    ns = cxp.NamespaceManager.LookupPrefix(nsVermittelte)
    Set rootNode = cxp.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]")
    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 7.33 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 7.33 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

    Set cpx = ActiveDocument.CustomXMLParts(nsVorschlag)
    ns = cpx.NamespaceManager.LookupPrefix(nsVermittelte)

    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
```

CD-ROM 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 mit XML-Werkzeugen zu bearbeiten (mehr dazu im Kapitel 22). Dank des Kompatibilitätpacks können alle Versionen von Word, bis einschließlich der Version 2000, solche Dokument öffnen, bearbeiten und speichern. Besonders, wenn mit den neuen Inhaltssteuerelementen, wie im Abschnitt »Inhaltssteuerelemente im Objektmodell« ab Seite 415 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.

Eine relativ schnörkellose Methode ist, Daten automatisiert in das Dokument einzufügen, wie im Abschnitt »Zielscheibe Textmarke: das *Bookmark*-Objekt« ab Seite 342 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 den Dateien im Ordner `\Beilagen\Zusatzmaterial Seriendruck` auf der CD-ROM zum Buch 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 seit Word 2002 sind:

- Zu den unterstützten Verbindungsmethoden DDE und ODBC gesellte sich OLE DB
- Neue Sicherheitsmaßnahmen erschwerten das Öffnen von Seriendruck-Hauptdokumenten
- Der eingeführte Aufgabenbereich lässt sich teilweise steuern
- Neue »MailMerge«-Ereignisse ermöglichten einen Eingriff in das Zusammenführen des Seriendrucks (siehe Kapitel 8)
- Ab Word 2007 wird DDE als Verbindungsmethode weiter verdrängt. DDE erkennt keine Datenquellen, welche die neuen Dateieindungen für das XML-Dateiformat – `.xlsx` (Excel) sowie `.accdb` (Access) – tragen. DDE kann die Dateien jedoch verwenden, wenn diese umbenannt und mit den alten Dateieindungen `.xls` bzw. `.mdb` versehen werden.
- Ab Word 2010 steht in der Benutzerschnittstelle die Option, getrennte Steuersatzdateien für Textdatenquellen zu nutzen, nicht mehr zur Verfügung. Somit wäre es ratsam, diese Technik zu meiden, da die weitere Unterstützung durch die Word-Anwendung in zukünftigen Versionen nicht garantiert ist.

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) 10;
```

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 Konstantenwerte 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
Folgende Konstantenwerte gelten nur für Word 2002 und später. Sie beziehen sich auf die Kontrollkästchen-Einstellungen im Dialogfeld <i>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. Sie wird zudem unter bestimmten Umständen in späteren Versionen nicht erkannt. In diesen Fällen muss der letzte Datensatz angewählt und ActiveRecord abgefragt werden, um die Anzahl Datensätze zu ermitteln.

HINWEIS

Das Navigieren der Datensätze wird im Beispiel zu den Seriendruckereignissen in Kapitel 8 veranschaulicht.



Mit Einführung des Dialogfelds *Seriendruckempfänger* in Word 2002 erhielt der Anwender mehr Flexibilität (Abbildung 7.34). Dadurch wurde jedoch die Navigierung und Auswertung von Datensätzen durch das Objektmodell komplizierter.

Abbildg. 7.34 Das Dialogfeld *Seriendruckempfänger* ermöglicht es dem Anwender, einzelne Datensätze vom Seriendruck auszuschließen

Datenquelle		Nachname	Vorname	Anrede	Ort	PLZ	Region
Nordwind.mdb	<input checked="" type="checkbox"/>	Davolio	Nancy	Frau	Seattle	98122	WA
Nordwind.mdb	<input type="checkbox"/>	Fuller	Andrew	Herr	Tacoma	98401	WA
Nordwind.mdb	<input checked="" type="checkbox"/>	Leverling	Janet	Frau	Kirkland	98033	WA
Nordwind.mdb	<input checked="" type="checkbox"/>	Peacock	Margaret	Frau	Redmond	98052	WA
Nordwind.mdb	<input type="checkbox"/>	Buchanan	Steven	Herr	London	SW1 8JR	
Nordwind.mdb	<input checked="" type="checkbox"/>	Suyama	Michael	Herr	London	EC2 7JR	
Nordwind.mdb	<input checked="" type="checkbox"/>	King	Robert	Dr.	London	RG1 9SP	
Nordwind.mdb	<input type="checkbox"/>	Callahan	Laura	Frau	Seattle	98105	WA
Nordwind.mdb	<input checked="" type="checkbox"/>	Dodsworth	Anne	Frau	London	WG2 7LT	

Datenquelle: Nordwind.mdb

Empfängerliste verfeinern

- [Sortieren...](#)
- [Filtern...](#)
- [Duplikate suchen...](#)
- [Empfänger suchen...](#)
- [Adressen überprüfen...](#)

Bearbeiten... Aktualisieren OK

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 Datensätzen 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 7.34 und Abbildung 7.35 ersichtlich.

Listing 7.34 Durch alle Datensätze Schleifen und feststellen, ob sie ausgeschlossen sind

```

Sub DurchDatensätzeSchleifen()
    Dim lAnzahlDS As Long
    Dim lZaehler As Long
    Dim lAnzahlAktiverDS As Long
    Dim ds As Word.MailMergeDataSource

    If ActiveDocument.MailMerge.MainDocumentType = wdNotAMergeDocument _
        Then Exit Sub
    Set ds = ActiveDocument.MailMerge.DataSource
    lAnzahlDS = ds.RecordCount
    lAnzahlAktiverDS = 0
    ds.ActiveRecord = wdFirstRecord
    For lZaehler = 1 To lAnzahlDS
        Debug.Print lZaehler, ds.ActiveRecord, ds.Included
        'Falls das Kontrollkästchen aktiviert ist, wird der Datensatz zusammengeführt
        If ds.Included Then
            lAnzahlAktiverDS = lAnzahlAktiverDS + 1
        End If
        'Den Laufzeitfehler 5852 vermeiden
        If ds.ActiveRecord <> wdLastDataSourceRecord Then
            ds.ActiveRecord = wdNextDataSourceRecord
        End If
    Next
    Debug.Print lAnzahlAktiverDS & " Datensätze werden zusammengeführt."
    'Den letzten sichtbaren Datensatz anwählen
    ds.ActiveRecord = wdLastRecord
End Sub

```

Abbildg. 7.35 Das Ergebnis von Listing 7.34 basiert auf den Einstellungen in Abbildung 7.34

Direktbereich		
1	1	Wahr
2	2	Falsch
3	3	Wahr
4	4	Wahr
5	5	Falsch
6	6	Wahr
7	7	Wahr
8	8	Falsch
9	9	Wahr
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.

CD-ROM

Die Beispieldatei *Bsp07_01_SD.docm* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap07*. Sie sollen sie mit der Tabelle *Personal* in der *Nordwind2007.accdb*-Datenbank im *\Datenbank*-Ordner verbinden.

Datensätze filtern

Das *Empfänger*-Dialogfeld bzw. die *Included*-Eigenschaft, ist eine Möglichkeit, 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. Klicken Sie auf den Link *Filtern* im Dialogfeld *Seriendruckempfänger*, um an diese Funktionalität in der Benutzerschnittstelle zu gelangen.

Eine mit der *QueryString*-Eigenschaft festgelegte Filtrierung ersetzt die mit der *OpenDataSource* Argumente *SQLStatement* und *SQLStatement1* (siehe »Wie *OpenDataSource* funktioniert«) festgelegte. Die SQL-Anweisung der *QueryString*-Eigenschaft darf höchstens 256 Zeichen betragen.

HINWEIS

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.accdb* 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 gefilterte Datensätze nicht im Dialogfeld *Empfänger*, werden von *RecordCount* nicht beachtet und können über *ActiveRecord* nicht angesprungen werden.

Um alle Datensätze der Datenquelle wieder verfügbar zu machen:

```
doc.MailMerge.DataSource.QueryString = "SELECT * FROM [Personal]"
```

Datensatz suchen



Die Methode *FindRecord* entspricht der Schaltfläche *Empfänger suchen* 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.

Die Problematik wurde in Word 2010 weiter verschärft, da die Methode auf jeden Fall den ersten Datensatz der Seriendruck-Datenquelle nicht »sieht«. Auch dieses Fehlverhalten muss berücksichtigt und umgangen werden.

Das Prinzip des Beispielcodes in Listing 7.35 funktioniert in allen Versionen von Word, nur muss die richtige Methode verwendet werden. Der gegenwärtige Datensatz wird in einer Variablen festgehalten, sodass 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 einige Versionen von Word Feldnamen so einfügen, müssen Sie unter Umständen zuerst die Feldcodes bearbeiten, um die Anführungszeichen zu entfernen.

Listing 7.35

Einen Datensatz suchen in Word

```
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 = (InStr(1, CStr(mm.DataSource.DataFields(strFeld).Value), strText) > 0)
    If Not bGefunden Then
        bGefunden = mm.DataSource.FindRecord(FindText:=strText, Field:=strFeld)
    End If
    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 Feldangaben 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.

CD-ROM Die Beispieldatei *Bsp07_01_SD.docm* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap07*. Sie sollen sie mit der Tabelle *Personal* in der *Nordwind2007.accdb*-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 zufriedenstellend. Das Verhalten kann durch Festlegung eines Registryeintrags ausgeschaltet werden. Dieser Vorgang ist im Knowledge Base-Artikel »Meldung "Das Öffnen dieses Dokuments wird den folgenden SQL-Befehl ausführen" beim Öffnen eines Word-Seriendruck-Hauptdokuments, das mit einer Datenquelle verknüpft ist« unter <http://support.microsoft.com/kb/825765/de> beschrieben.

Noch eine Änderung gibt es seit Word 2002: Es wird keine Seriendruckschnittstelle – weder der Assistent *Aufgabenbereich* noch die Registerkarte *Sendungen* – 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 7.36.

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 ersetzte in Word 2007 die Multifunktionsleiste die altbekannten Menü- und Symbolleisten früherer Versionen (in Word 2010 wurde die Multifunktionsleiste umbenannt in Menüband). Der Seriendruck wird nun 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.

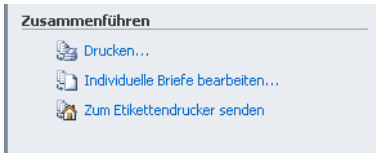
Der Inhalt des Aufgabenbereichs jedoch kann nur im sechsten und letzten Schritt beeinflusst werden: die Methode ShowSendToCustom ermöglicht die Hinzufügung eines zusätzlichen, vom Entwickler definierten Eintrags. Und das Ereignis MailMergeWizardSendToCustom wird beim Anklicken des Eintrags ausgeführt, 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. Falls Sie zudem die Befehle in der Registerkarte *Sendungen* des Menübands sperren möchten, müssen Sie das Menüband-XML der Dokumentvorlage entsprechend anpassen (Kapitel 16).

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 7.36 sowohl vom Document_New- als auch vom Document_Open-Ereignis aufgerufen.

Listing 7.36 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"
    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.

CD-ROM Die Beispieldatei *Bsp07_02_SD.dotm* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap07*. Sie ist mit der Tabelle *Personal* in der *Nordwind2007.accdb*-Datenbank im *\Datenbank*-Ordner verbunden.

Seriendruck zusammenführen



Ein Seriendruck kann in ein neues Dokument, direkt zum Drucker, zum Fax- oder zum E-Mail-Ver-sand 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 7.37.

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 deren Existenz 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 zum Menüband und die damit verbundenen Makros haben. Diese Verknüpfung ist in den Word-Versionen vor 2007 zu diesem Zeitpunkt jedoch nicht vollständig. Falls Sie diesem Problem begegnen, stellen Sie nach dem Zusammenführen die Verknüpfung zur Vorlage explizit wieder her:

```
ActiveDocument.AttachedTemplate = docSD_Hauptdokument.AttachedTemplate
```

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« ab Seite 380 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 7.37 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 einen 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 7.37 eingefügt.

Listing 7.37 Einen Umschlag als Seriendruck-Hauptdokument erstellen

```
Sub UmschlagSeriendruck()
    Dim doc As Word.Document
    Dim rng As Word.Range
    Dim strDatenPfad As String

    strDatenPfad = "..\Datenbank\Nordwind2007.accdb"
    Set doc = Documents.Add
    doc.PageSetup.PaperSize = wdPaperEnvelopeC5
    doc.PageSetup.Orientation = wdOrientLandscape

    Selection.Range.Text = "Absenderadresse kommt hier"
    Set rng = doc.Range
    rng.InsertAfter vbCr
    rng.Collapse wdCollapseEnd
    rng.Style = "Umschlagadresse"
    With doc.MailMerge
        .OpenDataSource Name:=strDatenPfad, _
            SQLStatement:="SELECT * FROM [Personal]"
        .MainDocumentType = wdEnvelopes
    End With
End Sub
```

Listing 7.37 Einen Umschlag als Seriendruck-Hauptdokument erstellen (Fortsetzung)

```

    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

```

CD-ROM Die Beispieldatei *Bsp07_03_SD.docm* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap07*.

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 Funktionalität *Alle Etiketten aktualisieren* fehlt. Stattdessen muss man den internen Word-Befehl über das WordBasic-Objektmodell aufrufen.

Wir legen deshalb Wert darauf, in Listing 7.38 als Beispiel die Technik zu veranschaulichen. Die Etikettenbezeichnung eines bestimmten Etiketts entnehmen Sie dem Feld *Bestellnummer* im Dialogfeld *Etiketten einrichten*, das über die Befehlsfolge *Sendungen/Erstellen/Beschriftungen* erreicht wird. Meistens wird nur die Nummer, und nicht die Beschreibung, gebraucht.

Falls Sie dem Benutzer die Möglichkeit bieten möchten, die Etikettenart selber auszuwählen, steht die neue Methode `Application.MailingLabel.LabelOptions` zur Verfügung. 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 Zeichenkettevariablen strAdresse festgelegt.

Als Datenquelle haben wir die Outlook-Kontaktliste gewählt und in der SQL-Anweisung die Felder, wo nötig, umbenannt, sodass 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 Etikettendokument 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.

Ab Word 2007 kann der Befehl über Application.CommandBars.ExecuteMso ausgeführt werden. Die Mso-Bezeichnung lautet: MailMergeUpdateLabels.

Listing 7.38

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 >><<" & vbCrLf & "\l 1031 \c 2 \e ""Deutschland"""
    strSQL = "Select Vorname, Nachname, Firma, [Position] as Anrede, " & _
        "[Straße] as Address1, Ort, Bundesland, [PLZ] as PostalCode, Land From [Kontakte]"

    'Die Etiketteneinteilung wird in ein neues Dokument erstellt
    Set doc = Application.MailingLabel.CreateNewDocument(Name:= strEtikettenname)
    'Etiketten werden immer als eine Tabelle aufgestellt
    'Erste Zeile der Etiketten oben (statt in der Mitte) ausrichten
    doc.Tables(1).Range.Cells.VerticalAlignment = wdCellAlignVerticalTop
    With doc.MailMerge
        'Dokument als Seriendruck-Hauptdokument bezeichnen
        .MainDocumentType = wdMailingLabels
    End With
End Sub
```

Listing 7.38 Seriendruck-Etiketten erstellen (Fortsetzung)

```
'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, 14
        'Ab 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 Addressblock, 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
```

CD-ROM Die Beispieldatei *Bsp07_04_SD.docm* 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])
```

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 wdOpenFormatAuto verwendet.
ConfirmConversions	* Optionaler Variant-Wert
ReadOnly	* Optionaler Variant-Wert
LinkToSource	Optionaler Variant-Wert. True , um die Abfrage-SQL-Anweisung der Connection - und SQLStatement -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 OpenDataSource -Methode fehl.
PasswordTemplate	* Optionaler Variant-Wert
Revert	* Optionaler Variant-Wert
WritePasswordDocument	Optionaler Variant-Wert. Falls ein Word-Dokument mit einem Kennwort zum Ändern gespeichert wurde, das Kennwort hier übergeben, um das Datenquellen-Dokument zu öffnen. Fehlt der Parameter oder wird nur eine leere Zeichenkette übergeben, blendet Word eine Eingabeaufforderung ein. Wird das falsche Kennwort übergeben, schlägt die OpenDataSource -Methode fehl.
WritePasswordTemplate	* Optionaler Variant-Wert
Connection	Optionaler Variant-Wert. Der Bereich, auf dem die Abfrage in den SQLStatement -Parametern ausgeführt wird. Wie der Bereich festzulegen ist, kommt auf die Datenverbindungsmethode an. Beispiele: Für Daten, die über eine ODBC-Verbindung eingebunden werden, ist ein »Connection String« mit gültigem DSN erforderlich. Für Excel-Daten, die mit einer DDE-Verbindung eingelesen werden, gibt man einen benannten Arbeitsblatt-Bereich ein. Eine DDE-Verbindung mit Access erfordert den Namen einer Tabelle oder Abfrage. Die Grundlagen für diesen Parameter werden am besten durch Aufzeichnung eines Makros ermittelt. Weitere Informationen folgen weiter unten.
SQLStatement	Optionaler Variant-Wert. Bestimmt eine Abfrage, um die Daten zu filtern.
SQLStatement1	Optionaler Variant-Wert. Falls die SQL-Anweisung länger als 255 Zeichen ist, kann sie mit diesem Parameter um zusätzliche 255 Zeichen erweitert werden.

Tabelle 7.7 Parameter der Methode *OpenDataSource* (Fortsetzung)

Parameter-Name	Bemerkungen
OpenExclusive (nicht verfügbar in Word 2000 und früher)	Optionaler Variant-Wert. Öffnet die Datenquelle angeblich »im exklusiven Modus«. Unsere Tests mit Access haben jedoch keine Wirkung gezeigt. Wird vom Makrorekorder nicht aufgezeichnet.
SubType (nicht verfügbar in Word 2000 und früher)	Optionaler Variant-Wert. In der VBA-Hilfe nicht dokumentiert, wird aber vom Makrorekorder aufgezeichnet. Akzeptiert einen der folgenden WdMergeSubType -Werte und beeinflusst, wie Word die Verbindung einer Datenquelle gestaltet. wdMergeSubTypeAccess wdMergeSubTypeOAL wdMergeSubTypeOLEDBText wdMergeSubTypeOLEDBWord wdMergeSubTypeOther wdMergeSubTypeOutlook wdMergeSubTypeWord wdMergeSubTypeWord2000 wdMergeSubTypeWorks
* Diese Parameter stammen aus der Open -Methode des Document -Objekts, haben für die Seriendruck- OpenDataSource -Methode jedoch keine Wirkung. Sie können problemlos 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, wenn **SubType:=wdMergeSubType2000**
- Für alle andere Datenbankarten wird Word mangels ODBC DSN versuchen, 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 ab Office 2007 nicht mehr im Lieferumfang enthalten ist, vorhanden sein und registriert sein muss.)
- Ohne **SubType:=wdMergeSubType2000** wird versucht, eine OLE DB-Verbindung zu erstellen

Neben der Word-Version 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 Beispieldatenverbindungen

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\Zusatzmaterial Seriendruck*. Ab 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 HTML-Dateien

Um zu einem Word-Dokument, einer RTF- oder HTML-Datei mit Tabelle eine Verbindung herzustellen:

```
ActiveDocument.MailMerge.OpenDataSource Name="c:\Datenbank\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
```

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.

Wie am Anfang dieses Abschnitts bemerkt, erkennt DDE Access-Datenbanken mit der Dateiendung *.accdb* nicht. Der Dateiname muss zuerst geändert werden, um eine DDE-Verbindung herzustellen (und anschließend wieder zurückgesetzt werden).

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.

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", _
    Subtype:=wdMergeSubTypeWord2000
```

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", _
    Subtype:=wdMergeSubTypeWord2000
```

Verbindung zu einer Abfrage, mit SQL-Anweisung

```
ActiveDocument.MailMerge.OpenDataSource Name="C:\Datenbank\Nordwind.mdb", _
    SQLStatement:="SELECT EinzelPreis FROM [Die zehn teuersten Artikel]", _
    Subtype:=wdMergeSubTypeWord2000
```

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*.

```
strDatenPfad = "C:\Datenbank\Nordwind.mdb"
ActiveDocument.MailMerge.OpenDataSource Name:= strDatenPfad, _
    Connection:="DSN=Microsoft Access-Datenbank;" & _
        "DBQ= & strDatenPfad & ";" & _
        "DriverId=25;FIL=MS Access;MaxBufferSize=2048;PageTimeout=5;", _
    SQLStatement:="SELECT * FROM [Bestellungen] WHERE [Kunden-Code]='CHOPS'", _
    Subtype:=wdMergeSubTypeWord2000
```

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.

```
strOrdner = "c:\Datenbank\"
MailMerge.OpenDataSource Name:=strOrdner & "leer.odc", _
    Connection:="Provider=Microsoft.Jet.OLEDB.4.0;Password=sdkw;" & _
        "User ID=wb;Data Source=" & strOrdner & "artikelsd.mdb;" & _
        "Jet OLEDB:System Database=" & strOrdner & "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
```

Word 2010 unterstützt die Angabe eines Zellenbereichs nicht, eines Bereichsnamens jedoch schon. Wobei es anscheinend keine Rolle spielt, ob der Bereichsname tatsächlich in der Mappe vorhanden ist. Falls nicht, wird der gesamte benutzte Bereich des ersten Arbeitsblatts zurückgegeben.

Wie am Anfang dieses Abschnitts bemerkt, erkennt DDE Excel-Mappen mit der Dateiendung .xlsx nicht. Der Dateiname muss zuerst geändert werden, um eine DDE-Verbindung herzustellen (und anschließend wieder zurückgesetzt werden).

HINWEIS

Eine DDE-Verbindung kann nur das erste Arbeitsblatt einer Excel-Arbeitsmappe ansprechen. Damals, als DDE eingeführt wurde, bestanden Excel-Mappen aus lediglich einem Arbeitsblatt. Die Technologie wurde nie überarbeitet und so können die neuen Möglichkeiten von Excel-Dateien nicht genutzt werden. Falls ein Bereichsname angegeben wird, und dieser sich auf einem anderen Arbeitsblatt befindet, werden die entsprechenden Zellen vom ersten Arbeitsblatt zurückgegeben.

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:

```
strOrdner = "c:\Datenbank\"
ActiveDocument.MailMerge.OpenDataSource Name="leer.odc", _
    Connection:="Provider=Microsoft.Jet.OLEDB.4.0;" & _
        "Data Source=" & strOrdner & "Bestellungen.xls;" & _
        "Extended Properties=""Excel 8.0;HDR=Yes"";", _
    SQLStatement:="SELECT * FROM [Bestellungen]", _
    SubType:=wdMergeSubTypeOther
```

Für Excel-Dateien im XML-Dateiformat (.xls bzw. .xslm) muss die Verbindung über den neuen OLE DB-Provider ACE erstellt werden, anstelle von JET: Provider=Microsoft.ACE.OLEDB.12.0. Die folgende Syntax, die den Namen der Datenquelle sowie eine SQL-Anweisung, jedoch keine Verbindungszeichenkette, angibt, müsste funktionieren:

```
ActiveDocument.MailMerge.OpenDataSource Name="c:\Datenbank\Bestellungen.xls", _
    SQLStatement:="SELECT * FROM [Bestellungen]"
```

WICHTIG

Obwohl ab der Version 2007 in einem Excel-Arbeitsblatt mehr als 255 Spalten unterstützt werden, gibt es keine Seriendruck-Verbindungsmethode, die diese neue Möglichkeit nutzt. Somit können nur die ersten 255 Spalten genutzt werden.

Microsoft SQL Server

Um eine Verbindung zu Microsoft SQL Server herzustellen, genügt es meistens, eine .odc-Datei mittels der Benutzerschnittstelle zu erstellen (über die Schaltfläche *Neue Quelle* im Dialogfeld *Datenquelle auswählen*). Dazu muss lediglich der Dateipfad an das Argument Name und fakultativ eine SQL-Anweisung an das Argument SQL-Statement übergeben werden.

Unter Umständen müssen die Informationen im Knowledge Base Artikel »Fehlermeldung, wenn Sie eine ODC-Datei, verwenden Verbindung zu einer bestimmten Tabelle in einer Seriendruck-Datenquelle in ein Office 2007-Programm oder Office 2003: "Datensatz 'Nummer' enthält zu wenige Datenfelder"« berücksichtigt werden (<http://support.microsoft.com/kb/918295/de>).

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 342)
- Anschließend folgte eine Vorstellung der verschiedenen Nutzen einer Tabelle in Word-Dokumenten (Seite 351 ff.)
- Danach wurde der Umgang mit Feldfunktionen (Seite 380 ff.) beschrieben
- In den nächsten Abschnitten wurden Formularfelder (Seite 393 ff.) und Inhaltssteuerobjekte (Seite 402 ff.) erläutert
- Eine Diskussion zum Thema Seriendruck (Seite 433 ff.) rundete dieses Kapitel ab

Basierend auf der Erfahrung der Autoren in den Newsgroups wurden Aspekte der Benutzerschnittstelle aufgezeigt, die bekanntlich für Missverständnisse und Probleme sorgen. Entsprechende Vorschläge zum Umgehen dieser Probleme und zur Automatisierung derselben sind auch vorhanden.

Kapitel 8

Ereignisse in Word

In diesem Kapitel:

AutoMakros als Pseudoereignisse	454
Ereignisse auf Dokumentebene	456
Ereignisse auf Applikationsebene	459
Klassenmodule einrichten und initialisieren	461
Übersicht über die verfügbaren Ereignisse	464
Seriendruckereignisse	478
ProtectedViewWindow-Ereignisse	484
Zusammenfassung	492

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.

Mithilfe 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.
- **Ereignisse auf Dokumentebene** Diese Ereignisse werden beim Erstellen, Öffnen oder Schließen von Dokumenten ausgeführt (siehe den Abschnitt »Ereignisse auf Dokumentebene« ab Seite 456).
- **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 Serienbriefherstellung und den Umgang mit Inhaltssteuerelementen (siehe den Abschnitt »Ereignisse auf Applikationsebene« ab Seite 459).
- **Automatisch ausgeführte Makros** Neben diesen Ereignissen gibt es zusätzlich noch fünf spezielle AutoMakros, die automatisch ausgeführt werden, wenn Sie eine der damit verbundenen Aktionen ausführen (siehe den folgenden Abschnitt).

In diesem Kapitel erfahren Sie, welche Ereignisse Word 2007 und Word 2010 auf Dokument- und Anwendungsebene besitzen und wie Sie diese nutzen können.

AutoMakros als Pseudoereignisse

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 sogenannten AutoMakros.

Diese Makros werden automatisch ausgeführt, wenn ein Makro mit diesem Namen erstellt und die mit dem AutoMakro verknüpfte Aktion ausgeführt wird. Damit Word diese auszuführenden Makros als solche erkennt, müssen sie besonders benannt werden.

Tabelle 8.1 Zusammenstellung der AutoMakros 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

Tabelle 8.1 Zusammenstellung der AutoMakros und deren Ausführungszeitpunkt (Fortsetzung)

Bezeichnung	Zeitpunkt der Ausführung
AutoNew	Beim Erstellen eines neuen Dokuments
AutoExit	Beim Beenden von Word bzw. beim Entladen eines Add-Ins

Die Makros `AutoOpen` und `AutoClose` können im aktuellen Dokument, der zugehörigen Dokumentvorlage oder in der *Normal.dotm* abgespeichert werden.

Das Makro `AutoNew` kann in einer Dokumentvorlage oder in der *Normal.dotm* abgespeichert werden.

Die Makros `AutoExec` und `AutoExit` können in der *Normal.dotm* oder einer globalen Vorlage (Dokument-Add-In; mehr darüber lesen Sie im Kapitel 10) abgespeichert werden.

Die AutoMakros werden unabhängig von ihrem Speicherort ausgeführt, wenn eine der beiden folgenden Bedingungen erfüllt ist:

- Das Makro besitzt einen der genannten Namen und ist in einem Modul oder im Bereich *This Document* eines Dokuments, einer Dokumentvorlage oder eines geladenen Add-Ins gespeichert
- Das Modul ist nach dem Namen des AutoMakros 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.dotm* und das Makro aus der *Normal.dotm* vor jenem aus dem Add-In abgearbeitet (siehe auch Kapitel 14).

ACHTUNG Diese Regelung gilt nicht nur für die AutoMakros sondern grundsätzlich für alle Makros. Spezialfälle sind lediglich `AutoExec` und `AutoExit`. Diese beiden Makros werden für die *Normal.dotm* 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.

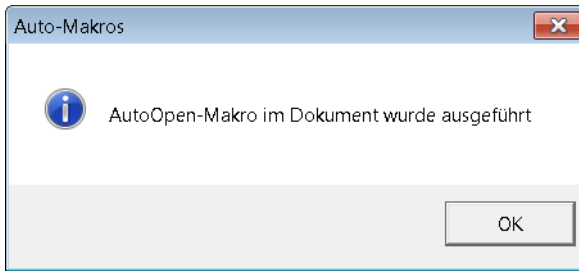
Die Reihenfolge für das Laden der im Autostart-Ordner gespeicherten Add-Ins kann dabei nicht beeinflusst werden.

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.dotm*,
- 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 AutoMakro `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« ab Seite 461).

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 zugrunde liegende Dokumentvorlage.
3. Die Standarddokumentvorlage *Normal.dotm*.
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 AutoMakros in einzelnen Fällen unterbinden möchten, halten Sie beim Öffnen der Datei die -Taste gedrückt.

Alternativ können Sie in einem Makro, das ein AutoMakro auslöst, indem es z.B. ein neues Dokument erstellt, das Ausführen aller AutoMakros mit folgendem Befehl unterbinden:

```
Application.AutomationSecurity
```

bzw.

```
WordBasic.DisableAutoMacros
```

Weitere Informationen mit Beispielen finden Sie in Kapitel 5.

Ereignisse auf Dokumentebene

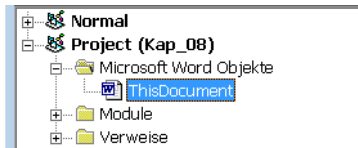
Neben den AutoMakros gibt es auf Dokumentebene vergleichbare Ereignisse, mit denen auf das Öffnen, Erstellen oder Schließen von Dokumenten reagiert werden kann.

Tabelle 8.2 Ereignisse auf Dokumentebene

Name des Ereignisses	Aurufende Aktion
Document_New()	Erstellen eines neuen Dokuments
Document_Open()	Öffnen eines vorhandenen Dokuments
Document_Close()	Schließen eines Dokuments

Im Gegensatz zu den AutoMakros, die an verschiedenen Stellen im Dokument oder der Dokumentvorlage gespeichert sein können, erfordern die Dokumentereignisse als Speicherort den Bereich *ThisDocument* in der Dokumentvorlage, die dem Dokument zugrunde 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 20.

Da sowohl die Dokumentvorlage wie auch das Dokument dieses Klassenmoduls *ThisDocument* besitzen, können beide Klassenmodule die Ereignisse auf Dokumentebene auslösen.

Wenn Sie also in beiden Klassenmodulen (dem eines Dokuments und dem der zugrunde 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 AutoMakros, auch in der Standarddokumentvorlage *Normal.dotm* Dokumentereignisse definieren, werden diese nur dann ausgelöst, wenn das jeweilige Dokument direkt auf dieser Vorlage basiert. Bei Dokumenten mit eigenen Dokumentvorlagen werden die Dokumentereignisse in der *Normal.dotm* nicht ausgelöst.

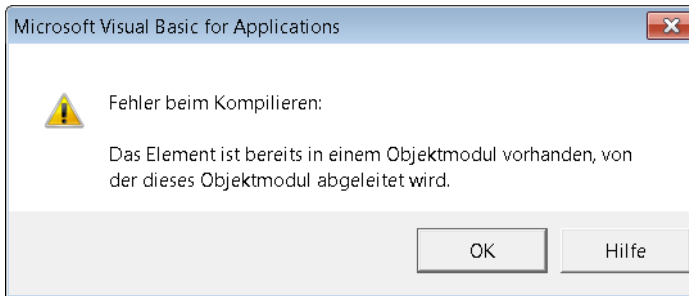
Definieren Sie zusätzlich zu den Dokumentereignissen entsprechende AutoMakros, werden diese in folgender Reihenfolge ausgelöst (z.B. beim Öffnen eines Dokuments):

- AutoMakro: `AutoOpen()`
- Dokumentereignis in der Dokumentvorlage: `Document_Open()`
- Dokumentereignis 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 Namenskonflikts 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 Ereignisprozedur in einem Dokument nicht den gleichen Namen besitzen darf wie eine gleichnamige öffentliche Ereignisprozedur der Dokumentvorlage, erhalten Sie die in Abbildung 8.3 gezeigte Fehlermeldung.

Abbildg. 8.3 Fehlermeldung beim Verwenden einer privaten und einer öffentlichen gleichnamigen Ereignisprozedur



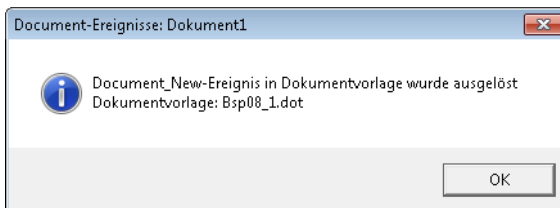
Als Beispiel für ein `Document_New()`-Ereignis auf Dokumentebene kopieren Sie das Makro aus Listing 8.1 in das Modul *ThisDocument* einer Dokumentvorlage oder erstellen Sie ein neues Dokument auf der Basis der Beispieldokumentvorlage *Bsp_08-1.dotm*.

Kopieren Sie die Dokumentvorlage in ein beliebiges Verzeichnis auf Ihrer Festplatte (z.B. in den Ordner *Eigene Dateien* unter Windows XP/Windows Server 2003 bzw. in den Ordner *Dokumente* unter Windows Vista/Windows 7) und führen einen Doppelklick auf diese Dokumentvorlage aus. Es wird daraufhin Word gestartet und ein neues leeres Dokument mit dem Dialogfeld aus Abbildung 8.4 angezeigt. Falls das Dialogfeld nicht dargestellt wird oder eine Sicherheitswarnung ausgegeben wird, dann müssen die Sicherheitseinstellungen geprüft werden (siehe in Kapitel 1 den Abschnitt »Makrosicherheit«).

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 = "Document-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
```

Abbildg. 8.4 Ausgabe beim Erstellen eines neuen Dokuments basierend auf einer Vorlage mit `Document_New`-Ereignis



CD-ROM Die Beispieldokumentvorlage *Bsp08_1.dotm* finden Sie auf der CD-ROM zum Buch im Ordner `\Beispiele\Kap08`.

HINWEIS

Die weiteren Dokumentereignisse, die objektgebunden 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 Inhaltssteuerelemente und einen dazugehörenden CustomXML-Teil beziehen
- Ereignisse, die sich auf das E-Porto-Add-In beziehen
- Ereignisse, die sich auf Dokumente beziehen, die in einem geschützten Ansichtsfenster angezeigt werden

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 Dokumentereignisse müssen sich die Applikationsereignisse 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« ab Seite 456) oder im Projekt-Explorer des Visual Basic-Editors ein neues Klassenmodul im Bereich *Klassenmodule* (siehe Kapitel 2) anlegen.

Diese beiden Speicherorte für die Applikationsereignisse 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, sodass Sie dort nur die Ereignisse definieren müssen, während Sie die Klassenmodule im Bereich *Klassenmodule* erst einrichten und initialisieren müssen (siehe den Abschnitt »Klassenmodule einrichten und initialisieren« ab Seite 461).

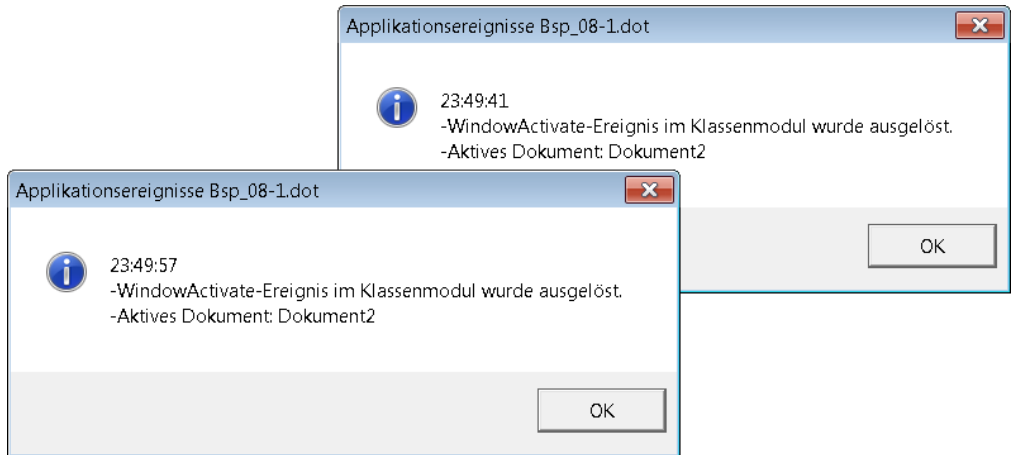
Ereignisse im Klassenmodul *ThisDocument* werden bei Add-Ins (z.B. Dokumentvorlagen, die im *Startup*-Ordner von Word gespeichert sind) nicht ausgeführt.

CD-ROM

Als Test für das unterschiedliche Verhalten können Sie die Dokumentvorlage *Bsp08_1.dotm* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap08* zum einen direkt per Doppelklick aufrufen und zum anderen in den *Startup*-Ordner von Word kopieren und dann Word starten.

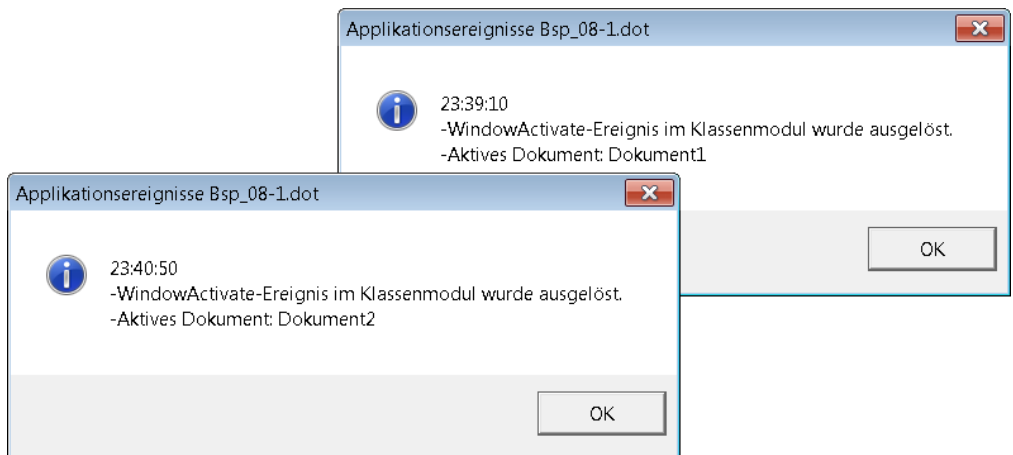
Wenn Sie mit einem Doppelklick auf die Dokumentvorlage eine neue Datei erstellen und anschließend über das Symbol *Neues leeres Dokument* oder über den entsprechenden Menübefehl *Datei/Neu* und dann im Aufgabenbereich über den Befehl *Leeres Dokument* weitere Dateien erstellen, werden Ihnen in Dialogfelder 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 Applikationsereignissen zuerst die zugrunde 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 Codefenster 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 Ereignisvariablen 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 anschließend in der Objektauswahlliste das Objekt *App* auswählen, stehen Ihnen in der Prozedurauswahlliste alle verfügbaren Ereignisse dieses Objekts zur Verfügung. Wenn Sie einen Prozedureintrag aus der Liste anklicken, wird automatisch eine syntaktisch korrekte Prozedur im Codefenster 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 Prozedurauswahlliste bequem alle Ereignisse mit korrekter Syntax erstellen.

Insgesamt stehen ab Word 2003 folgende Ereignisse zur Verfügung und werden in der Prozedurauswahlliste aufgeführt:

- Applikationsereignisse
 - WindowActivate
 - WindowDeactivate
 - WindowSize
 - WindowBeforeDoubleClick
 - WindowBeforeRightClick
 - WindowSelectionChange
 - Sync
 - Quit
- Dokumentspezifische Ereignisse
 - NewDocument
 - DocumentOpen
 - DocumentChange
 - DocumentBeforeClose
 - DocumentBeforePrint
 - DocumentBeforeSave
 - DocumentSync
- Serienbrief-spezifische Ereignisse
 - MailMergeAfterMerge
 - MailMergeAfterRecordMerge
 - MailMergeBeforeMerge
 - MailMergeBeforeRecordMerge
 - MailMergeDataSourceLoad
 - MailMergeDataSourceValidate
 - MailMergeDataSourceValidate2 (ab Word 2007)
 - MailMergeWizardSendToCustom
 - MailMergeWizardStateChange
- XML-spezifische Ereignisse
 - XMLSelectionChange
 - XMLValidationError
- ProtectedViewWindow-spezifische Ereignisse
 - ProtectedViewWindowActivate
 - ProtectedViewWindowOpen
 - ProtectedViewWindowBeforeEdit
 - ProtectedViewWindowDeactivate
 - ProtectedViewWindowBeforeClose
 - ProtectedViewWindowSize

- E-Porto-Ereignisse
 - EPostageInsert
 - EPostageInsertEx
 - EPostagePropertyDialog

Diese Ereignisse werden im Abschnitt »Übersicht über die verfügbaren Ereignisse« ab Seite 464 im Detail behandelt.

Klassenmodule initialisieren

Damit das System auf die Ereignisse im Klassenmodul reagieren und die entsprechenden Ereignisprozeduren ausführen kann, muss das Klassenmodul über die deklarierte Objektvariable initialisiert werden.

Dazu müssen Sie in einem normalen Modul, im Beispiel wird `modCLSInit` verwendet, eine globale Objektvariable auf Modulebene deklarieren, die ein neues Instanzobjekt 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), sodass in der Auswahlliste nur diese Variable `App` angeboten wird (siehe Abbildung 8.7).

Als Nächstes muss die Schnittstelle zum Klassenmodul mithilfe 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 20) 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 Ereignisprozeduren 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 2007 und in Word 2010 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 Ereignisprozedur *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

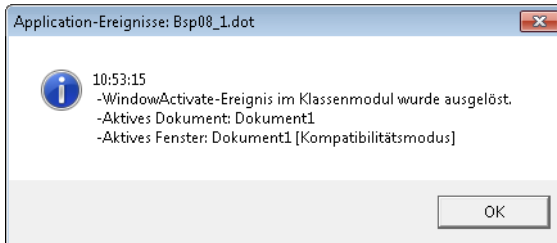
Listing 8.6 Beispiel zur Ereignisprozedur *WindowActivate*, das die jeweiligen Parameter anzeigt

```
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 Ereignisprozedur *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 Ereignisprozedur *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 Fenster 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.

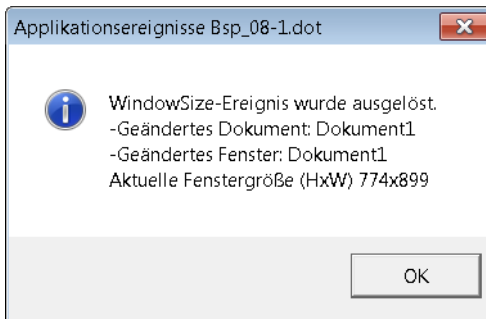
Listing 8.7 Die Ereignisprozedur *WindowSize* gibt die Größe des aktuellen Fensters aus, wenn es verkleinert oder vergrößert wird

```
Private Sub App_WindowSize(ByVal Doc As Document, ByVal Wn As Window)
    Dim strMSG As String
    Const c_Title As String = "Applikationsereignisse "
    strMSG = "WindowSize-Ereignis wurde ausgelöst." & vbCrLf & _
        " -Geändertes Dokument: " & Doc.Name & vbCrLf & _
        " -Geändertes Fenster: " & Wn.Caption & vbCrLf & _
```

Listing 8.7 Die Ereignisprozedur *WindowSize* gibt die Größe des aktuellen Fensters aus, wenn es verkleinert oder vergrößert wird (Fortsetzung)

```
Select Case Wn.WindowState
Case wdWindowStateNormal, wdWindowStateMinimize
    strMSG = strMSG & "Aktuelle Fenstergröße (HxW): "
    & PointsToPixels(Wn.Height) & "x" & PointsToPixels(Wn.Width) & vbCrLf
End Select
MsgBox strMSG, vbInformation, c_Title & ThisDocument.AttachedTemplate.Name
End Sub
```

Abbildg. 8.9 Ausgabe der aktuellen Fenstergröße durch die Ereignisprozedur *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 Ereignisprozedur *WindowBeforeDoubleClick*

Parameter	Beschreibung
Sel	Referenz auf die Markierung, auf die der Doppelklick erfolgte, mit allen Eigenschaften und Methoden eines Selection -Objekts
Cancel	Parameter zum Steuern der Ereignisausführung. Standardmäßig wird die Aktion mit Cancel=False ausgeführt; durch Setzen von Cancel=True kann die Aktion unterbunden werden.

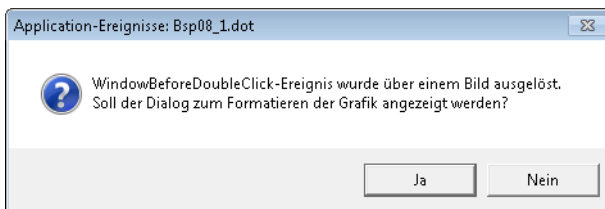
Sie können mit diesem Ereignis z.B. den Typ der Markierung abfragen und nur gezielt auf einen Doppelklick auf ein Formularfeld reagieren. In dem Beispiel in Listing 8.8 wird geprüft, ob das markierte Objekt ein *FormField* ist. Nur für diesen Fall wird gefragt (Abbildung 8.10), ob das standardmäßig angezeigte Dialogfeld zum Formatieren des Formularfeldes angezeigt werden soll oder nicht.

Listing 8.8 Beispiel zur Ereignisprozedur *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.FormFields.Count > 0 Then
        strMSG = "WindowBeforeDoubleClick-Ereignis wurde über einem " _
            "Formularfeld ausgelöst." & vbCrLf & _
            "Soll das Dialogfeld zum Formatieren des Formularfeldes " _
            "angezeigt werden?" & vbCrLf
        ret = MsgBox(strMSG, vbQuestion + vbYesNo, c_Title & _
            ThisDocument.AttachedTemplate.Name)
        If ret = vbNo Then
            MsgBox "Dialogfeld wird nicht angezeigt.", vbInformation, "Aktion ausführen"
            Cancel = True
        End If
    End If
End Sub

```

Abbildg. 8.10 Ausgabe der Aktion der Ereignisprozedur *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 Ereignisprozedur *WindowBeforeRightClick* blendet bei Formularfeldern ein eigenes Kontextmenü ein

```

Private Sub App_WindowBeforeRightClick(ByVal Sel As Selection, Cancel As Boolean)
    If ActiveDocument.ProtectionType = wdAllowOnlyFormFields Then
        CreateContextMenu.ShowPopup
        Cancel = True
    End If
End Sub

```

Listing 8.9 Die Ereignisprozedur *WindowBeforeRightClick* blendet bei Formularfeldern ein eigenes Kontextmenü ein (Fortsetzung)

```

ElseIf ActiveDocument.ProtectionType = wdNoProtection Then
    Dim fldFF As FormField
    For Each fldFF In ActiveDocument.FormFields
        If Sel.Range.InRange(fldFF.Range) Then
            CreateContextMenu.ShowPopup
            Cancel = True
        End If
    Next fldFF
End If
End Sub

Function CreateContextMenu() As CommandBar
    Dim cbar As CommandBar
    Dim ctl1 As CommandBarControl
    Dim ctl2 As CommandBarControl
    CustomizationContext = ActiveDocument.AttachedTemplate
    Set cbar = CommandBars("Form Fields")
    cbar.Reset
    With cbar
        Set ctl1 = .FindControl(msoControlButton, 1, "Dokumentschutz aufheben", , True)
        If ctl1 Is Nothing Then
            Set ctl1 = .Controls.Add(msoControlButton, 1, , , True)
        End If
        ctl1.Caption = "Dokumentschutz aufheben"
        ctl1.Tag = "Dokumentschutz aufheben"
        ctl1.OnAction = "UnprotectDoc"
        Set ctl2 = .FindControl(msoControlButton, 1, "Dokumentschutz setzen", , True)
        If ctl2 Is Nothing Then
            Set ctl2 = .Controls.Add(msoControlButton, 1, , , True)
        End If
        ctl2.Caption = "Dokumentschutz setzen"
        ctl2.Tag = "Dokumentschutz setzen"
        ctl2.OnAction = "ProtectDoc"
        If ActiveDocument.ProtectionType = wdNoProtection Then
            ctl1.Enabled = False
            ctl2.Enabled = True
        ElseIf ActiveDocument.ProtectionType = wdAllowOnlyFormFields Then
            ctl1.Enabled = True
            ctl2.Enabled = False
        End If
    End With
    Set CreateContextMenu = cbar
    Set ctl2 = Nothing
    Set ctl1 = Nothing
    Set cbar = Nothing
End Function

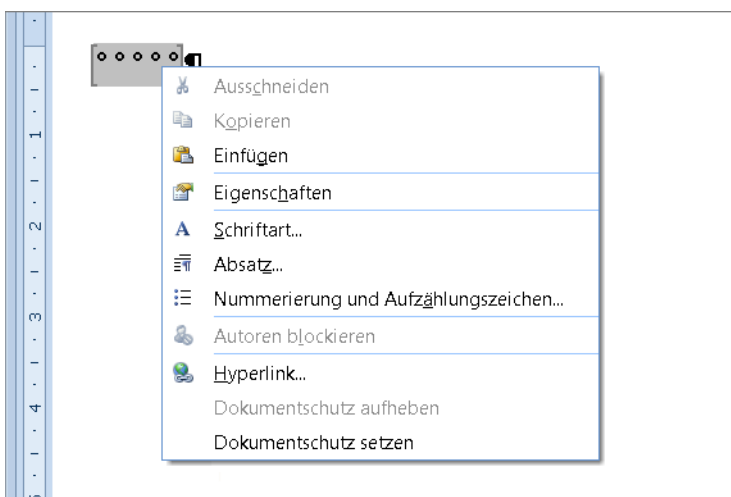
```

Damit das Beispiel aus Listing 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
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*



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 Ereignisprozedur *WindowSelectionChange*

Parameter	Beschreibung
Sel	Referenz auf die Markierung, auf die die Einfügemarke verschoben wurde, mit allen Eigenschaften und Methoden eines Selection -Objekts

Sie können z.B. den Typ der Markierung abfragen und nur gezielt auf das Markieren eines Feldes reagieren.

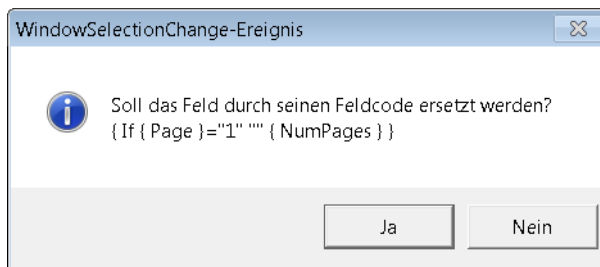
ACHTUNG Das Ereignis *WindowSelectionChange* wird nicht ausgelöst, wenn Sie mit der Tastatur die Einfügemarke verschieben, z.B. bei der Texteingabe.

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 Ereignisprozedur *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 Ereignisprozedur *NewDocument*

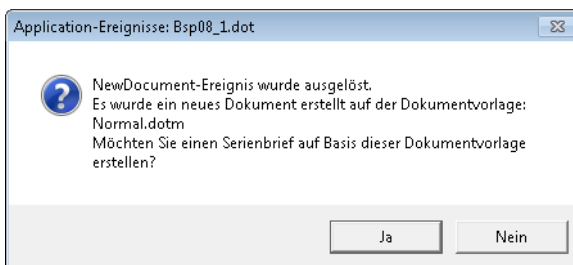
Parameter	Beschreibung
Doc	Referenz auf das neu erstellte Dokument mit allen Eigenschaften und Methoden eines Document-Objekts

Sie können mit diesem Ereignis z.B. direkt eine Aktion mit dem erstellten Dokument verknüpfen. Wenn es auf einer bestimmten Dokumentvorlage basiert (in Listing 8.12 auf der *Normal.dotm*), können Sie z.B. direkt den Assistenten zum Erstellen eines Serienbriefs aufrufen.

Listing 8.12 Verwenden der Ereignisprozedur *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 Ereignisprozedur *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 angezeigt ist. Dieses Dialogfeld besitzt keine eigene benannte Konstante, sodass 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 Ereignisprozedur *DocumentBeforeClose*

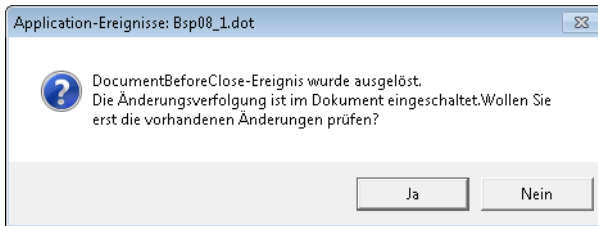
Parameter	Beschreibung
Doc	Referenz auf das zu schließende Dokument mit allen Eigenschaften und Methoden eines Document -Objekts
Cancel	Parameter zum Steuern der Ereignisausführung. Standardmäßig wird die Aktion mit Cancel=False ausgeführt. Durch Setzen von Cancel=True kann das Ausführen der Aktion verhindert 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 vorhandenen Änderungen prüfen?"  
            ret = MsgBox(strMSG, vbQuestion + vbYesNo, c_Title & _  
                ThisDocument.AttachedTemplate.Name)  
            If ret = vbYes Then  
                Doc.ShowRevisions = True  
                Cancel = True  
                Doc.Revisions(1).Range.Select  
            End If  
        End If  
    End If  
End Sub
```

Abbildg. 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 Ereignisprozedur *DocumentBeforeSave*

Parameter	Beschreibung
Doc	Referenz auf das aktuell zu speichernde Dokument mit allen Eigenschaften und Methoden eines Document -Objekts
SaveAsUI	Parameter zur Anzeige des <i>Speichern unter</i> -Dialogfelds. Leider wird dieser Parameter entgegen der Onlinehilfe von Word 2003 nicht unterstützt.
Cancel	Parameter zum Steuern der Ereignisausführung. Standardmäßig wird die Aktion mit Cancel=False ausgeführt. Durch Setzen von Cancel=True 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.

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 Ereignisprozedur *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
```

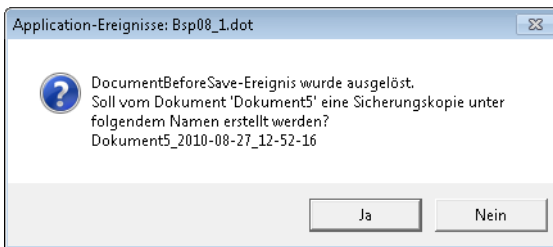
Listing 8.16 Die Ereignisprozedur *DocumentBeforeSave* fragt nach, ob eine Sicherungskopie erstellt werden soll (*Fortsetzung*)

```

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 Schriftzugs »Kopie« in das Dokument einzufügen (Abbildung 8.16).

Listing 8.17 Einfügen eines Wasserzeichens mit der Ereignisprozedur *DocumentBeforePrint*

```

Private Sub App_DocumentBeforePrint(ByVal Doc As Document, Cancel As Boolean)
    Dim strMSG As String
    Dim ret As Integer
    Dim oHead As HeaderFooter
    Dim shpHead As Shape

```

Listing 8.17 Einfügen eines Wasserzeichens mit der Ereignisprozedur *DocumentBeforePrint* (Fortsetzung)

```

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 stehen in den Kapiteln 6 und 12 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 Ereignisprozedur *DocumentSync*

Parameter	Beschreibung
Doc	Referenz auf das lokale zu synchronisierende Dokument mit allen Eigenschaften und Methoden eines Document -Objekts
SyncEventType	Parameter, der den Status der Synchronisation wiedergibt. Folgende msoSyncEventType -Werte sind möglich: msoSyncEventDownloadFailed msoSyncEventDownloadInitiated msoSyncEventDownloadNoChange msoSyncEventDownloadSucceeded msoSyncEventOffline msoSyncEventUploadFailed msoSyncEventUploadInitiated msoSyncEventUploadSucceeded

Über den Parameter **SyncEventType** können Sie den Status der Synchronisation ablesen.

Listing 8.18 Ausgabe der Ereignisprozedur *DocumentSync* bei einem Synchronisationsfehler

```
Private Sub App_DocumentSync(ByVal Doc As Document, _
    ByVal SyncEventType As Office.MsoSyncEventType) _
    Dim strMSG As String
    Const c_Title As String = "Applikationsereignisse "
    If SyncEventType = msoSyncEventDownloadFailed Or _
        SyncEventType = msoSyncEventUploadFailed Then _
        strMSG = "Fehler beim Synchronisieren des Dokumentes '" & Doc & "'"
        MsgBox strMSG, vbCritical, c_Title & ThisDocument.AttachedTemplate.Name
    End If
End Sub
```

Seriendruckereignisse

Neben den Ereignissen des Seriendruck-Aufgabenbereichs, die in Kapitel 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 Ereignisprozedur *MailMergeBeforeMerge*

Parameter	Beschreibung
<code>Doc</code>	Gibt ein Document -Objekt zurück, das sich auf das Seriendruck-Hauptdokument 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 False . Wird ausdrücklich der Wert True 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 zu 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

<p>Nordwind GmbH</p> <p>«Firma» «Kontaktperson» «Straße» «PLZ» «Ort» «Land»</p> <p>MERCHANDISE ORDER</p> <p>Dear «Kontaktperson»</p> <p>Please send us the following articles:</p> <p>I</p> <p>For transportation and payment conditions, the agreements in our contract apply.</p>	<p>Munich, 31. October 2010</p>
--	---------------------------------

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*

<p>Formaggi Fortini s.r.l. Elio Rossi Viale Dante, 75 48100 Ravenna Italien</p> <p>MERCHANDISE ORDER</p> <p>Dear Elio Rossi</p> <p>Please send us the following articles:</p>							
<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: left;">Product</th> <th style="text-align: right;">Quantity</th> </tr> </thead> <tbody> <tr> <td>Gorgonzola Telino 12 x 100-g-Packungen</td> <td style="text-align: right;">22</td> </tr> <tr> <td>Mozzarella di Giovanni 24 x 200 g-Packungen</td> <td style="text-align: right;">22</td> </tr> </tbody> </table>	Product	Quantity	Gorgonzola Telino 12 x 100-g-Packungen	22	Mozzarella di Giovanni 24 x 200 g-Packungen	22	
Product	Quantity						
Gorgonzola Telino 12 x 100-g-Packungen	22						
Mozzarella di Giovanni 24 x 200 g-Packungen	22						

Diese Voraussetzungen bedingen, dass die Datensätze nach Lieferant sortiert sind, wofür die Prozedur, die den Seriendruck ausführt, sorgt, indem der SQL-Anweisung (QueryString-Eigenschaft) eine Order By-Klausel hinzugefügt wird. Jetzt kann durch die Datensätze geschleift werden, um die Anzahl der Einträge jedes Lieferanten zu ermitteln. Da während der Zusammenführung der Seriendruck nur vorwärts durch die Datensätze schreiten kann, muss dies vor der Zusammenführung getan werden. 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 Ereignisprozedur *MailMergeBeforeRecordMerge*

```
Private Sub wdApp_MailMergeBeforeRecordMerge(ByVal doc As Document, Cancel As Boolean)
    If DatenSatzPrüfen(doc) Then
        Cancel = True
    End If
End Sub
```

Die Funktion *DatenSatzPrüfen* arbeitet mit dem aktuellen Datensatz. Beim Wechsel der Firma werden die globalen Variablen, worin die Arrayelemente 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 umwandelt.

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, sodass das Dokument wieder in seinem

```

Listing 8.22 Im Ereignis *MailMergeAfterRecordMerge* wird das Hauptdokument zurückgesetzt (*Fortsetzung*)

```
'ursprünglichen Zustand für den nächsten Datensatz bereitsteht.
rng.Tables(1).Delete
doc.Bookmarks.Add Name:=sBkmName, Range:=rng
End Sub
```

Nachdem alle Datensätze abgearbeitet wurden, kommt das Ereignis *MailMergeAfterMerge* (Listing 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 Abschlusshandlungen 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
```

CD-ROM

Die Beispieldatei *Bsp08_03.docm* mit dem Beispielcode sowie die Datei *Bsp08_Beiispiel.doc* mit dem Seriendruckresultat finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap08*. Die Seriendruckdatenquelle *Artikelliste.txt* befindet sich im Ordner *\Beispiele\Datenbank*.

ProtectedViewWindow-Ereignisse



Neu in Word 2010 hinzugekommen sind die Ereignisse, die beim Umgang mit geschützten Dokumentansichten ausgelöst werden. Geschützte Dokumentansichten (geschützte Ansichtsfenster) werden immer dann verwendet, wenn die Herkunftsseite oder der Herkunftsort nicht sicher oder vertrauenswürdig ist. Dokumente, die in einem geschützten Ansichtsfenster angezeigt werden, können nicht bearbeitet werden und dürfen keine aktiven Inhalte ausführen (VBA-Makros oder Datenverbindungen).

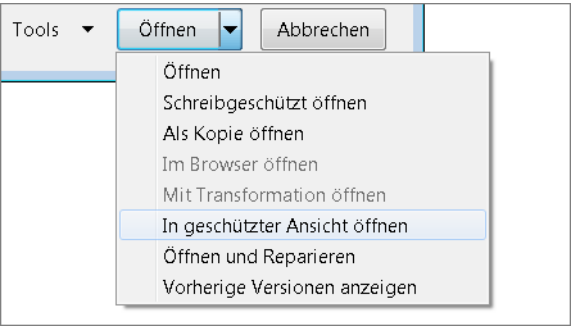
Zentrales Element der geschützten Ansichtsfenster ist das *ProtectedViewWindow*-Objekt, das das geschützte Ansichtsfenster, in dem ein Dokument ggf. geöffnet wird, darstellt. Werden mehrere Dokumente in geschützten Ansichtsfenstern geöffnet, erfolgt der Zugriff auf die verschiedenen Ansichtsfenster über die Index-Nummer des Fensters.

HINWEIS

Auch das *Öffnen*-Dialogfeld in Word bietet die Option, ein Dokument in geschützter Ansicht zu öffnen.

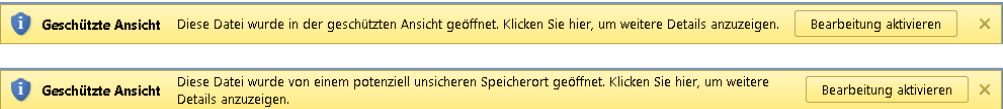
Klicken Sie dazu auf den Pfeil neben der Schaltfläche *Öffnen* und wählen Sie im Dropdownmenü den Eintrag *In geschützter Ansicht öffnen* aus.

Abbildg. 8.19 Datei in geschützter Ansicht öffnen



Sobald ein Dokument im geschützten Ansichtsfenster geöffnet wird, wird ein Hinweis über den Grund der geschützten Darstellung angezeigt.

Abbildg. 8.20 Hinweise beim Öffnen eines Dokuments in der geschützten Ansicht



HINWEIS Weitere Informationen zur geschützten Ansicht und seine Einstellmöglichkeiten in Word (über das Sicherheitszentrum) finden Sie in der Word-Hilfe unter dem Stichpunkt »Was ist die geschützte Ansicht?«.

Tabelle 8.11 Eigenschaften und Methoden des *ProtectedViewWindow*-Objekts

Eigenschaft	Beschreibung
Active	Referenz auf das lokale zu synchronisierende Dokument mit allen Eigenschaften und Methoden eines Document -Objekts
Application	Gibt das Word-Objekt zurück
Caption	Gibt den Namen des Ansichtsfensters zurück, oder legt ihn fest
Creator	Gibt die 32-Bit-Zahl der Word-Anwendung zurück Const wdCreatorCode = 1297307460 (&H4D535744)
Document	Gibt das zugeordnete Document -Objekt des angezeigten geschützten Ansichtsfensters zurück
Height/Width	Gibt die Höhe und Breite des Ansichtsfensters zurück, oder legt sie fest
Index	Gibt den Index des Ansichtsfensters zurück
Left/Top	Gibt die Position (Links und Oben) des Ansichtsfensters zurück, oder legt sie fest
Parent	Gibt das übergeordnete Objekt des Ansichtsfensters zurück
SourceName/ SourcePath	Gibt den Namen und Pfad des im Ansichtsfenster geöffneten Dokuments an

Tabelle 8.11 Eigenschaften und Methoden des *ProtectedViewWindow*-Objekts (Fortsetzung)

Eigenschaft	Beschreibung
Visible	Gibt den Sichtbarkeitszustand (True/False) des angegebenen geschützten Ansichtsfensters zurück, oder legt diesen fest
WindowState	Gibt den Zustand (wdWindowState -Konstante) des angegebenen geschützten Ansichtsfensters zurück, oder legt diesen fest
Activate	Aktiviert das angegebene Ansichtsfenster
Close	Schließt das angegebene Ansichtsfenster
Edit	Aktiviert ein Dokument in einem geschützten Ansichtsfenster zur Bearbeitung. Als Parameter können Kennwörter zum Öffnen der Vorlage, zum Speichern von Änderungen am Dokument oder zum Speichern von Änderungen an der Vorlage angegeben werden.
ToggleRibbon	Blendet das Menüband ein bzw. aus

Tabelle 8.12 Ereignisse des *ProtectedViewWindow*-Objekts

Eigenschaft	Beschreibung
ProtectedViewWindowActivate	Dieses Ereignis wird ausgeführt, wenn ein geschütztes Ansichtsfenster aktiviert wird
ProtectedViewWindowDeactivate	Dieses Ereignis wird ausgeführt, wenn ein geschütztes Ansichtsfenster deaktiviert wird
ProtectedViewWindowOpen	Dieses Ereignis wird ausgeführt, sobald ein Dokument im geschützten Ansichtsfenster geöffnet wird
ProtectedViewWindowSize	Dieses Ereignis wird ausgeführt, wenn ein geschütztes Ansichtsfenster in der Größe verändert oder verschoben wird
ProtectedViewWindowBeforeClose	Dieses Ereignis wird ausgeführt, bevor ein geschütztes Ansichtsfenster geschlossen wird
ProtectedViewWindowBeforeEdit	Dieses Ereignis wird ausgeführt, bevor ein geschütztes Ansichtsfenster zur Bearbeitung aktiviert wird

ProtectedViewWindowActivate

Das Ereignis `ProtectedViewWindowActivate` tritt ein, wenn Word ein geschütztes Ansichtsfenster aktiviert oder ein Dokument in einem geschützten Ansichtsfenster öffnet. Der Parameter wird in Tabelle 8.13 erläutert. Die Syntax lautet:

```
ProtectedViewWindowActivate(ByVal PvWindow As ProtectedViewWindow)
```

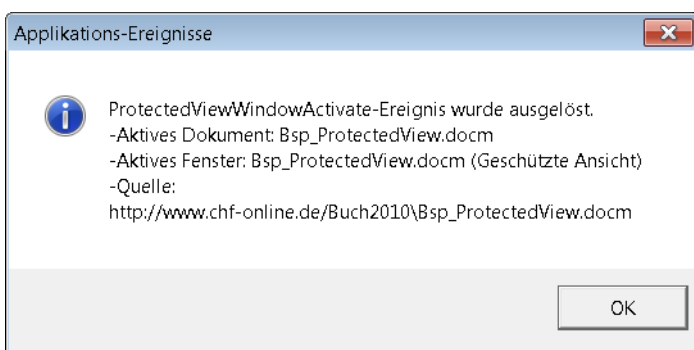
Tabelle 8.13 Der Parameter der Ereignisprozedur *ProtectedViewWindowActivate*

Parameter	Beschreibung
PvWindow	Das geschützte Ansichtsfenster, das aktiviert wird

Listing 8.24 Beispiel zur Ereignisprozedur *ProtectedViewWindowActivate*, das den aktuellen Dateinamen und die Quelle angibt

```
Private Sub objApp_ProtectedViewWindowActivate(ByVal PvWindow As ProtectedViewWindow)
    Dim strMSG As String
    Const c_Title As String = "Applikations-Ereignisse "
    strMSG = "ProtectedViewWindowActivate-Ereignis wurde ausgelöst." & vbCrLf & _
        "-Dokumentname: " & PvWindow.Document.Name & vbCrLf & _
        "-Fenstertitel: " & PvWindow.Caption & vbCrLf & _
        "-Quelle: " & PvWindow.SourcePath & Application.PathSeparator & PvWindow.SourceName & vbCrLf
    MsgBox strMSG, vbInformation, c_Title
End Sub
```

Abbildg. 8.21 Ausgabe der Informationen zum aktiven geschützten Ansichtsfenster



ProtectedViewWindowOpen

Das Ereignis *ProtectedViewWindowOpen* tritt ein, wenn Word ein Dokument in einem geschützten Ansichtsfenster öffnet. Dabei wird zuerst von Word ein geschütztes Ansichtsfenster erstellt, in dem das Dokument dann geöffnet wird. Daher wird beim Öffnen eines Dokuments aus einer nicht vertrauenswürdigen Quelle zuerst das *ProtectedViewWindowActivate*-Ereignis und erst anschließend das *ProtectedViewWindowOpen*-Ereignis ausgelöst.

Der Parameter wird in Tabelle 8.14 erläutert. Die Syntax lautet:

```
ProtectedViewWindowOpen(ByVal PvWindow As ProtectedViewWindow)
```

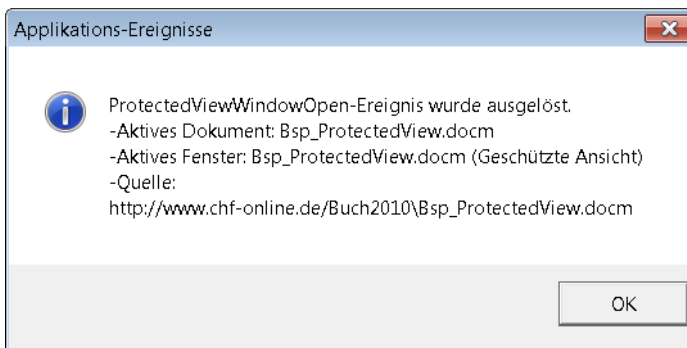
Tabelle 8.14 Der Parameter der Ereignisprozedur *ProtectedViewWindowOpen*

Parameter	Beschreibung
PvWindow	Das geschützte Ansichtsfenster, in dem das Dokument geöffnet wird

Listing 8.25 Beispiel zur Ereignisprozedur *ProtectedViewWindowOpen*, das den aktuellen Dateinamen und die Quelle angibt

```
Private Sub objApp_ProtectedViewWindowOpen(ByVal PvWindow As ProtectedViewWindow)
    Dim strMSG As String
    Const c_Title As String = "Applikations-Ereignisse "
    strMSG = "ProtectedViewWindowOpen-Ereignis wurde ausgelöst." & vbCrLf & _
        "-Dokumentname: " & PvWindow.Document.Name & vbCrLf & _
        "-Fenstertitel: " & PvWindow.Caption & vbCrLf & _
        "-Quelle: " & PvWindow.SourcePath & Application.PathSeparator & _
        PvWindow.SourceName & vbCrLf
    MsgBox strMSG, vbInformation, c_Title
End Sub
```

Abbildg. 8.22 Ausgabe der Informationen zum geöffneten Dokument im geschützten Ansichtsfenster



ProtectedViewWindowBeforeEdit

Das Ereignis *ProtectedViewWindowBeforeEdit* tritt ein, wenn in der Hinweiszeile auf die Schaltfläche *Bearbeitung aktivieren* geklickt wird (Abbildung 8.20).

Die Parameter werden in Tabelle 8.15 erläutert. Die Syntax lautet:

```
ProtectedViewWindowBeforeEdit(ByVal PvWindow As ProtectedViewWindow, Cancel As Boolean)
```

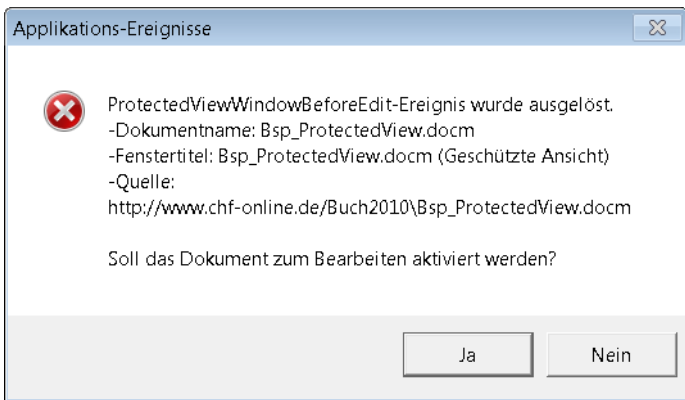
Tabelle 8.15 Die Parameter der Ereignisprozedur *ProtectedViewWindowBeforeEdit*

Parameter	Beschreibung
PvWindow	Das geschützte Ansichtsfenster, das das Dokument enthält, das zur Bearbeitung aktiviert wird
Cancel	False , wenn das Ereignis eintritt. Wenn die Ereignisprozedur dieses Argument auf True festlegt, wird die Bearbeitung für das Dokument nicht aktiviert

Listing 8.26 Beispiel zur Ereignisprozedur *ProtectedViewWindowBeforeEdit*, bei dem nur bei Bestätigung mit *Ja* das Dokument zur Bearbeitung aktiviert wird

```
Private Sub objApp_ProtectedViewWindowBeforeEdit(ByVal PvWindow As ProtectedViewWindow,
Cancel As Boolean)
Dim strMSG As String
Dim ret As Integer
Const c_Title As String = "Applikations-Ereignisse "
strMSG = "ProtectedViewWindowBeforeEdit-Ereignis wurde ausgelöst." & vbCrLf & _
" -Dokumentname: " & PvWindow.Document.Name & vbCrLf & _
" -Fenstertitel: " & PvWindow.Caption & vbCrLf & _
" -Quelle: " & PvWindow.SourcePath & Application.PathSeparator & _
PvWindow.SourceName & vbCrLf & vbCrLf & _
"Soll das Dokument zum Bearbeiten aktiviert werden?"
ret = MsgBox(strMSG, vbCritical + vbYesNo, c_Title)
If ret = vbNo Then Cancel = True ' Nicht zum Bearbeiten aktivieren
End Sub
```

Abbildg. 8.23 Abfrage, ob das geschützte Dokument zum Bearbeiten aktiviert werden soll



ProtectedViewWindowDeactivate

Das Ereignis *ProtectedViewWindowDeactivate* tritt ein, wenn Word ein geschütztes Ansichtsfenster deaktiviert oder ein Dokument in einem geschützten Ansichtsfenster schließt. Der Parameter wird in Tabelle 8.16 erläutert. Die Syntax lautet:

```
ProtectedViewWindowDeactivate(ByVal PvWindow As ProtectedViewWindow)
```

Tabelle 8.16 Der Parameter der Ereignisprozedur *ProtectedViewWindowDeactivate*

Parameter	Beschreibung
PvWindow	Das geschützte Ansichtsfenster, das deaktiviert wird

Listing 8.27 Beispiel zur Ereignisprozedur *ProtectedViewWindowDeactivate*, das den aktuellen Dateinamen und die Quelle angibt

```
Private Sub objApp_ProtectedViewWindowDeactivate(ByVal PvWindow As ProtectedViewWindow)
    Dim strMSG As String
    Const c_Title As String = "Applikations-Ereignisse "
    strMSG = "ProtectedViewWindowDeactivate-Ereignis wurde ausgelöst." & vbCrLf & _
        "-Dokumentname: " & PvWindow.Document.Name & vbCrLf & _
        "-Fenstertitel: " & PvWindow.Caption & vbCrLf & _
        "-Quelle: " & PvWindow.SourcePath & Application.PathSeparator & _
        PvWindow.SourceName & vbCrLf
    MsgBox strMSG, vbInformation, c_Title
End Sub
```

ProtectedViewWindowBeforeClose

Das Ereignis *ProtectedViewWindowBeforeClose* tritt ein, bevor ein Dokument in einem geschützten Ansichtsfenster geschlossen wird.

Die Parameter werden in Tabelle 8.17 erläutert. Die Syntax lautet:

```
ProtectedViewWindowBeforeClose(ByVal PvWindow As ProtectedViewWindow, ByVal CloseReason As Long, Cancel As Boolean)
```

Tabelle 8.17 Die Parameter der Ereignisprozedur *ProtectedViewWindowBeforeClose*

Parameter	Beschreibung
PvWindow	Das geschützte Ansichtsfenster, das das Dokument enthält, das geschlossen werden soll
CloseReason	Gibt die Art an, wie das Dokument geschlossen wurde (<i>WdProtectedViewCloseReason</i> -Enumeration)
Cancel	False , wenn das Ereignis eintritt. Wenn die Ereignisprozedur dieses Argument auf True festlegt, wird die Bearbeitung für das Dokument nicht aktiviert.

Listing 8.28 Beispiel zur Ereignisprozedur *ProtectedViewWindowBeforeClose*, bei dem nur bei Bestätigung mit *Ja* das Ansichtsfenster geschlossen wird

```
Private Sub objApp_ProtectedViewWindowBeforeClose(ByVal PvWindow As ProtectedViewWindow,
    ByVal CloseReason As Long, Cancel As Boolean)
    Dim strMSG As String
    Dim ret As Integer
    Dim strReason As String
    Const c_Title As String = "Applikations-Ereignisse "
    strMSG = "ProtectedViewWindowBeforeClose-Ereignis wurde ausgelöst." & vbCrLf & _
        "-Dokumentname: " & PvWindow.Document.Name & vbCrLf & _
        "-Fenstertitel: " & PvWindow.Caption & vbCrLf & _
        "-Quelle: " & PvWindow.SourcePath & Application.PathSeparator & _
        PvWindow.SourceName & vbCrLf & vbCrLf & _
        "Soll das geschützte Ansichtsfenster wirklich geschlossen werden?"
    ret = MsgBox(strMSG, vbCritical + vbYesNo, c_Title)
    If ret = vbNo Then
        Cancel = True ' Nicht schließen
    End If
End Sub
```

Listing 8.28 Beispiel zur Ereignisprozedur *ProtectedViewWindowBeforeClose*, bei dem nur bei Bestätigung mit *Ja* das Ansichtsfenster geschlossen wird (*Fortsetzung*)

```
Else
    Select Case CloseReason
    Case 0
        strReason = "Das Dokument wurde normal geschlossen."
    Case 1
        strReason = "Das Fenster wurde geschlossen, nachdem der Benutzer "
        "in der geschützten Ansicht auf eine der Schaltflächen "
        "Bearbeitung aktivieren und Trotzdem bearbeiten geklickt hat."
    Case 2
        strReason = "Das Fenster wurde geschlossen, weil es von der Anwendung "
        "zwangsweise geschlossen wurde oder nicht mehr reagiert hat."
    End Select
    MsgBox "Die geschützte Ansicht wurde aus folgendem Grund geschlossen: " & _
        vbCrLf & strReason, vbInformation, c_Title
End If
End Sub
```

ProtectedViewWindowSize

Das Ereignis *ProtectedViewWindowSize* tritt ein, wenn Word ein neues geschütztes Ansichtsfenster erstellt oder ein geschütztes Ansichtsfenster in der Größe verändert wird. Der Parameter wird in Tabelle 8.18 erläutert. Die Syntax lautet:

```
ProtectedViewWindowActivate(ByVal PvWindow As ProtectedViewWindow)
```

Tabelle 8.18 Der Parameter der Ereignisprozedur *ProtectedViewWindowSize*

Parameter	Beschreibung
PvWindow	Das geschützte Ansichtsfenster, das erzeugt bzw. in der Größe geändert wird

Listing 8.29 Beispiel zur Ereignisprozedur *ProtectedViewWindowSize*, das den aktuellen Dateinamen und die Quelle angibt

```
Private Sub objApp_ProtectedViewWindowSize(ByVal PvWindow As ProtectedViewWindow)
    Dim strMSG As String
    Const c_Title As String = "Applikations-Ereignisse "
    strMSG = "ProtectedViewWindowSize-Ereignis wurde ausgelöst." & vbCrLf & _
        "-Dokumentname: " & PvWindow.Document.Name & vbCrLf & _
        "-Fenstertitel: " & PvWindow.Caption & vbCrLf & _
        "-Quelle: " & PvWindow.SourcePath & Application.PathSeparator & _
        PvWindow.SourceName & vbCrLf
    MsgBox strMSG, vbInformation, c_Title
End Sub
```

Ereignis-Ablaufreihenfolge in der geschützten Ansicht

Die Ablaufreihenfolge beim Öffnen eines Dokuments in der geschützten Ansicht und Aktivieren zur Bearbeitung ist folgende:

1. `ProtectedViewWindowSize`
Das geschützte Ansichtsfenster wird erzeugt
2. `ProtectedViewWindowActivate`
Das geschützte Ansichtsfenster wird aktiviert
3. `ProtectedViewWindowOpen`
Das Dokument wird im geschützten Ansichtsfenster geöffnet
4. `ProtectedViewWindowBeforeEdit`
Das geschützte Ansichtsfenster soll zur Bearbeitung aktiviert werden
5. `ProtectedViewWindowDeactivate`
Das geschützte Ansichtsfenster wird durch das Aktivieren zur Bearbeitung deaktiviert
6. `WindowActivate`
Das `WindowActivate`-Ereignis für Standard-Word-Fenster wird ausgelöst (siehe den Abschnitt »`WindowActivate`« ab Seite 464)
7. `ProtectedViewWindowBeforeClose`
Das geschützte Ansichtsfenster wird durch das Aktivieren zur Bearbeitung geschlossen

Durch das Aktivieren des Dokuments zur Bearbeitung wird also das geschützte Ansichtsfenster geschlossen und das Dokument kann in einem normalen Fenster bearbeitet werden.

CD-ROM Die Beispieldatei *Kap_08_Bsp_2010.docm* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap08*.

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 19), stehen diese Automatisierungscodes auch außerhalb eines Word-VBA-Projekts zur Verfügung. Sie können beispielsweise in COM-Add-Ins sowie VSTO-Projekten benutzt werden.

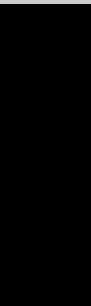
- Dieses Kapitel stellt auf Seite 456 sowie auf Seite 459 eine Übersicht der Dokument- bzw. Anwendungsereignisse vor
- Wie das Klassenmodul, das die Ereignisprozeduren enthält, erstellt und initialisiert wird, steht auf Seite 461 ff. beschrieben
- Die Anwendungs- sowie Dokumentereignisse sind ab Seite 464 näher erläutert
- Die Seriendruckereignisse werden ab Seite 478, gefolgt von einem praxisnahen Beispiel ab Seite 479, behandelt
- Abschließend, beginnend auf Seite 484, werden die in Word 2010 neu hinzugekommenen *ProtectedViewWindow*-Ereignisse vorgestellt

Teil C

Steuern und gesteuert werden

In diesem Teil:

Kapitel 9	Grundlagen der Fernsteuerung	495
Kapitel 10	Word von anderen Umgebungen aus steuern	513
Kapitel 11	Andere Programme von Word aus steuern	573
Kapitel 12	Eingebettete Objekte	605



Kapitel 9

Grundlagen der Fernsteuerung

Steuern und gesteuert werden

In diesem Kapitel:

Verweise auf externe Bibliotheken im VB-Editor	496
Early versus Late Binding	503
Zusammenfassung	511

Im Teil B dieses Buchs wurden das Objektmodell von Word sowie die Dokument- und Programmereignisse näher vorgestellt. In diesem Teil erläutern wir nun das Zusammenspiel von verschiedenen Applikationen:

- In diesem 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. Die »Visual Studio Tools for Office« werden vorgestellt.
- 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 sogenannter 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 for Applications
- Microsoft Word 14.0 Object Library
- Die zugehörige Dokumentvorlage

- OLE-Automation
- Microsoft Office 14.0 Object Library

Die beiden letzten Einträge sind optional und können bei Nichtgebrauch deaktiviert werden. Der Verweis auf die »Microsoft Forms 2.0 Object Library« wird automatisch gesetzt, sobald im Projekt eine UserForm eingefügt wird.

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 (XP) – 10.0; Office 2003 – 11.0; Office 2007 – 12.0, Office 2010 – 14.0.

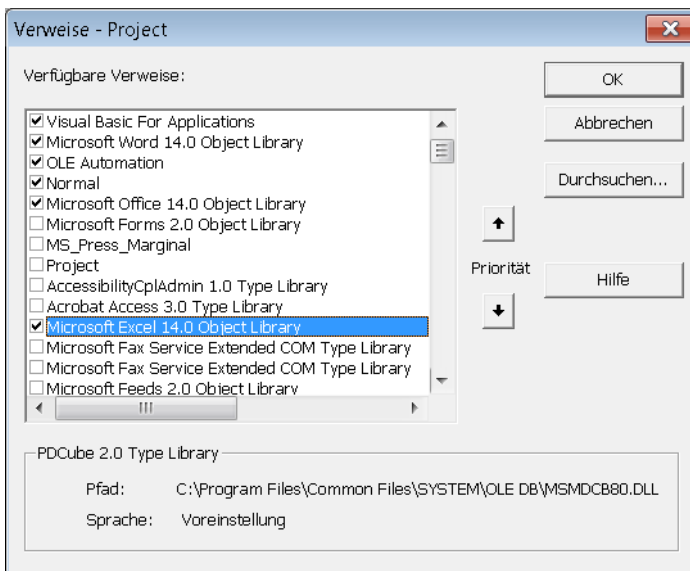


Verweis
einfügen

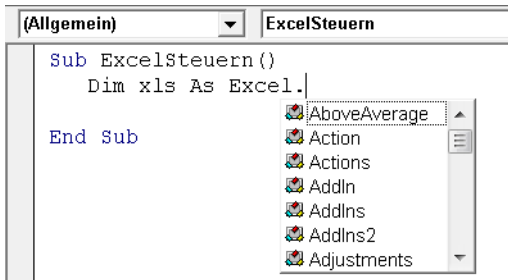
Normalerweise werden die Verweise während der Entwicklung eines Programms manuell innerhalb des Editors gesetzt. Rufen Sie hierzu in der Visual Basic-Entwicklungsumgebung (VB-Editor) den Menübefehl *Extras/Verweise* auf und aktivieren Sie, wie in Abbildung 9.1 ersichtlich, im Listenfeld *Verfügbare Verweise* die benötigte Bibliothek.

Wie Verweise auch dynamisch während der Laufzeit des Programms gesetzt werden können, ist in Kapitel 20 detailliert beschrieben.

Abbildg. 9.1 Fügen Sie einen Verweis auf die *Microsoft Excel 14.0 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 14.0 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 14.0 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
End Sub
```

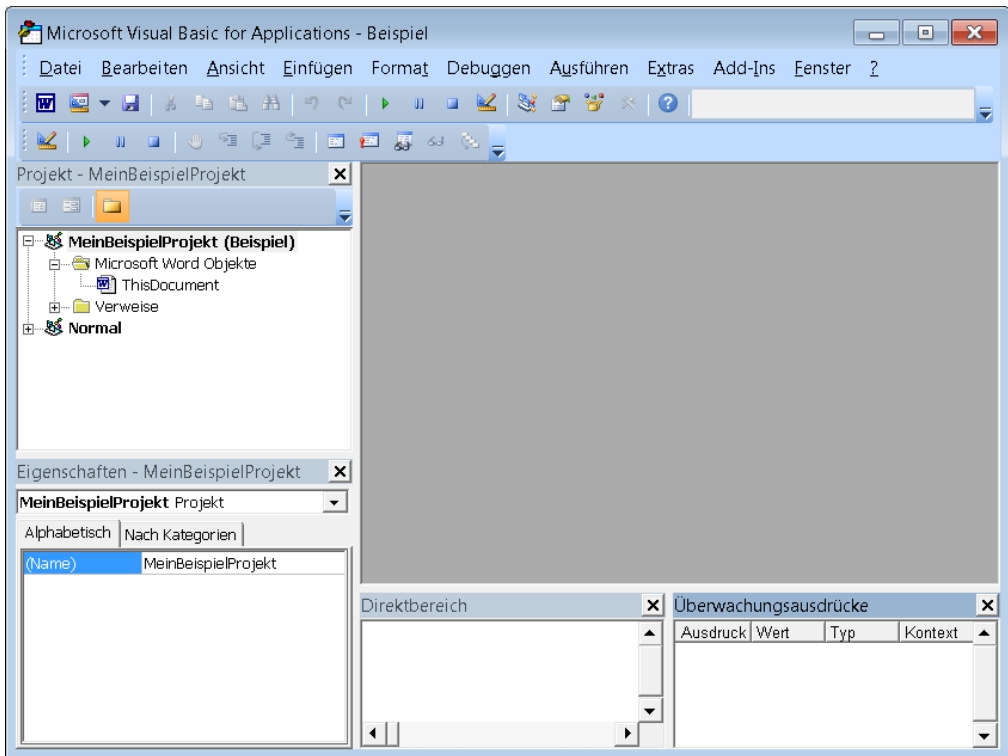
Listing 9.1 Erstellen einer Excel-Instanz und diese steuern (Fortsetzung)

```
Set wkb = Nothing
Set xls = Nothing
End Sub
```

In Abbildung 9.1 erkennen Sie unter *Verfügbare Verweise* den Eintrag »MS_Press_Marginal«. Es handelt sich dabei um ein geladenes Add-In (.dotm-Datei), das Makros enthält. Auf dieses Add-In kann ebenfalls ein Verweis gesetzt werden, der es ermöglicht, die als »Public« bezeichneten 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 (.dotm-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 20 wird gezeigt, wie fehlende Verweise dynamisch korrigiert werden können.

Einbinden von zusätzlichen Steuerelementen

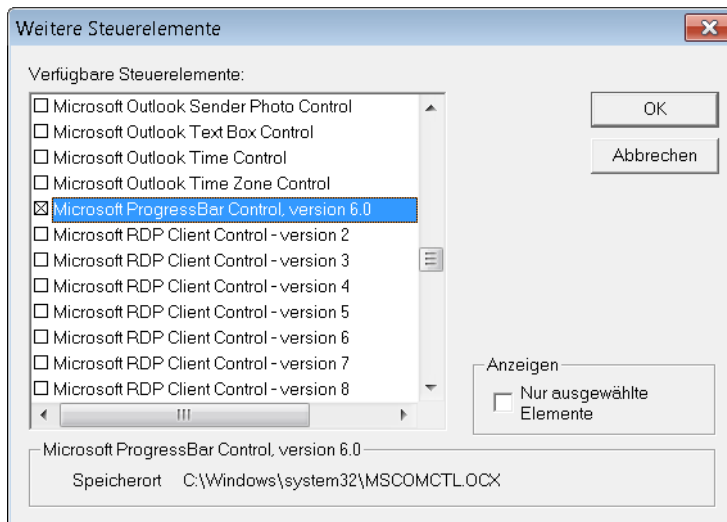
Grundsätzlich können neben den bestehenden Steuerelementen, die durch die *Microsoft Forms 2.0 Object Library* zur Verfügung gestellt werden, weitere Komponenten eingebunden werden.



Steuerelement
einbinden

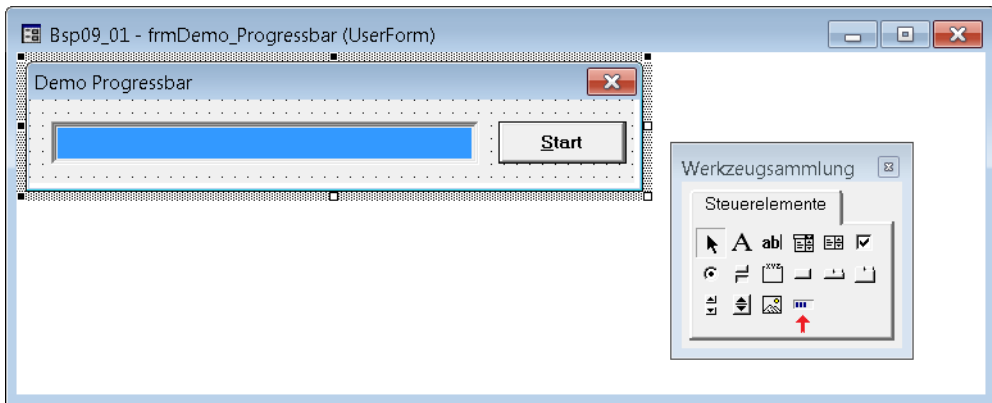
Dazu öffnen Sie innerhalb der VB-Entwicklungsumgebung ein Formular und wählen anschließend den Menübefehl *Extras/Zusätzliche Steuerelemente*. Im Listenfeld *Verfügbare Steuerelemente* aktivieren Sie nun, wie in Abbildung 9.4 ersichtlich, das Kontrollkästchen des gewünschten Steuerelements. Die Abbildung 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 im Projekt 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.

CD-ROM Das zusätzliche Steuerelement ist in der Datei *Mscocomctl.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.

ACHTUNG Unter der 64-Bit-Version von Word 2010 kommt es beim Ausführen des Makros zu einer Fehlermeldung (»Ein Objekt konnte nicht geladen werden, da es auf diesem Computer nicht verfügbar ist« oder einer ähnlich lautenden Meldung). Dieser Fehler beruht auf dem Umstand, dass in der 64-Bit-Version von Word 2010 keine 32-Bit ActiveX-Elemente geladen werden können.



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>)

Seit Office 2007 können in den Optionen die Sicherheitsstufen für ActiveX-Elemente gesetzt werden. Wählen Sie dazu *Datei/Optionen/Sicherheitscenter*, betätigen Sie die Schaltfläche *Einstellungen für das Sicherheitscenter* und wechseln in die Kategorie *ActiveX-Einstellungen*.

CD-ROM Das dargestellte Beispiel mit dem zusätzlichen Steuerelement finden Sie in der Beispieldatei *Bsp09_01.docm* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap09*.

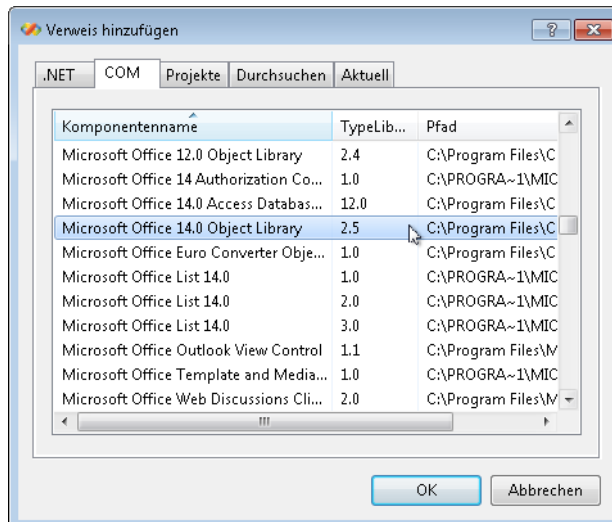
Verweise in Visual Studio 2008



Microsoft hat .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 Objektbibliothek 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 2010, Office 2007, Office 2003 sowie Office XP stellt Microsoft »Primary Interop Assemblies« (PIAs) zur Verfügung, die für diese Aufgabe optimiert sind. Außer für die Version XP gehören die PIAs zum Lieferumfang, können aber erst installiert werden, wenn .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 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« ab Seite 510 kurz vorgestellt.

Etwas erleichtert wird die Problematik der Verteilung und der Versionsabhängigkeit der PIAs für Entwickler, die mit Visual Studio 2010 arbeiten. Diese Version kann die für das Projekt benötigten Objektmodell-Datentypen im Projekt einbetten, sodass das Vorhandensein der versionspezifischen (P)IAs nicht notwendig ist. Mehr zu diesem Thema finden Sie unter den folgenden Links:

- <http://blogs.msdn.com/b/somasegar/archive/2009/01/10/office-client-developer-enhancements-with-vs-2010.aspx>
- [http://msdn.microsoft.com/en-us/library/ee342218\(VS.100\).aspx](http://msdn.microsoft.com/en-us/library/ee342218(VS.100).aspx)
- [http://msdn.microsoft.com/en-us/library/bb383815\(VS.100\).aspx](http://msdn.microsoft.com/en-us/library/bb383815(VS.100).aspx)
- <http://channel9.msdn.com/learn/courses/VS2010/ManagedLanguages/Dev10Office/Removing-the-PIA-Dependence-from-the-Application/>
- [http://msdn.microsoft.com/en-us/library/ee317478\(VS.100\).aspx](http://msdn.microsoft.com/en-us/library/ee317478(VS.100).aspx)

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 (ISBN-13: 978-0-7356-2132-9).

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 im VB-Editor« ab Seite 496 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 14.0 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
```

CD-ROM Das dargestellte Beispiel finden Sie in der Beispieldatei *Bsp09_02.docm* 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, kann dieser Verweis dynamisch auf die Version 14.0 angepasst werden, wenn das Makro auf einer Arbeitsstation mit installiertem Word 2010 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, 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 olAnw As Object
    Dim olOrdner As Object
    Dim olKontakt As Object
    Dim olItem As Object

    'Neues Dokument erzeugen
    Set doc = Application.Documents.Add

    'Outlook-Instanz setzen
    Set olAnw = CreateObject("Outlook.Application")

    'Kontakte-Ordner setzen
    Set olOrdner = olAnw.Session.GetDefaultFolder(10)

    'Einträge setzen
    Set olItem = olOrdner.Items

    'Alle Kontakte auflisten
    For Each olKontakt In olItem
        If olKontakt.Class = 40 Then
            doc.Range.InsertAfter olKontakt.FileAs & vbVerticalTab
        End If
    Next olKontakt
End Sub
```

Listing 9.3 Auflisten aller Kontakte aus Outlook unter Verwendung von Late Binding (Fortsetzung)

```

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« ab Seite 507 wird erläutert, 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 Compilerkonstante gesteuert werden.

```
#Const EARLYBINDING = False 'oder True
```

Wird der Wert der Compilerkonstante geändert, muss zusätzlich der Verweis auf die *Microsoft Outlook 14.0 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

```

Listing 9.4 Die Vorteile von Early und Late Binding, kombiniert im selben Programm (Fortsetzung)

```
#If EARLYBINDING Then
'Wird mit Early Binding gearbeitet, muss der Verweis auf die
'Microsoft Outlook 14.0 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 Compilerkonstanten 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
```

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.

CD-ROM Das dargestellte Beispiel finden Sie in der Beispieldatei *Bsp09_03.docm* 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 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
```

Listing 9.5 Late Binding in VB.NET mit *Option Strict* ein- sowie ausgeschaltet (Fortsetzung)

```
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>)

HINWEIS **Die Fernsteuerung und *Klick-und-los* (Click-to-Run) Versionen**

Mit Office 2010 hat Microsoft eine neue Art von Office veröffentlicht: sie wird als "Klick-und-los" bezeichnet. Der Benutzer lädt sie vom Internet herunter. Was er meistens nicht realisiert ist, dass *Klick-und-los* in einer virtuellen Umgebung läuft, statt direkt auf dem Rechner. Dies macht die Fernsteuerung der Office-Anwendungen von außerhalb der virtuellen Umgebung unmöglich. Dieses Verhalten ist von Microsoft gewollt; somit läuft Office »abgesichert«. Mehr Informationen finden Sie auf: <http://support.microsoft.com/kb/2028653/de>, <http://support.microsoft.com/kb/982431> sowie <http://support.microsoft.com/kb/983266>.

Klick-und-los wird dem privaten Benutzer sowie kleinen Geschäften empfohlen. Meistens sind die Benutzer sich dieser Funktionalitätsbegrenzung nicht bewusst. Falls die voll lauffähige Version doch vorzuziehen ist, kann diese nachgeladen werden. Mehr Informationen steht auf <http://office.microsoft.com/en-us/excel-help/click-to-run-switch-to-using-an-msi-based-office-edition-HA101850538.aspx>.

CD-ROM Das Beispielprojekt finden Sie in der Beispieldatei *Kap09_VB.zip* auf der CD-ROM zum Buch im Ordner `\Beispiele\Kap09\VB.NET`.

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 496)
- Im Weiteren wurde gezeigt, wie zusätzliche Steuerelemente eingebunden und auf einem Userform-Objekt verwendet werden können (Seite 500)
- Dann wurden die Themen Verweise und Interop Assemblies in Visual Studio .NET 2008 vorgestellt (Seite 502)
- Es wurden die Vor- und Nachteile von Early Binding gegenüber von Late Binding erläutert (Seite 503). Und es wurde aufgezeigt, wie man sich die Vorteile beider Varianten zunutze machen kann (Seite 507)
- Abschließend wurde Late Binding innerhalb von .NET Framework kurz diskutiert (Seite 510)

Kapitel 10

Word von anderen Umgebungen aus steuern

In diesem Kapitel:

Fernsteuern von Microsoft Word	514
Programmzeilen zentral verwalten und gemeinsam nutzen (Add-Ins)	522
Fernsteuerung aus Office-fremden Umgebungen	527
Die Word-Anwendung über Visual Studio .NET fernsteuern	527
VSTO COM-Add-Ins	533
VSTO-Dokumentlösungen	551
Zusammenfassung	571

Dieses Kapitel beschäftigt sich mit dem Fernsteuern von Word. Es wird gezeigt, 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 2010 Tools for Office (VSTO) vorgestellt.

HINWEIS

Neben den auf einer Dokumentvorlage (*.dotm*) basierenden Add-Ins können alternativ sogenannte COM-Add-Ins zum Einsatz gelangen. Wie solche Add-Ins in Programmierumgebungen außerhalb von VSTO erzeugt und genutzt werden, übersteigt den Rahmen dieses Buchs. 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« ab Seite 520 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 Menüband über *Entwicklertools/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 im 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.dotm* 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.dotm* 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 sogenanntes 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.dotm 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

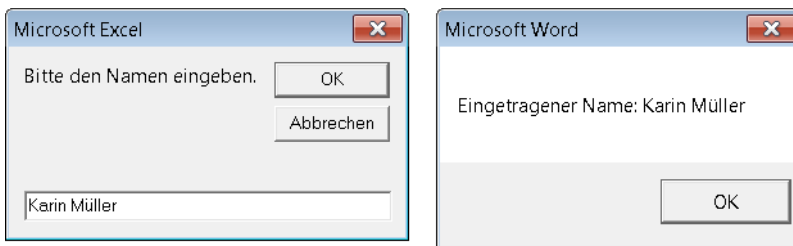
        Set docApp = Nothing
    End If
End Sub
```


CD-ROM

Die Prozedur *Demo_WordFernsteuernRun* befindet sich in der Datei *Bsp10_02.xlsm* 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.xlsm* und klicken in der Registerkarte *Entwicklertools* auf die Schaltfläche *Makros*. Wählen Sie anschließend in der Liste der verfügbaren Makros die Prozedur *Demo_WordFernsteuernRun* aus und klicken Sie auf die Schaltfläche *Ausführen*.

Die Funktionsweise des Makros ist schnell erklärt. Der Anwender soll seinen Namen eintragen. Dazu wird die Funktion *InputDialog* 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 gezeigten 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 (*.dotm*) gespeichert sein.

Als Voraussetzung, dass das Listing 10.1 fehlerfrei ausgeführt werden kann, muss die Datei *Bsp10_03.dotm* zuerst als Add-In zur Verfügung gestellt werden. Diese Voraussetzung ist jedoch

nicht zwingend nötig, denn ein sogenanntes Add-In kann zur Laufzeit aus irgendeinem beliebigen Ordner 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.xlsm*).

Listing 10.3 Zusätzliche Add-Ins können bei Bedarf dynamisch geladen und entladen werden

```
Const strADDIN As String = "C:\WordBuch\Beispiele\Kap10\Bsp10_04.dotm"
'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, 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.dotx* 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.dotx"

    #Const EARLYBINDING = False 'oder True

    'Variablen und Objekte anlegen
```

Listing 10.4 Aus Excel heraus Word fernsteuern und die Empfängeradresse in den Brief eintragen (Fortsetzung)

```

#If EARLYBINDING Then
    'Wird mit Early Binding gearbeitet, muss ein Verweis auf
    'die "Microsoft Word 14.0 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

```

Listing 10.4 Aus Excel heraus Word fernsteuern und die Empfängeradresse in den Brief eintragen (Fortsetzung)

```
MsgBox "Neue Instanz von Word konnte nicht erzeugt werden."  
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
```

CD-ROM

Die dargestellten Beispiele finden Sie in der Beispieldatei *Bsp10_02.xlsm*. Als zusätzliche Hilfsdateien werden die drei Dokumentvorlagen *Bsp10_03.dotm*, *Bsp10_04.dotm* und *Bsp10_Brief.dotx* 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.docx"  
Dim intAntwort As Integer  
  
System.Cursor = wdCursorWait
```

Listing 10.5 Ausdrucken eines Dokuments mittels Fernsteuerung von Word (Fortsetzung)

```

'Variablen und Objekte anlegen
Dim docApp As Word.Application
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
            intAntwort = MsgBox("Soll der Ausdruck erstellt werden?", vbYesNo)
            If intAntwort = vbYes Then
                .PrintOut Background:=False, Copies:=1,
                    Range:=wdPrintFromTo, From:="1", To:="1"
            End If
            .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.

CD-ROM Das dargestellte Beispiel finden Sie in der Beispieldatei *Bsp10_01.docm* 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, werden einzelne Funktionen und Prozeduren schon bald mehrmals verwendet. Als Beispiele können die Dateisystemoperationen 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 (.dotm) als Add-In



Vorlage
als Add-
In

Aus der Sicht von Word kann eine *.dotm*-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 das Menüband *Datei/Optionen/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 (siehe den Abschnitt »Makros von außen anstoßen« ab Seite 514).

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 über 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

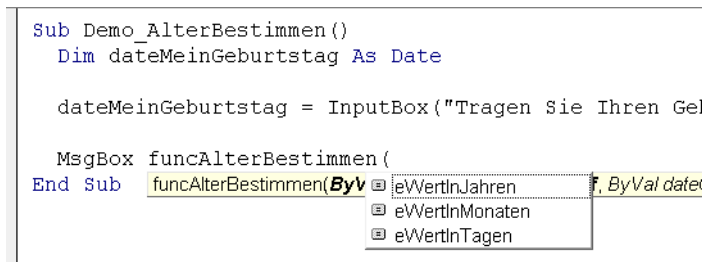
    dateMeinGeburtstag = InputBox("Tragen Sie Ihren Geburtstag ein.", , "24.12.2001")

    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 *.dotm*-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
```

Listing 10.7 Zentrale Funktion zur Berechnung des Alters (Fortsetzung)

```

    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)

```

CD-ROM

Das dargestellte Beispiel aus Listing 10.6 finden Sie in der Beispieldatei *Bsp10_06.docm*. Als zusätzliche Hilfsdatei wird die Datei *Bsp10_05.dotm* 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.

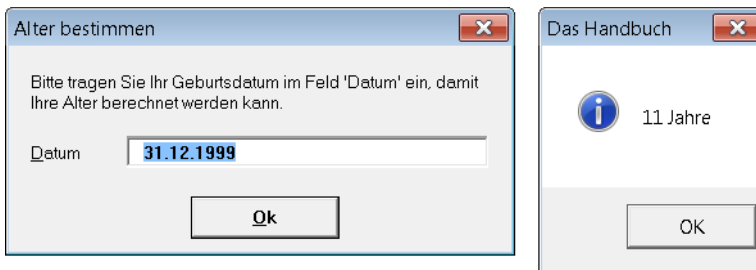
Listing 10.8 Indirekter Aufruf der UserForm über die allgemeine Funktion *funcUserFormAlterBestimmen*

```
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
```

CD-ROM

Das Beispiel aus Listing 10.8 finden Sie in der Beispieldatei *Bsp10_06.docm*. Als zusätzliche Hilfsdatei wird die Datei *Bsp10_05.dotm* 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 mithilfe 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 in den Vordergrund 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 dateMeinGeburstag As Date

    On Error Resume Next
    Set wdapp = GetObject(, "Word.Application")
    If Err.Number = 429 Then
        'Keine Word-Instanz vorhanden
        Exit Sub
    End If
    On Error GoTo 0
    'Bildschirmflackern minimieren
    wdapp.ScreenUpdating = False
    'Word in den Vordergrund 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.dotm").Path & "\" _
        & wdapp.AddIns("Bsp10_05.dotm").Name
    'Add-In unsichtbar öffnen
```

Listing 10.10 Prozeduren in einem Word-Add-In außerhalb von Word aufrufen (Fortsetzung)

```

Set doc = wdapp.Documents.Open(Filename:=sDateiName, _
    AddToRecentFiles:=False, Visible:=False)

'Funktion im ThisDocument-Modul
dateMeinGeburtstag = doc.AltersEingabe
wdapp.Tasks("Microsoft Excel").Activate
'Wird in Excel angezeigt
MsgBox doc.AlterBestimmen(2, dateMeinGeburtstag), vbInformation

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

```

CD-ROM

Das dargestellte Beispiel finden Sie in der Beispieldatei *Bsp10_07.xlsm*. Diese befindet sich auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap10*. Es ruft Prozeduren in der Vorlage *Bsp10_05.dotm* 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.

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-Dokuments 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 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.GetActiveObject` 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-Objektvariablen *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 (.NET): 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 (.NET): 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 *GetWordInstance_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 (NET): 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 Beendung 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, erschienen bei Microsoft Press, USA (ISBN-13: 978-0-7356-2132-9).

Listing 10.13 (.NET): 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 (.NET): COM-Ressourcen im .NET-Projekt freigeben (Fortsetzung)

```

{
    GC.Collect();
    GC.WaitForPendingFinalizers();
    GC.Collect();
    GC.WaitForPendingFinalizers();
}
}

```

CD-ROM Zu diesem Abschnitt sind Beispieldateien für C# (\C_Sharp\Kap10_CS.zip) sowie VB.NET (\VB_NET\Kap10_VB.zip) vorhanden. Diese befinden sich auf der CD-ROM zum Buch im Ordner \Beispiele\Kap10.

VSTO COM-Add-Ins



Office-Anwendungen unterstützen Add-Ins, welche die Benutzerschnittstelle und Funktionalität erweitern. Für Word und Excel gibt es sowohl »Dokument-« (im Abschnitt »Programmzeilen zentral verwalten und gemeinsam nutzen (Add-Ins)« ab Seite 522 vorgestellt) als auch »COM-Add-Ins«. Der Code eines Dokument-Add-Ins befindet sich in einem im Dokument gespeicherten VBA-Projekt und wird innerhalb der laufenden Anwendungsinstanz ausgeführt. Ein COM-Add-In besteht aus (mindestens) einer *dll*-Datei, die Word zur Laufzeit einbindet. Der Code wird außerhalb der Office-Anwendung, die er automatisiert, ausgeführt. Da es nicht möglich ist, eine solche DLL mit den in Office enthaltenen Programmierwerkzeugen zu erstellen, setzt dies eine andere Entwicklerumgebung voraus wie beispielsweise Delphi, Visual Basic 6.0 oder eine Ausgabe von Visual Studio .NET.

Ein COM-Add-In für Office basiert auf der Schnittstelle *IDTExtensibility2*, die zusammen mit Office 2000 verfügbar wurde. Die Vorlage *Gemeinsames Add-In*, Teil des Lieferumfangs von Visual Studio .NET, bietet dem .NET-Entwickler einen Anfangspunkt für die Erstellung von COM-Add-Ins basierend auf dieser Schnittstelle. Damit kann er COM-Add-Ins erstellen die gleichzeitig mehrere Office-Anwendungen sowie –Versionen unterstützen. Um mit einer spezifischen Anwendung zu arbeiten, muss der 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 prinzipiell 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*. VSTO_COM-Add-Ins können für Office 2003- bis Office 2010-Anwendungen erstellt werden.

HINWEIS

Visual Studio Tools for Office ist im Lieferumfang von ausgewählten Visual Studio Produkten enthalten. Für Visual Studio 2008 sowie 2010 ist es Teil der »Professional« und höheren Ausgaben. Alle Ausgaben von Word 2007 sowie Word 2010 hingegen enthalten die notwendigen Teile.

VSTO 2005 und 2008 enthalten Add-In-Vorlagen für Office 2003 (.NET Framework 3.0 oder 3.5) sowie 2007 (.NET Framework 3.5). VSTO 2010 stellt Vorlagen für Office 2007 und 2010 (.NET Framework 3.5 oder 4.0) zur Verfügung. Eine neuere Version von Visual Studio kann ein in einer

älteren Version erstelltes Projekt öffnen und konvertieren. Sofern die passende .NET Framework-Version sowie VSTO Runtime auf dem Rechner installiert ist, können Projekte, die für eine ältere Version von Word erstellt wurden, unter einer neueren laufen, ohne neu kompiliert werden zu müssen. Wie immer ist ein umfangreiches Testen der Softwarekombinationen empfehlenswert.

Die zwei Add-Ins-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 Domäne gesperrt. Abhilfe schafft nur ein sogenanntes *Shim*, ein mit C++ erstellter Teil, der dafür sorgt, dass das Add-In in eine eigene Domäne geladen wird.

VSTO-Add-Ins werden hingegen vom »Office Solution Loader« verwaltet und in getrennten Domänen geladen. Somit kann kein Add-In ein anderes zum Absturz bringen.

Wir stellen hier ein kleines Beispiel für Word 2007/2010 vor. Es veranschaulicht, wie die VSTO-Add-In-Vorlage den Umgang mit der Ribbon- und Custom Task Pane-Funktionalität (Menüband/Multifunktionsleiste bzw. frei definierbarer Aufgabenbereich) erleichtert. Ziel dieses Add-Ins ist es, die Seriendruckfunktionalität anzupassen und zu erweitern, wie in Abbildung 10.4 ersichtlich. Dabei werden nicht benötigte Steuerelemente aus dem Menüband 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.

HINWEIS

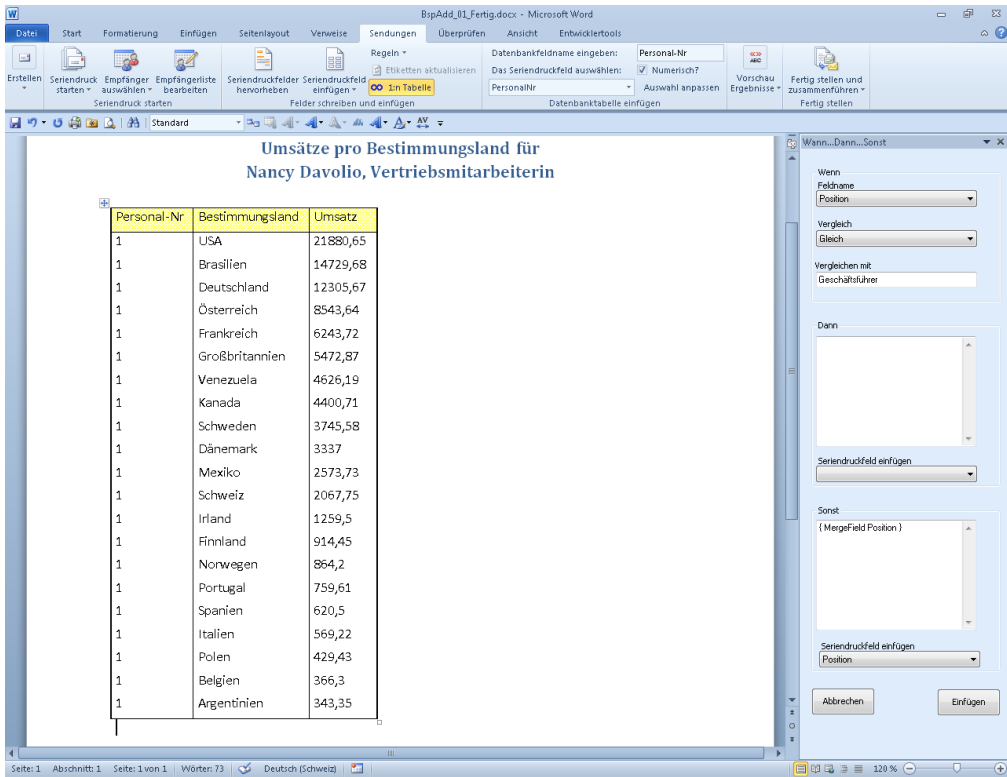
Im Gegensatz zu den Codebeispielen für die Word- und Office-Objektmodelle wird in den folgenden Teilen VB.NET verwendet. Da die von VSTO zur Verfügung gestellten Eigenschaften und Methoden den Regeln von .NET Framework entsprechen, erfolgt die Umwandlung in C# problemlos. Hingegen wird die Leserschaft, die aus der VBA-Ecke kommt, den VB-Codebeispielen besser folgen können, was den Einstieg in VSTO etwas erleichtern wird.

Das hier vorgestellte Beispiel basiert auf der Grundlage der Versionen 2010 von VSTO sowie Office. Die Grundlagen bleiben für VSTO 2008 mit Word 2007 die gleichen, nur die Befehlsfolgen in der Visual Studio-Umgebung unterscheiden sich teilweise.

HINWEIS

Die Verteilung von VSTO-Add-Ins kann sich komplex gestalten und wird hier nicht behandelt. Die Aufnahme von Office-Lösungen in der ClickOnce-Technologie hat die Sache aber erheblich erleichtert. Eine ausführliche Dokumentation steht auf den MSDN-Seiten zur Verfügung. Ein guter Anfangspunkt bietet diese Nachricht im englischsprachigen VSTO-Forum: <http://social.msdn.microsoft.com/Forums/en-US/vsto/thread/1666d2b0-a4d0-41e8-ad86-5eab3542de1e>.

Abbildg. 10.4 Die Registerkarte *Sendungen* wird angepasst und die Seriendruckfunktionalität erweitert



Das Anlegen eines VSTO-Add-In-Projekts

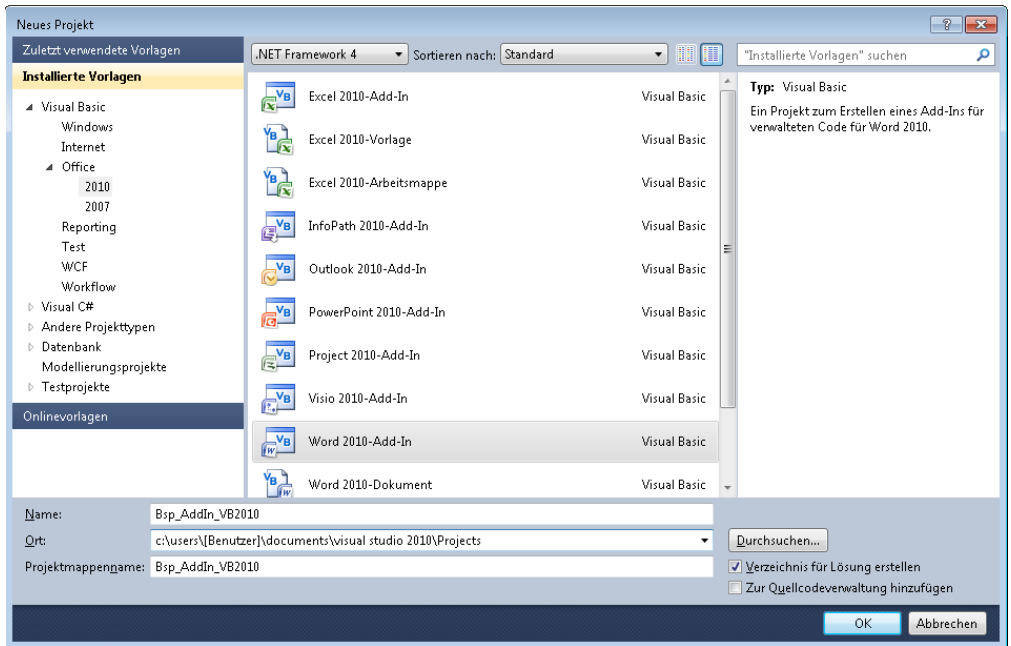
Nach erfolgreicher Installation von Visual Studio Tools for Office erscheinen, wie in Abbildung 10.5 ersichtlich, unter den Visual Basic- und C#-Ordern des Visual Studio-Dialogfelds *Neues Projekt* neue Ordner für *Office*.

Um ein VSTO-Add-In zu erstellen, wird im Visual Studio-Dialogfeld *Neues Projekt* der Ordner *Visual Basic* (oder *C#*) gewählt. Aus dem Unterordner *Office/[Version]* (Abbildung 10.5) wird der Eintrag *Word [Version]-Add-In* markiert und im unteren Teil des Dialogfelds ein eindeutiger Name für das Projekt eingegeben und der Speicherort festlegt.

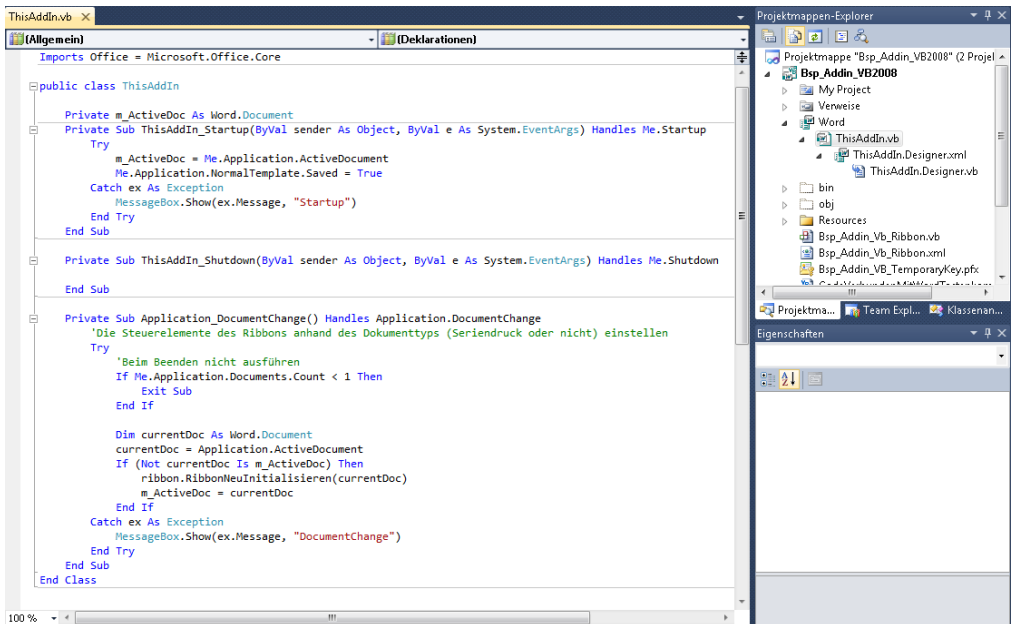
Die *ThisAddIn*-Klasse des Beispielprojekts ist in Abbildung 10.6 ersichtlich. Von allgemeinem Interesse ist die Zeile `Me.Application.NormalTemplate.Saved = True`. Beim Laden eines COM-Add-Ins werden unvermeidliche Änderungen in der Vorlage *Normal.dotm* vorgenommen. Diese Anweisung unterdrückt die störende Aufforderung, diese Änderungen zu speichern.

Visual Studio zeigt nach einigen Sekunden im Codefenster 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.

Abbildg. 10.5 Ein VSTO-Projekt anlegen



Abbildg. 10.6 Die *ThisAddin*-Klasse eines VSTO-Add-In-Projekts



VSTO-Add-In-Benutzerschnittstellen

Für Word 2007 wurde die Benutzerschnittstelle grundlegend überarbeitet. Symbolleisten gibt es seither keine mehr. Stattdessen werden Befehle im Menüband (Multifunktionsleiste) – auch Ribbon genannt – angeboten. Das Menüband wird nicht mehr über das Objektmodell *Commandbars* verwaltet, sondern in einer XML-Datei mit Callbacks definiert. Eine ausführliche Beschreibung hierzu finden Sie in Kapitel 16.

Neben dem Menüband 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 VSTO-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 Buchs. Für mehr Informationen wenden Sie sich bitte an die MSDN-Webseiten und -Foren. Gute Anfangspunkte bieten die Artikel unter <http://msdn.microsoft.com/en-us/library/bb264456.aspx> sowie [http://msdn.microsoft.com/en-us/library/bb608620\(VS.90\).aspx](http://msdn.microsoft.com/en-us/library/bb608620(VS.90).aspx).

In den folgenden Abschnitten werden die Erstellung und der Umgang mit den VSTO-Werkzeugen für diese zwei Schnittstellen näher vorgestellt.

Das Add-In mit einer Menüband-Erweiterung ergänzen

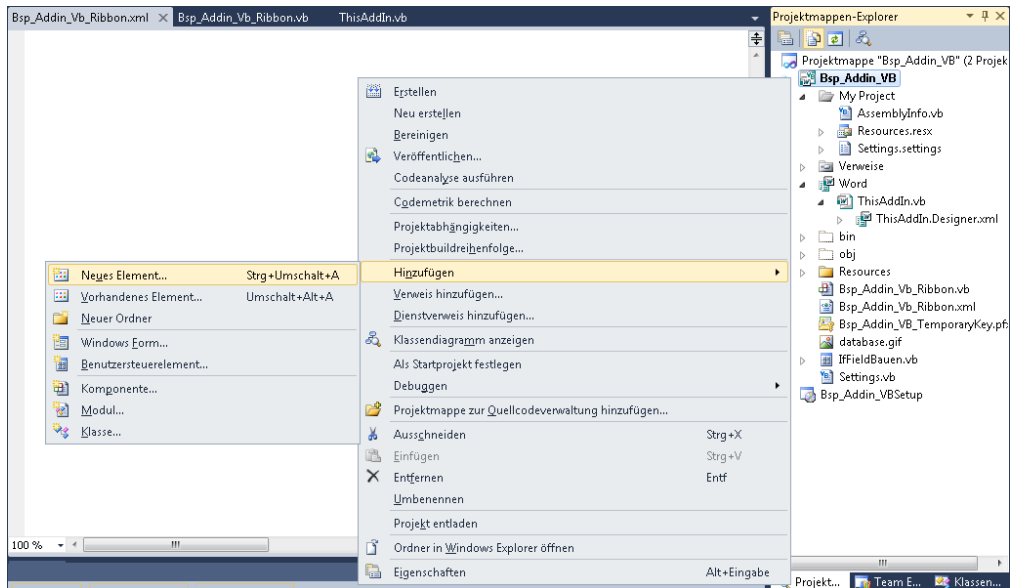
In Kapitel 16 ist beschrieben, wie eine Menüband-Erweiterung in ein Dokument oder eine Vorlage eingebunden wird. Das resultierende Menüband ist begrenzt auf diese Datei (oder, im Fall einer Vorlage, auf die mit der Vorlage verbundenen Dokumente).

Dagegen steht die Menüband-Erweiterung eines Add-Ins der ganzen Anwendung zur Verfügung und wird beim Laden des Add-Ins in das vorhandene Menüband integriert. Im Fall eines COM-Add-Ins ist die Datei mit dem definierenden XML integrierter Teil des Visual Studio-Projekts.

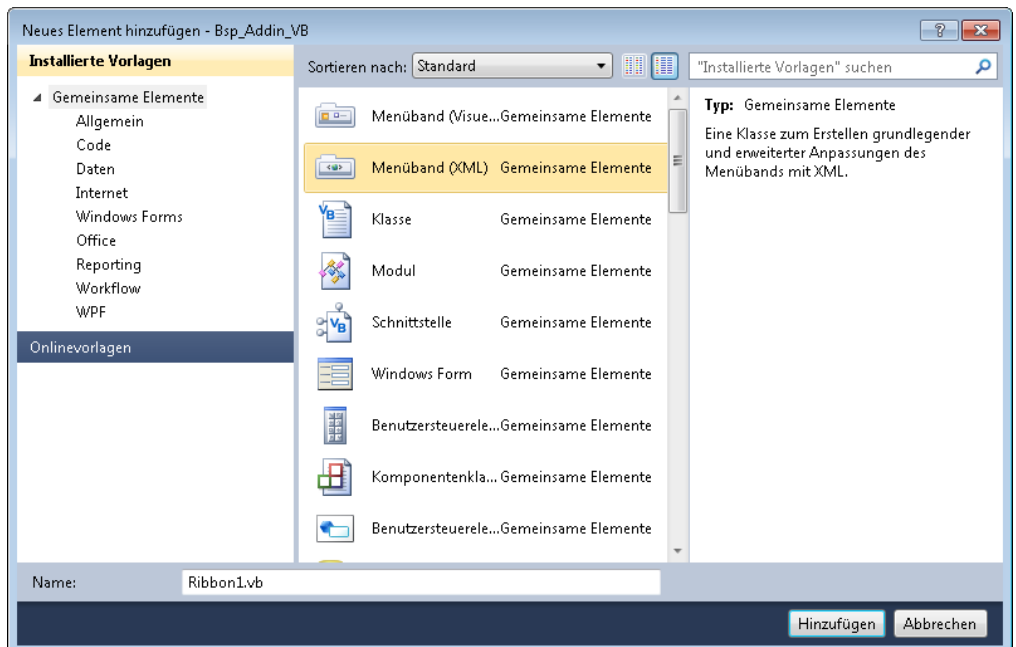
Um dem VSTO COM-Add-In eine Menüband-Erweiterung hinzuzufügen, gehen Sie wie folgt vor:

1. Im Fenster *Projektmappe-Explorer* klicken Sie mit rechter Maustaste auf den Projektnamen.
2. Wählen Sie im Kontextmenü den Untermenübefehl *Hinzufügen/Neues Element* (Abbildung 10.7).
3. Im Dialogfeld *Neues Element hinzufügen* wählen Sie den Eintrag *Menüband-(XML)* aus und geben unten einen aussagekräftigen Namen ein (Abbildung 10.8).

Abbildg. 10.7 Das Projekt um ein neues Element erweitern



Abbildg. 10.8 Dem Projekt eine Klasse für die Menüband-Erweiterung hinzufügen



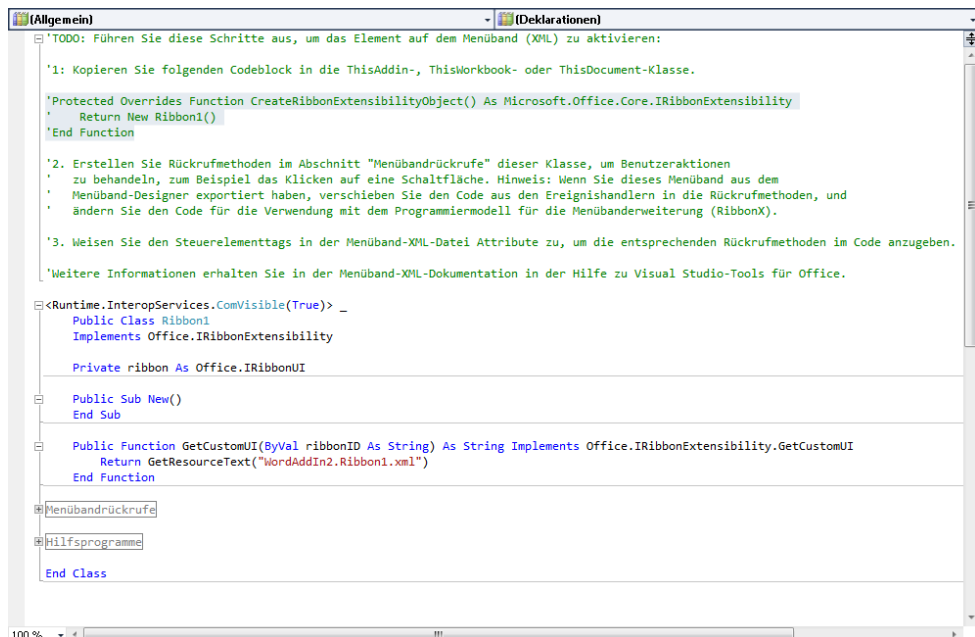
Nach Bestätigung des Dialogfelds erscheinen im Fenster *Projektmappen-Explorer* zwei neue Einträge mit dem im Dialogfeld eingegebenen Namen: eine Klasse (je nach Programmiersprache mit der Endung *vb* oder *cs*) sowie eine XML-Datei.

Die neu erstellte Klasse ist in Abbildung 10.9 erkennbar. VSTO hat unter anderem die folgenden Handlungen für Sie erledigt:

- Die Klasse für die Menüband-Erweiterung wurde erstellt. Sie enthält u.a. eine globale Variable für das Ribbon-Objekt, einen Abschnitt *Helpers* mit unterstützenden Funktionen, einen Abschnitt *Ribbon Callbacks* für Prozeduren für Menüband-Steuerelemente sowie die Funktion *GetCustomUI*. Diese Prozedur sorgt in einem COM-Add-In für das Laden des Menüband-Erweiterung-XMLs. Normalerweise wird es, wie hier, als Datei übergeben. Es wäre aber auch möglich, eine Zeichenkette mit dem gesamten XML-Code zurückzugeben.
- Zudem enthält die *Ribbon1*-Klasse eine auskommentierte *RequestService*-Funktion. Falls im Projekt noch keine Prozedur mit diesem Namen vorhanden ist (wie hier), sollten die Kommentarzeichen entfernt und die Prozedur in die *ThisAddin*-Klasse verschoben werden. (Sonst muss die vorhandene Prozedur um die Informationen ergänzt werden, die das ribbon-Objekt betreffen.)
- Die Grundstrukturen des Menüband-Erweiterungs-XMLs wurden in die XML-Datei geschrieben, mit Elementen für eine neue Gruppe in der Word-eigenen, integrierten Registerkarte *Add-Ins*.

Abbildg. 10.9

Die von VSTO erstellte Klasse für die Menüband-Erweiterung



HINWEIS

Die VSTO-Versionen 2008 und 2010 verfügen zudem über ein grafisches Werkzeug für die Erstellung von Menüband-Anpassungen. Dieses unterstützt jedoch nicht die ganze Palette der für Menüband-XML definierten Funktionalität, sondern nur vergleichsweise einfache Anpassungen. Es lohnt sich also, ein Projekt eingehend zu analysieren, bevor die Entscheidung

getroffen wird, ob eine »Schnelllösung« mit dem Werkzeug oder eine Menüband-Erweiterung mit XML das Richtige ist. Mehr über das grafische Werkzeug finden Sie im Abschnitt »VSTO-Dokumentlösungen« ab Seite 551 beschrieben.

Aufgabe der Menüband-Erweiterung

Das XML für unser Beispielprojekt befindet sich in Listing 10.14. Einzelheiten zu den darin verwendeten Steuerelementen und Attributen finden Sie in Kapitel 16. Hier stellen wir lediglich die Lösung sowie Besonderheiten des VSTO-Werkzeugs vor.

Listing 10.14 Das XML für die Menüband-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="GroupMailMergeStart" visible="false" />
        <group idMso="GroupMailMergeWriteInsertFields" visible="false" />
        <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">
            <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">
```

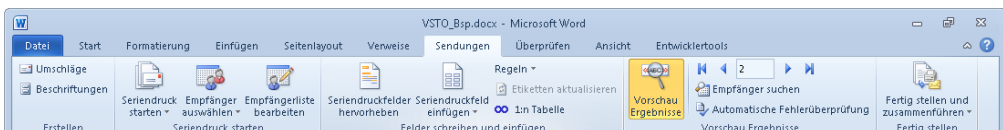

Listing 10.14 Das XML für die Menüband-Erweiterung des VSTO-Beispielprojekts (Fortsetzung)

```

        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>

```

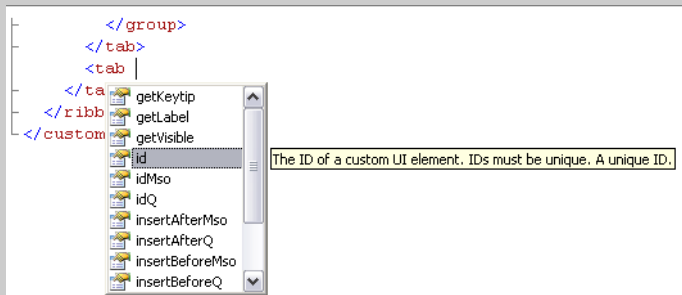
Aus dem Listing ist zu erkennen, dass die Word-eigenen Gruppen *Seriendruck starten* sowie *Felder schreiben und einfügen* ausgeblendet werden, da deren Funktionalität angepasst werden soll. Der Anfangsstand der Menüband-Erweiterung ist in Abbildung 10.10 ersichtlich.

Abbildg. 10.10 Die durch das COM-Add-In angepasste Registerkarte *Sendungen*

IntelliSense für die Menüband-Erweiterung

Wie Sie in Kapitel 16 lesen können, gestaltet sich das Erfassen des Menüband-XMLs mit einem einfachen Texteditor mühsam. Jede Einzelheit muss stimmen, sonst läuft es nicht. IntelliSense bei der Eingabe wäre eine unschätzbare Hilfe. Die in Visual Studio enthaltenen Werkzeuge erleichtern die Aufgabe erheblich, wie die Abbildung 10.11 veranschaulicht.

Abbildg. 10.11 IntelliSense für Menüband-XML in Visual Studio



Um IntelliSense beim Erstellen einer XML-Datei zu aktivieren, muss die XML-Datei auf ein Schema verweisen. Im Fall der Menüband-Erweiterung heißt das Schema *CustomUI.xsd*. Dieses Schema wird automatisch von VSTO 2008 und 2010 beim Hinzufügen einer Menüband-Erweiterung geladen.

Sie dürfen durchaus ein VSTO Add-In erstellen, zwecks Entwurfs einer Menüband-Erweiterung für die Einbindung in ein Dokument. Sobald das visuelle Ergebnis erreicht wurde, kann das XML kopiert und in die XML-Datei für das Dokument eingefügt werden.

Falls Sie mit einer kostenlosen »Express«-Ausgabe von Visual Studio arbeiten, kann das Schema manuell für die Arbeit mit einer angelegten XML-Datei geladen werden.

Anpassungen in der Registerkarte *Sendungen*

Die Gruppe *Seriendruck starten* wird ersetzt, um die Funktionalität der Befehle, die Datenquellen in das Dokument verbinden, so zu erweitern, dass die im Menüband-XML definierten Steuerelemente sich wie Word-eigene verhalten. Bestimmte Word-eigene Befehle stehen einzig Seriendruckdokumenten zur Verfügung. Um dies auch für selbst definierte Steuerelemente zu verwirklichen, muss die Menüband-Erweiterung beim Einbinden einer Datenquelle neu initialisiert werden, um Steuerelemente dynamisch verfügbar zu machen. Leider funktioniert das Ereignis *MailMergeDataSourceLoad* nicht wunschgemäß. Deshalb muss in der Callback-Prozedur *AddinMailMergeStartMenus_Click*, worauf alle Befehle dieser Dropdownliste verweisen, zuerst der Word-Befehl (der anhand des tag-Attributs des jeweiligen Steuerelements identifiziert wird) standardmäßig mithilfe der Methode *CommandBars.ExecuteMso* und anschließend die Neuinitialisierung der Menüband-Erweiterung ausgeführt werden (Listing 10.15).

HINWEIS

Die Prozedur *RibbonNeuinitialisieren* wird auch vom Ereignis *Application_DocumentChange* aufgerufen, wie in Abbildung 10.6 ersichtlich. Somit wird die Sperrung der Seriendruck-Befehle dynamisch für das aktuelle Dokument angepasst.

Listing 10.15 Die Menüband-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 zufriedenstellend 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« ab Seite 545 näher vorgestellt). Zudem wurden aus dieser Liste die Einträge *Nächster Datensatz wenn* sowie *Datensatz überspringen wenn* entfernt, da diese seit Jahren als Auslaufmodelle gelten und vom Benutzer nicht eingesetzt werden sollen.

Anstelle 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.

Grafische Symbole in die Menüband-Erweiterung einbinden

Neben der Beschriftung dieser letzterwähnten Schaltfläche befindet sich ein Symbol. Im Gegensatz zu einer in einem Dokument gespeicherten Menüband-Erweiterung (wie in Kapitel 16 beschrieben) müssen für die Menüband-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 Menüband-Erweiterung, ausgeführt. Die Funktion erhält von jedem Steuerelement der Menüband-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 Menüband-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 10.16 aufzeigt. (In diesem Beispiel wurde die Grafik dem Projekt als Ressource hinzugefügt und befindet sich im gleichen Ordner wie die anderen Projektdateien.)

Listing 10.16 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 ins Dokument verknüpfen. Falls der Benutzer dies erfolgreich tut, wird die Gruppe *Datenbanktabelle einfügen* (siehe Abbildung 10.4) 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 dynamisch verknüpften 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 den *editBox*- sowie *dropDown*-Steuerelementen gelesen. Einzelheiten entnehmen Sie bitte dem Beispielprojekt auf der CD-ROM zum Buch.

```
"SELECT * FROM 'Personalumsätze nach Land_SD' WHERE [Personal-Nr]={ Mergefield PersonalNr
}"
```

CD-ROM

Das Beispielprojekt befindet sich im Unterordner *Bsp_Addin_VB* der Datei *Bsp_VSTO_VB2010.zip* bzw. *Bsp_VSTO_VB2008.zip* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap10\VB.NET*. Auch darin finden Sie das Beispieldokument *BspAddin_10_Fertig.docx*, das mit dem VSTO-Add-In erstellt wurde und mit der Beispieldatenbank *Nordwind.mdb* verknüpft ist.

Ein Custom Task Pane im COM-Add-In definieren

Das VSTO-Werkzeug vereinfacht den Einsatz des in Office 2007 eingeführten frei definierbaren Aufgabenbereichs (Custom Task Pane).

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 zufriedenstellend.

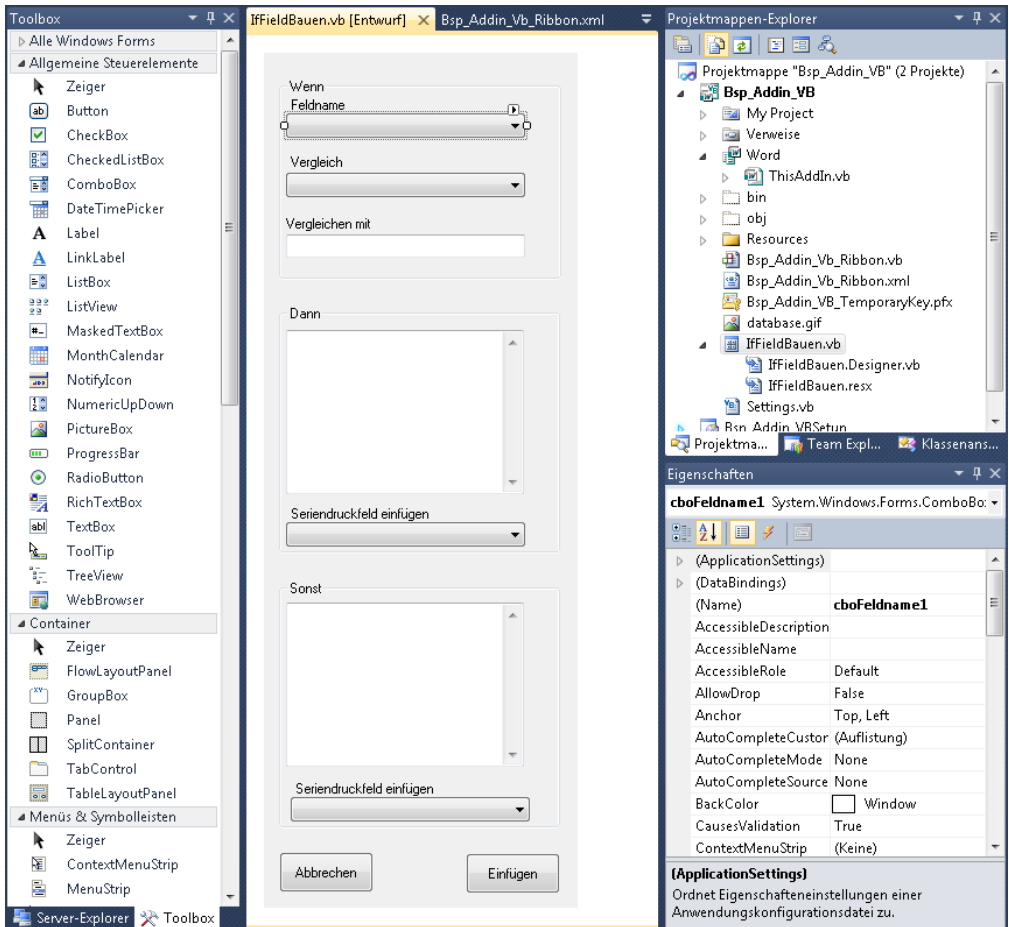
Optimaler ist es, ein WinForms-Benutzersteuerelement im Projekt zu erstellen und dieses in den Aufgabenbereich einzubinden. Damit entfällt die Definition mehrerer einzelner Steuerelemente bei Laufzeit – sie können im Entwurfsmodus erstellt und formatiert werden. Ein Benutzersteuerelement fügen Sie dem 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 Eigenschaftenfenster steht wie üblich bei der Arbeit mit WinForms zur Verfügung.

Im vorliegenden Beispiel wird das vom Seriendruck eingesetzte Dialogfeld zum Einfügen einer *If*-Feldfunktion durch einen Aufgabenbereich ersetzt. Im Gegensatz zur Word-eigenen 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 10.4 zeigt. Die erweiterte Funktionalität setzt die Feldfunktion aus den verschiedenen Steuerelementinhalten zusammen und fügt das Ergebnis mittels der in Kapitel 7 vorgestellten Methode *FeldCodeEinfuegen* in das Dokument ein.

Bei der Auswahl des Menüpunkts *Wenn...dann...sonst* wird die Prozedur *AddinDialogMailMergeInsertIf* aus Listing 10.17 angestoßen. 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* (im Abschnitt »Der Aufgabenbereich *Dokumentaktionen*« ab Seite 557 vorgestellt) können mehrere frei definierbare Aufgabenbereiche erstellt werden. Deshalb gibt es für dieses Objekt eine *Add*-Methode.

Die Listen im Aufgabenbereich sind noch leer, weshalb die Prozedur *IfFieldInitialisieren* ausgeführt wird. Die weiteren mit Steuerelementen im Aufgabenbereich verbundenen Prozeduren sind ebenfalls in Listing 10.17 enthalten.

Abbildg. 10.12 Das benutzerdefinierte Steuerelement für das Custom Task Pane des Beispiel-Add-Ins

Listing 10.17 Code für das Initialisieren eines Custom Task Pane sowie für das benutzerdefinierte Steuerelement

```
'Code, der von der Schaltfläche im Menü "Regeln" ausgeführt wird.
'Initialisiert ein Custom Task Pane mit
'einer Instanz des Benutzersteuerelements ucIfField.
'Der Aufgabenbereich wird bei der Initialisierung automatisch eingeblendet.
Friend 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
```

Listing 10.17 Code für das Initialisieren eines Custom Task Pane sowie für das benutzerdefinierte Steuerelement (*Fortsetzung*)

```

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.cboFeldname1.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 MergeField-Feldfunktion in die passende Textbox,
'an die Stelle der gegenwärtigen 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
            'Empty case
    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
    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

```

Listing 10.17 Code für das Initialisieren eines Custom Task Pane sowie für das benutzerdefinierte Steuerelement (*Fortsetzung*)

```

    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

```

VSTO-Befehl einer Tastenkombination zuweisen

Eine oft gestellte Frage ist, wie einer Tastenkombination in Word ein Befehl in einem COM-Add-In zugewiesen werden kann. Bekanntlich können Tastenkombinationen in Word nur mit Befehlen verbunden werden, die innerhalb der Word-Anwendung laufen (siehe auch Kapitel 17).

Auch VSTO kann diese Regel nicht umgehen. Das Werkzeug macht es aber relativ einfach, das Add-In in einem VBA-Projekt so einzubinden, dass eine VBA-Prozedur, die mit einer Tastenkombination verknüpft ist, den Befehl im Add-In anstößt.

Das Listing 10.18 veranschaulicht die Vorgehensweise. Eine neue Klasse wird dem Add-In-Projekt zugefügt. Die benötigten Imports-Anweisungen (using-Anweisungen in C#) müssen eingegeben werden.

Um eine Prozedur in einer DLL anzusprechen, sucht COM zuerst eine Schnittstelle, die jede Prozedur definiert, die zugänglich gemacht wird. Das vorgestellte Beispiel enthält nur eine solche Prozedur: eine, welche die gleiche Handlung für die Schaltfläche im Menüband für das Ein- und Ausblenden des Aufgabenbereichs auslöst.

Diese Schnittstelle, sowie die Klasse, müssen für COM sichtbar gemacht werden. Die Klasse sowie die Prozedur müssen zudem die Schnittstelle implementieren. Weiter muss der Hauptklasse des Add-Ins – *ThisAddIn* – eine Prozedur zugefügt werden, die die neue Klasse instanziert und als COM-Add-In »veröffentlicht«.

Nun steht Word-VBA die Prozedur *WennDannSonstEinAusBlenden* zur Verfügung – aber VBA muss zuerst das Add-In ansprechen und einbinden. Dieser Code befindet sich zuunterst im Listing 10.18. Um ihn für den Eigengebrauch zu benutzen, müssen Sie den Namen des Add-Ins anpassen, sowie den Namen der zu rufenden Prozedur. Der Name der VBA-Prozedur darf, muss aber nicht geändert werden.

Da das Projekt ein Add-In ist, sollen diese Prozedur und die damit verbundene Tastenkombination allgemein zur Verfügung stehen. Deshalb steht der Code in einer Dokumentvorlage, die Word automatisch als Dokument-Add-In zur Verfügung stellt, weil sie im *Startup*-Ordner von Word gespeichert ist.

Der letzte Schritt ist, die VBA-Prozedur mit einer beliebigen Tastenkombination zu verbinden. Diese Schritte finden Sie im Kapitel 17 beschrieben. Alles speichern, dann die Tastenkombination drücken, um die Zuweisung zu testen.

PROFITIPP

Statt die Dokumentvorlage in den *Startup*-Ordner des Benutzers zu installieren, kann das Add-In beim Laden, in der *ThisAddin_Startup*-Prozedur, die Vorlage explizit laden, und in *ThisAddin_Shutdown* sie wieder entfernen. Siehe den Abschnitt »Makros von außen anstoßen« für die hierfür benötigte Syntax.

Listing 10.18 Nötige Strukturen, um Code im VSTO-Add-In für VBA zugänglich zu machen

```
'Code einer Klasse im VSTO-Add-In
Imports System.Runtime.InteropServices

<ComVisible(True)>
Public Interface ICodeVerbundenMitWordTastenkombinationen
    Sub WennDannSonstEinAusBlenden()
End Interface

<ComVisible(True)>
<ClassInterface(ClassInterfaceType.None)>
Public Class CodeVerbundenMitWordTastenkombinationen
    Implements ICodeVerbundenMitWordTastenkombinationen

    Public Sub WennDannSonstEinAusBlenden() Implements
        ICodeVerbundenMitWordTastenkombinationen.WennDannSonstEinAusBlenden

        Globals.ThisAddIn.ribbon.AddinDialogMailMergeInsertIf( _
            Globals.ThisAddIn.Application.ActiveDocument)
    End Sub

End Class

Partial Public Class ThisAddIn

    Private vbaCodeLink As CodeVerbundenMitWordTastenkombinationen

    Protected Overrides Function RequestComAddInAutomationService() As Object
        If vbaCodeLink Is Nothing Then
            vbaCodeLink = New CodeVerbundenMitWordTastenkombinationen()
        End If
        Return vbaCodeLink
    End Function

End Class

'Code im ThisDocument-Modul einer Word *.dotm Vorlage
'die sich im Startup-Ordner für die Word-Anwendung befindet.
Public Sub CallVSTOMethod()
    Dim addIn As COMAddIn
    Dim automationObject As Object
```

Listing 10.18 Nötige Strukturen, um Code im VSTO-Add-In für VBA zugänglich zu machen (Fortsetzung)

```
Set addIn = Application.COMAddIns("Bsp_Addin_VB")
Set automationObject = addIn.Object
automationObject.WennDannSonstEinAusBlenden
End Sub
```

CD-ROM Das Beispielprojekt befindet sich im Unterordner *Bsp_Addin_VB* der Datei *Bsp_VSTO_VB2010.zip* bzw. *Bsp_VSTO_VB2008.zip* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap10\VB_NET*. Im gleichen Unterordner finden Sie die Dokumentvorlage-Add-In *Bsp_Addin.dotm* mit der zugewiesenen Tastenkombination.

VSTO-Elemente Dokumenten dynamisch zufügen

In VSTO 2008 wurden die Add-In-Werkzeuge um die Funktionalität erweitert, jedes beliebige Word-Dokument zur Laufzeit dynamisch mit VSTO-Elementen zu ergänzen. Somit können beispielsweise mit einer Datenquelle verbundene Inhaltssteuerelemente oder Windows Forms-Steuerelemente auf dem Dokument platziert oder dokumentspezifische Smarttags definiert werden. Bislang standen diese nur Dokumenten zur Verfügung, die im VSTO-Entwurfsmodus erstellt wurden.

HINWEIS Smarttags sind ab Office 2010 mißbilligt. Statt auf der Dokumentoberfläche ein Symbol einzublenden, werden die Einträge im Kontextmenü als *Aktionen* geführt. Sie funktionieren jedoch nicht mehr 100%-ig.

Im Gegensatz zu herkömmlichen Dokumentlösungen werden die VSTO-Elemente nicht im Dokument gespeichert – sie müssen beim Laden des Dokuments dynamisch wieder hergestellt werden.

Mehr über Dokumentlösungen lesen Sie im folgenden Abschnitt. Informationen zum Einsatz von VSTO-Elementen in Add-Ins finden Sie auf <http://msdn.microsoft.com/en-us/library/cc442981.aspx>.

Das Beispiel ausprobieren

Falls Sie Visual Studio 2008 oder 2010 installiert haben, öffnen Sie das passende Beispielprojekt in Visual Studio. Falls Sie Visual Studio 2010 auf einem Rechner mit Office 2007 installiert haben, öffnen Sie das Projekt für Visual Studio 2008, um es von Visual Studio 2010 konvertieren zu lassen.

Durch Drücken der Taste **[F5]** wird das Add-In im Debugmodus 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. Schließen Sie die Word-Anwendung, laden Sie das Projekt in Visual Studio und wählen im Menü *Erstellen* den Menüpunkt *Projektmappe bereinigen*. Dieser Vorgang entfernt den Registry-Eintrag. Anschließend können die Ordner samt Inhalt gelöscht werden.

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.

TIPP In Word 2010 kann dieses Dialogfeld direkt über die Schaltfläche COM-Add-Ins in der Registerkarte *Entwickler-Tools* aufgerufen werden.

Falls Sie keine professionelle Version von Visual Studio 2008 oder 2010 haben, können Sie die Lösung aus dem Ordner *Bsp_Addin_VB2008* der ZIP-Datei *VSTO_Setup_Beispiele.zip* auf Ihren Rechner installieren, um die Wirkung zu sehen. Die darin enthaltene *Setup.exe* führt eine ClickOnce-Installation auf dem Rechner aus. Sie können diese über die *Systemsteuerung/Programme und Funktionen* wieder entfernen.

VSTO-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 – Dokumente entwickelt und bereitgestellt.

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 in .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 bereitliegt.
- 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 Lösungsdokuments sowie die Komplexität der Verteilung und Installation einer Lösung zu nennen.

Die »Tools« bieten dem Office-Entwickler viele nützliche Werkzeuge:

- **Dokument-Entwurfsmodus** Das mit der VSTO-Lösung verbundene Dokument wird in Visual Studio angezeigt und kann dort bearbeitet werden (siehe Abbildung 10.15)
- **Windows Forms- sowie WPF-Steuerelemente** Werden von VSTO in ein Office-gerechtes ActiveX-Format »gepackt« und bereitgestellt, sodass 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- und WPF-Steuer-elemente enthalten kann.

- **Data Binding** Daten mit einer externen Datenquelle dynamisch verbinden und im Dokument (über Textmarken- oder Inhaltssteuerelemente) 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
- **Grafisches Werkzeug für die Erstellung von Menüband-Erweiterungen** Vergleichsweise einfache Anpassungen des Menübands können wie Windows Forms erstellt werden, statt mühsamen Tippens des XML und händisch erstellten Callback-Prozeduren, wie im Abschnitt »Das Add-In mit einer Menüband-Erweiterung ergänzen« ab Seite 537 beschrieben.

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

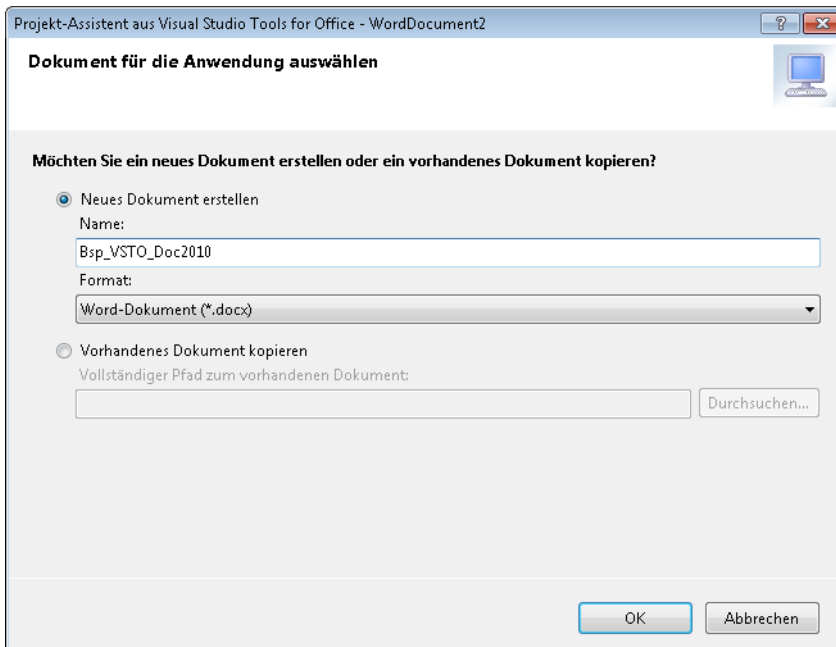
Visual Studio Tools for Office ist im Lieferumfang von ausgewählten Visual Studio Produkten. Für Visual Studio 2008 sowie 2010 ist es Teil der »Professional« und höheren Ausgaben. Alle Ausgaben von Word 2007 sowie Word 2010 enthalten die notwendigen Teile.

Eine VSTO-Lösung anlegen

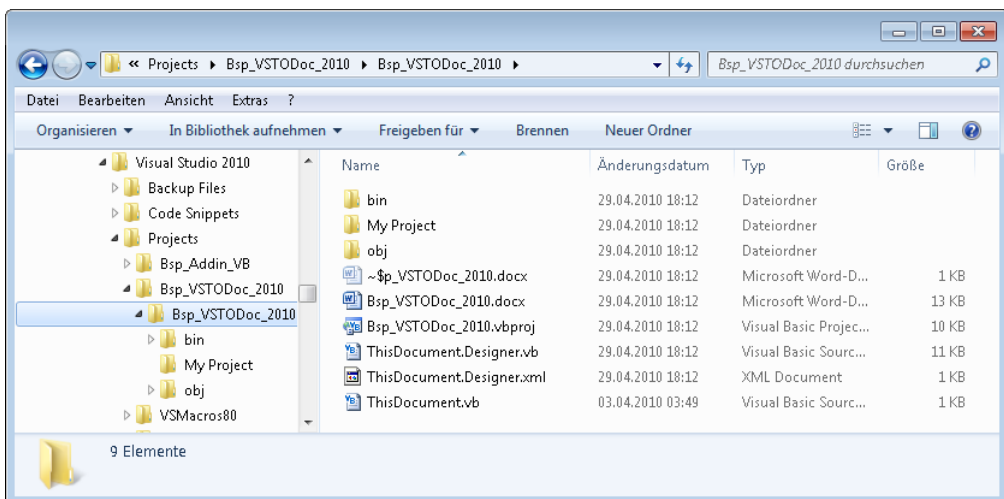
Nach erfolgreicher Installation von Visual Studio Tools for Office erscheinen, wie in Abbildung 10.5 ersichtlich, unter den Visual Basic- und C#-Ordern des Visual Studio-Dialogfelds *Neues Projekt* neue Ordner für *Office*.

Nach Auswahl eines Word-Eintrags (*Word-Dokument* bzw. *Word-Vorlage*) wird das Dialogfeld aus Abbildung 10.13 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 10.14). Hier kann auch gewählt werden, ob das Dokument im binären Word 97-2003-Dateiformat oder im XML-Dateiformat von Word 2007-2010 erstellt werden soll.

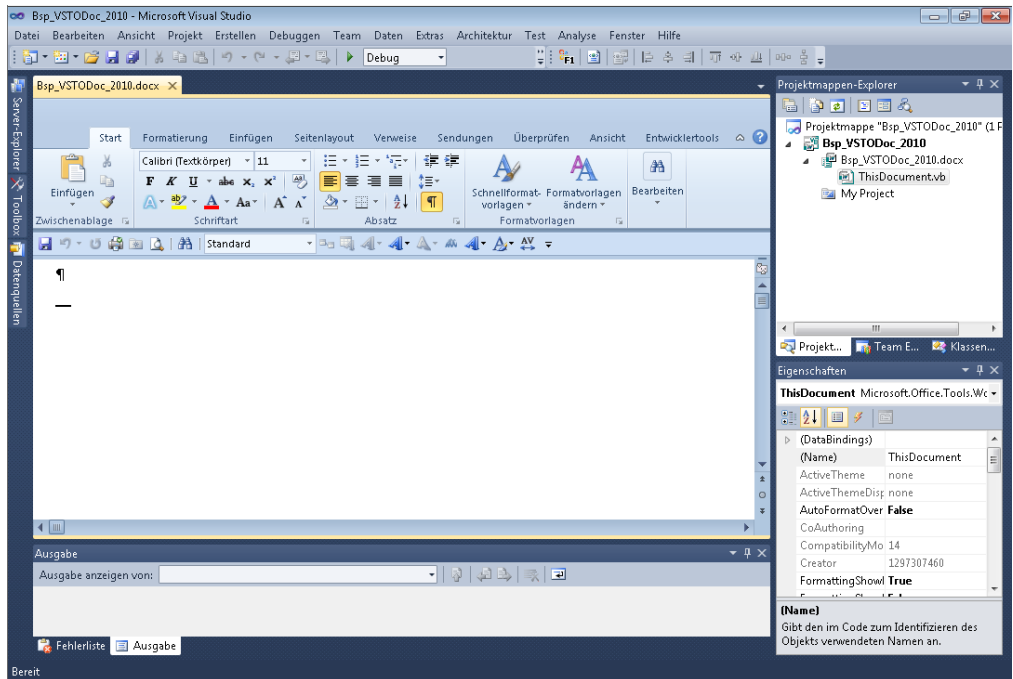
Abbildg. 10.13 Festlegen, ob das Projekt auf einem neuen oder vorhandenen Dokument basieren soll



Abbildg. 10.14 Von VSTO angelegte Ordnerstruktur und Projektdateien, inklusive des Word-Dokuments



Nach einigen Sekunden erscheint im Visual Studio-Anwendungsfenster der VSTO Designer (der Entwurfsmodus) samt Word-Dokument, dargestellt in Abbildung 10.15.

Abbildg. 10.15 Das Word-Dokument kann in Visual Studio bearbeitet werden.


Hinter den Kulissen hat VSTO verschiedene CustomXMLParts sowie Dokumenteigenschaften ins Dokument eingefügt, die wichtige Informationen für die Verwaltung der Anwendung enthalten.

Der Code »hinter dem Dokument«

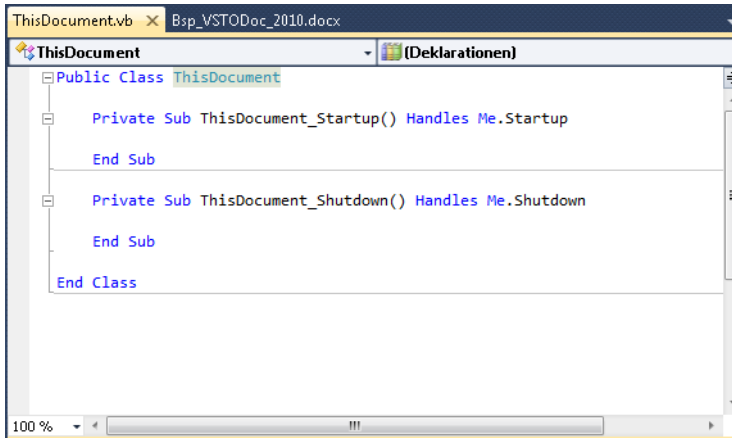
Statt VBA-Code im Dokument zu speichern, wird in einer VSTO-Lösung der Code getrennt aufbewahrt. Als Verbindung dient die *ThisDocument*-Datei (mit der Dateierendung *.cs* für C# bzw. *.vb* für Visual Basic) mit der Teilklasse *ThisDocument*. Sie enthält nur den Code, der für die Zusammenarbeit mit dem Word-Dokument benötigt wird. Wie in Visual Studio üblich, werden die vom Visual Studio Designer erstellten Supportstrukturen durch weitere (in Abbildung 10.14 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.

VSTO erledigt einige Aufgaben für den Entwickler. Verweise zu den benötigten .NET-DLL und COM-Bibliotheken (PIAs) werden erstellt und die C#-Schnittstelle fügt hilfreiche using-Befehlszeilen ein (der VB-Entwickler muss gewünschte Imports-Zeilen selbst eingeben). Außerdem definiert VSTO die zwei Prozeduren *ThisDocument_Startup* sowie *ThisDocument_Shutdown* (Abbildung 10.16). Diese werden beim Starten bzw. beim Schließen des Dokuments ausgeführt: Hier ist der Ansatzpunkt für jede VSTO-Lösung.

TIPP

Die Codeansicht kann über den entsprechenden Befehl im Kontextmenü des *ThisDocument*-Eintrags im Projektmappen-Explorer eingeblendet werden.

Abbildg. 10.16 Der von VSTO generierte Code »hinter dem Dokument« (VB.NET-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. Etwaige WinForms- oder VSTO-Steuerelemente 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-Inhaltssteuerelementen 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 Gruppe im Menüband ein- und ausgeblendet.

Die in den Inhaltssteuerelementen enthaltenen Daten werden in den Daten-Cache (Dokumentzwischenpeicher) geschrieben, wo sie für die VSTO-Technologie auch bei geschlossenem Dokument lesbar sind. Dazu dient das *ServerDocument*-Objekt der Visual Studio Tools for Applications (das auch weitere Anwendungen wie InfoPath unterstützt).

Ferner zeigt das Beispiel, wie der Benutzer eine VSTO-Dokumenten Anpassung entfernen kann, falls das Dokument an jemanden weitergeleitet wird, der keinen Zugang zur VSTO-Lösung hat.

VSTO-Steuerelemente

VSTO umgibt einige Objekte des Word-Objektmodells mit erweiterter Funktionalität und stellt sie als VSTO-Steuerelemente zur Verfügung. Gleichzeitig werden ihnen eine Schnittstelle verpasst, die es dem .NET-Entwickler ermöglicht, sie eher wie .NET- statt COM-Objekte anzusprechen. In Word 2007 und Word 2010 haben wir es primär mit Textmarken und Inhaltssteuerelementen zu tun. Dieses Beispiel veranschaulicht die Arbeit mit Inhaltssteuerelementen.

HINWEIS

Falls lieber mit Textmarken gearbeitet wird, befindet sich das Beispiel der zweiten Auflage dieses Buchs, welches auf Textmarken basiert, auf der CD-ROM zum Buch. Die entsprechende Datei befindet sich im Ordner `\Beilagen\Kap10_VSTO-Dokumentlösung`.

Wie im Kapitel 7 beschrieben, wurden Inhaltssteuerelemente in Word 2007 eingeführt. Sie dienen hauptsächlich der strukturierten Texteingabe und -verwaltung in einem Dokument. VSTO erweitert die von Word zur Verfügung gestellte Funktionalität um .NET-DataBinding- sowie VSTO-Data-Cache-Fähigkeiten.

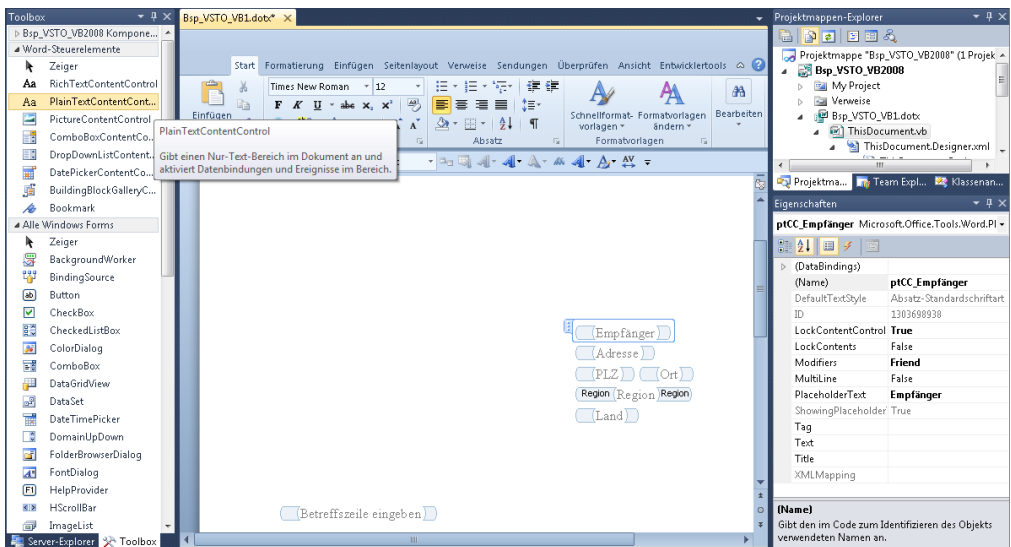
Ein Inhaltssteuerelement kann direkt mit einer .NET-Datenquelle verbunden werden, um den Inhalt einer Datenbank anzuzeigen, sowie Benutzereingaben in die Datenquelle zurückzuschreiben. Zudem kann der Inhalt in den VSTO-Datenspeicher geschrieben werden, was dieses Beispiel veranschaulicht.

Im VSTO-Entwurfsmodus werden Inhaltssteuerelemente über die Visual Studio-Toolbox (Abbildung 10.17) eingefügt. Die Werkzeuge im Word-Menüband stehen *nicht* zur Verfügung. Den Namen des Inhaltssteuerlements sowie weitere Eigenschaften wie Platzhaltertext und Schutzeinstellung werden im Eigenschaftenfenster vorgenommen.

ACHTUNG

Wenn Sie im VSTO-Entwurfsmodus auf ein VSTO-Steuerelement doppelklicken, wird eine Ereignisprozedur in der Codeansicht erstellt.

Abbildg. 10.17 Ein Inhaltssteuerelement über die Visual Studio-Toolbox einfügen und Eigenschaften festlegen



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-Menüband 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. Die Ereignisse für Inhaltssteuerelemente wurden im Kapitel 7 beschrieben.

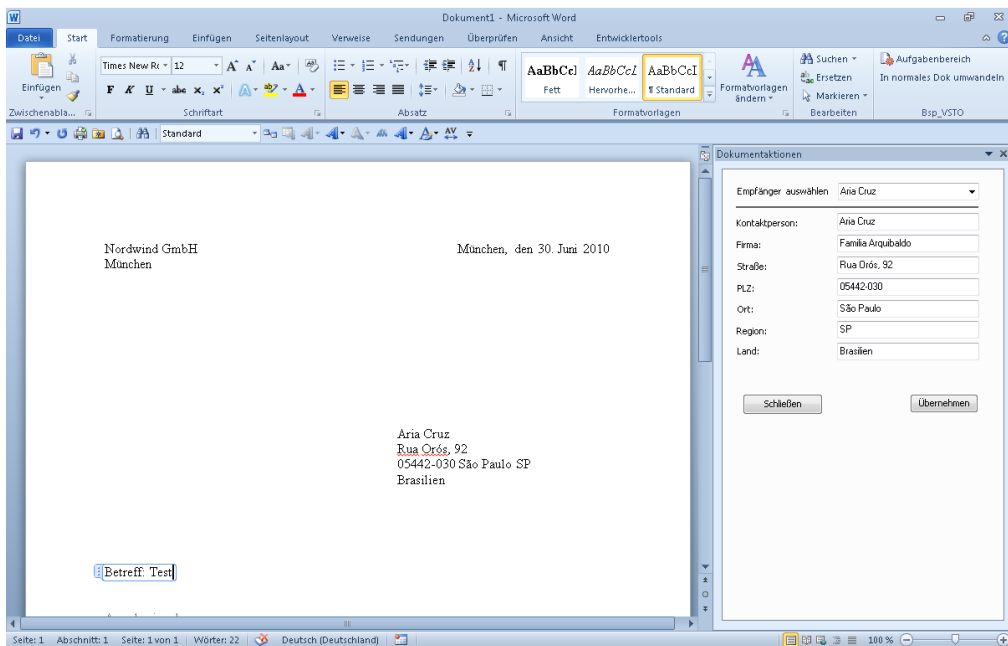
Egal, welche Schnittstellen der Entwickler wählt, die Grundlagen dafür werden in der Prozedur *ThisDocument_Startup* gelegt. Und wo erforderlich, werden sie in der Prozedur *ThisDocument_Shutdown* außer Kraft gesetzt.

Die Beispiellösung zeigt auf, wie eine Gruppe mit zwei Schaltflächen der Word-eigenen Registerkarte *Start* zugefügt sowie ein *Dokumentaktionen*-Aufgabenbereich angelegt werden, wie in Abbildung 10.18 zu sehen.

Die erste Schaltfläche der Gruppe *Bsp_VSTO* schaltet den Aufgabenbereich ein und aus. Die zweite entfernt die VSTO-Lösung aus dem Dokument, sodass Word-Benutzer ohne Zugang zur Lösung störungsfrei damit arbeiten können.

Der Aufgabenbereich ermöglicht die Auswahl einer Kundenadresse aus der Firmendatenbank und deren nachträgliche Bearbeitung, bevor die Informationen in die Inhaltssteuerelemente des Dokuments eingefügt werden.

Abbildg. 10.18 VSTO-Dokumentlösung mit Menüband und Aufgabenbereich *Dokumentaktionen*



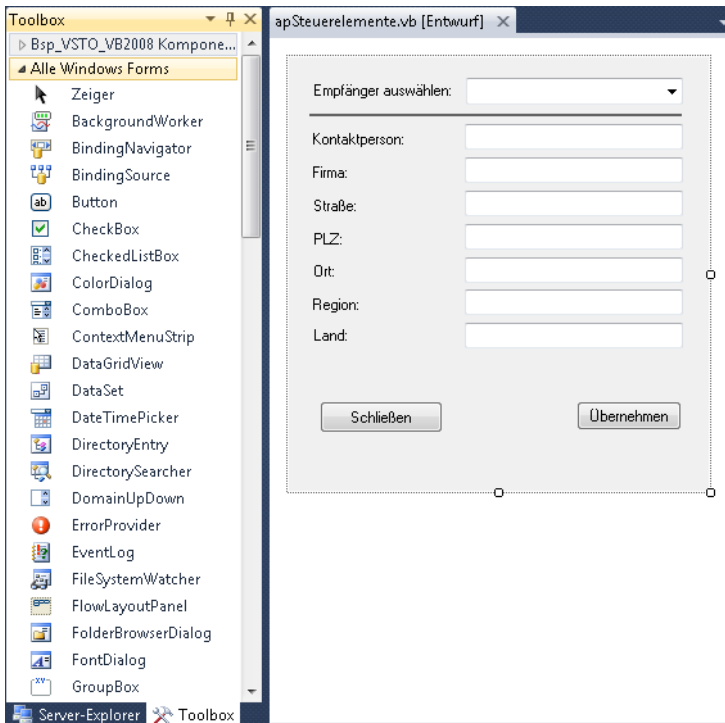
Der Aufgabenbereich *Dokumentaktionen*

Der von VSTO zur Verfügung gestellte Aufgabenbereich *Dokumentaktionen* ist eigentlich Teil der in Word 2003 eingeführten Smart Document-Funktionalität. Die Nachfrage seit Erscheinen von Office XP nach einem frei definierbaren Aufgabenbereich war 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.

Die Vorgehensweise zur Erstellung eines dokumentspezifischen Aufgabenbereichs ist der im Abschnitt »Ein Custom Task Pane im COM-Add-In definieren« ab Seite 545 beschriebenen sehr ähnlich. Es ist auch in diesem Fall vorteilhaft, den Inhalt in einem Benutzersteuerelement zu definieren (Abbildung 10.19).

Abbildg. 10.19 Das Benutzersteuerelement für den VSTO-Aufgabenbereich



Der Code, der den Aufgabenbereich initialisiert und einblendet, ist in Listing 10.19 ersichtlich. Da diese Handlungen ebenfalls Speicheraufforderungen für die *Normal.dotm* 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.

ACHTUNG

Da nur ein dokumentspezifischer Aufgabenbereich zur Verfügung steht, im Gegensatz zu Custom Task Panes in COM-Add-Ins, unterscheidet sich der Code in dieser Hinsicht gegenüber dem in Listing 10.17. Also aufgepasst und immer die passenden Methoden für die Art VSTO-Lösung und Aufgabenbereich benutzen!

Bei der Einblendung des Aufgabenbereichs müssen drei Objekte berücksichtigt werden: das im Word-Aufgabenbereich eingebettete VSTO-Steuerelement (ein ActiveX-Steuerelement, das die Ver-

bindung zwischen Word und VSTO unterhält), 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 (`ActionsPane`) zur Verfügung steht, existiert dieser schon, wir müssen ihn lediglich ansprechen: `Me.ap = Me.ActionsPane`. Beim Benutzersteuerelement (`apSteuerelemente`) 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`.

Am Schluss 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 10.19 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 AufgabenBereichEin
    '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
    End Try
End Sub
```

Listing 10.19 Die Betätigung der entsprechenden Symbolschaltfläche führt diese Prozedur aus (Fortsetzung)

```

'Sicherstellen, dass der Aufgabenbereich eingeblendet ist.
Me.tp.Visible = True
'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
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.ap.Controls = 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

```

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 für unser Beispiel mit der VSTO-Dokumentvorlage 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")

```

HINWEIS Wir gehen hier auf die Arbeit mit Datenquellen in .NET nicht näher ein. Diese werden in Büchern über .NET Windows Forms oder zum Thema ADO .NET eingehend vorgestellt.

Die Prozedur *AufgabenbereichDatenEinbinden* in Listing 10.20 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 10.20 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. Die andere schreibt den Inhalt der WindowsForms-Textboxen in die Inhaltssteuerelemente des Dokuments. Listing 10.21 veranschaulicht den Vorgang.

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 10.21 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
        .ptCC_Empfänger.Text = Me.KontaktpersonTextBox.Text
        .ptCC_Adresse.Text = Me.StraßeTextBox.Text
        .ptCC_PLZ.Text = Me.PLZTextBox.Text
        .ptCC_Ort.Text = Me.OrtTextBox.Text
        .ptCC_Land.Text = Me.LandTextBox.Text

        'Falls Informationen für "Region" im Aufgabenbereich vorhanden sind...
        Dim strRegion As String = Me.RegionTextBox.Text
        If strRegion.Length > 0 Then

```

Listing 10.21 Im Aufgabenbereich enthaltene Daten in die Dokument-Textmarken schreiben (Fortsetzung)

```
'...sie in das Inhaltssteuerelement schreiben.
.ptCC_Region.Range.Font.Hidden = False
.ptCC_Region.Range.Text = strRegion
Else
    'Steht keine Information bereit,
    'das Inhaltssteuerelement unsichtbar machen.
    .ptCC_Region.Range.Font.Hidden = True
End If
End With
End Sub
```

Aber was passiert, wenn der Benutzer den Aufgabenbereich geschlossen hat und wieder benutzen möchte? Der Word-eigene Befehl *Actions Pane* steht ihm unter *Ansicht/Hide Show* zur Verfügung. Das ist aber den meisten Benutzern eher unbekannt und zudem etwas weit weg von den Befehlen, die meistens für die Herstellung eines Briefes gebraucht werden. Es wäre also sinnvoll, die Schaltfläche in der Registerkarte *Start* zur Verfügung zu stellen, was uns zum nächsten Thema bringt.

Das Menüband im VSTO Menüband-Entwurfsmodus anpassen

Im Abschnitt »VSTO-Add-In-Benutzerschnittstellen« ab Seite 537 wurde beschrieben, wie ein Menüband über Einbindung eines Menüband-XML-Teils angepasst wird. VSTO 2008 ergänzte die Menüband-Unterstützung um einen neuen Entwurfsmodus, der die Erstellung einfacher Menübänder (Multifunktionsleisten) erheblich erleichtert.

Um dem Projekt ein Menüband im Entwurfsmodus zuzufügen, klicken Sie rechts auf den Projektnamen. Aus dem Kontextmenü *Hinzufügen*, dann *Neues Element* wählen (wie Abbildung 10.7 veranschaulicht). Im Dialogfeld wird der Eintrag *Menüband (Visueller Designer)* statt *Menüband (XML)*, wie in Abbildung 10.8 zu sehen ist, gewählt.

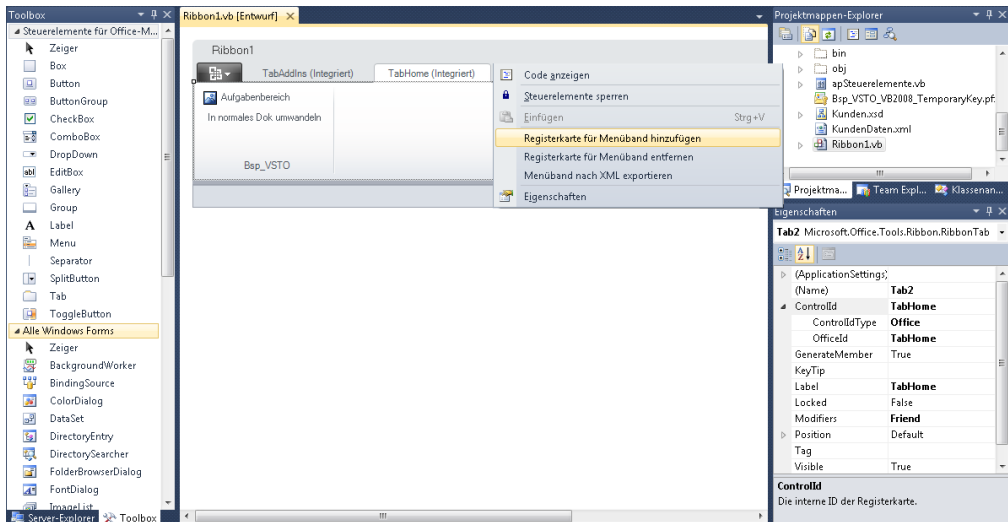
VSTO fügt eine neue Klasse – *Ribbon1* – in das Projekt ein und zeigt ein Menüband im Entwurfsmodus (Abbildung 10.20). Die Word-eigene Registerkarte Add-Ins mit einer Gruppe mit dem allgemeinen Namen *Gruppe1* ist angewählt. Es ist möglich, weitere, eigene Registerkarten zu definieren, oder die integrierte mit weiteren Anwendungen zu teilen.

Die Arbeit im Entwurfsmodus ist der Erstellung eines Window-Forms ähnlich. Statt XML-Tags zu schreiben, um das Menüband zu definieren werden Steuerelemente mit der Maus aus der Toolbox in die Zielgruppe einer Registerkarte gezogen. Einstellungen werden im Eigenschaftenfenster vorgenommen, statt Attribute in XML-Tags zu definieren.

Mit einem Doppelklick auf ein Steuerelement wird ein Codefenster geöffnet mit Fokus in der passenden »Click«-Prozedur. Alle Elemente der *Ribbon1*-Klasse stehen als vollwertige Objekte mit Eigenschaften und Methoden zur Verfügung – die mühsame Erstellung von »get«-Callbacks, um den Zustand der Steuerelemente zu ermitteln bzw. festzulegen, entfällt.

Im vorliegenden Beispiel wird die *Start*-Registerkarte um eine eigene Gruppe ergänzt. Um das zu erreichen, wird im Entwurfsmodus dem Menüband eine neue Registerkarte zugefügt. Standardmäßig nimmt VSTO an, es handelt sich um eine nicht integrierte Registerkarte. Wir können die Einstellungen unter *ControlID* im Eigenschaftenfenster anpassen, um auf eine integrierte Registerkarte zu zeigen.

Abbildg. 10.20 Entwurfsmodus in VSTO 2010 für das Menüband von Word 2010



Wählen Sie aus der Liste *ControlIDType* den Eintrag *Office*. Für die Eigenschaft *OfficeID* geben Sie den ControlID-Wert der gewünschten Registerkarte ein. Die Zeichenkette »TabHome« entspricht dem der Registerkarte *Start*. Die Beschriftung im Menüband-Designer wird automatisch angepasst.

HINWEIS

Die ControlID-Werte für alle Office-Anwendungen sind in Excel-Mappen zusammengefasst und können heruntergeladen werden <http://www.microsoft.com/downloads/details.aspx?familyid=4329D9E9-4D11-46A5-898D-23E4F331E9AE&displaylang=en> für Office 2007 bzw. <http://www.microsoft.com/downloads/details.aspx?FamilyID=3f2fe784-610e-4bf1-8143-41e481993ac6&displaylang=en> für Office 2010. Sie finden die entsprechenden .exe-Dateien auch auf der CD-ROM zum Buch im Ordner *\Beilagen\Ribbon*.

Blenden Sie die Visual Studio-Toolbox ein und wählen Sie den Abschnitt *Office Ribbon Controls*. Per Doppelklick oder durch Ziehen mit der Maus können die gewünschten Steuerelemente auf das Menüband platziert werden.

TIPP

Einen Klick auf die Schaltfläche *Automatisch im Hintergrund* dockt die Toolbox neben dem Menüband, sodass mehrere Steuerelemente auf die Entwurfsfläche gezogen werden können.

Als Erstes benötigen wir ein *Gruppe*-Steuerelement. Nach dessen Hinzufügen werden die Eigenschaften *Name* und *Label* im Eigenschaftenfenster festgelegt.

Die erste Schaltfläche soll die Handlung der Word-eigenen *ViewDocumentActionsPane* aus *Ansicht/Show Hide* widerspiegeln. Es handelt sich um ein Steuerelement des Typs *Toggle*, weshalb der Toolbox-Eintrag *ToggleButton* der Gruppe hinzuzufügen ist.

Da der Entwurfsmodus keine Definierung eines Word-eigenen Steuerelements zulässt, können wir lediglich das Ikone automatisch übernehmen. Dies wird über die Eigenschaft *OfficeImageID* bewerkstelligt, wo wir den idMSO-Wert (*ViewDocumentActionsPane*) eingeben. Die eigentliche Handlung muss im Steuerelement-Ereignis definiert werden.

Doppelklicken Sie auf die Schaltfläche: VSTO öffnet die Codedatei für das Menüband und erstellt die Prozedur für die Schaltfläche (Listing 10.22). Sie brauchen nur eine einzige Codezeile, die den internen Befehl ausführt.

Listing 10.22 Prozedur für eine Schaltfläche im Menüband definiert im Entwurfsmodus

```
Private Sub tglDokumentAktionen_Click(ByVal sender As System.Object, _
    ByVal e As Microsoft.Office.Tools.Ribbon.RibbonControlEventArgs) _
    Handles tglDokumentAktionen.Click
    Globals.ThisDocument.Application.CommandBars.ExecuteMso("ViewDocumentActionsPane")
End Sub
```

HINWEIS Vergleichen Sie die Struktur dieser Prozedur mit jener eines Callbacks für Menüband-XML im Abschnitt »Das Add-In mit einer Menüband-Erweiterung ergänzen« ab Seite 537.

Die zweite Schaltfläche ist ein gewöhnliches *Button*-Steuerelement. Auch hier wird darauf doppelt geklickt, um die auszuführende Prozedur zu erstellen (Listing 10.25). Diese wird im Abschnitt »VSTO-Lösung von einem Dokument abtrennen« ab Seite 568 eingehender vorgestellt.

Die Menüband-Anpassung für das Beispiel ist nun angeschlossen. Drücken Sie **F5**, um das Resultat in der Word-Anwendung zu betrachten.

PROFITIPP

Da der Entwurfsmodus nicht alle Möglichkeiten der Menüband-Anpassung unterstützt, sondern nur die »meist gebrauchten«, soll ein Projekt von vorn herein eingehend analysiert werden. Falls die Benutzerschnittstelle Erweiterungen verlangt, die der Entwurfsmodus nicht unterstützt, soll von Anfang an mit Menüband-XML gearbeitet werden. Es ist zwar möglich, eine mit dem Entwurfsmodus erstellte Menüband-Anpassung in Menüband-XML zu exportieren, aber der damit verbundene Code muss von Grund auf neu geschrieben werden.

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 Listing 10.23 veranschaulicht, wie der Datencache 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* steht die Kommandozeile *InitializeDataCache()*, die die entsprechende Methode aufruft (Listing 10.23). Diese 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 Inhaltssteuerelemente in die Variablen des Zwischenspeichers geschrieben werden.

Listing 10.23 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 (CachedEmpfänger Is Nothing) Then
        CachedEmpfänger = "Empfänger nicht bekannt"
    End If
    If (CachedAdresse Is Nothing) Then
        CachedAdresse = "Adresse nicht bekannt"
    End If
    If (CachedPLZ Is Nothing) Then
        CachedPLZ = "PLZ nicht bekannt"
    End If
    If (CachedOrt Is Nothing) Then
        CachedOrt = "Ort nicht bekannt"
    End If
    If (CachedLand Is Nothing) Then
        CachedLand = "Land nicht bekannt"
    End If
    If (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
    CacheSteuerlementenDaten (CachedEmpfänger, Me.ptCC_Empfänger.InnerObject)
    CacheSteuerlementenDaten (CachedAdresse, Me.ptCC_Adresse.InnerObject)
    CacheSteuerlementenDaten (CachedPLZ, Me.ptCC_PLZ.InnerObject)
    CacheSteuerlementenDaten (CachedOrt, Me.ptCC_Ort.InnerObject)
    CacheSteuerlementenDaten (CachedLand, Me.ptCC_Land.InnerObject)
    CacheSteuerlementenDaten (CachedBetreff, Me.rtCC_Betreff.InnerObject)
End Sub

Private Function CacheSteuerlementenDaten(ByRef CacheVariable As Object, _
    ByVal cc As Word.ContentControl) As String

    Dim success As Boolean = False
    If Not cc Is Nothing Then
        Try
            CacheVariable = cc.Range.Text
            success = True
        Catch ex As Exception
            MessageBox.Show(ex.Message)
        End Try
    End If
End Function
```

Listing 10.23 Den Zwischenspeicher initialisieren und Daten hineinschreiben (*Fortsetzung*)

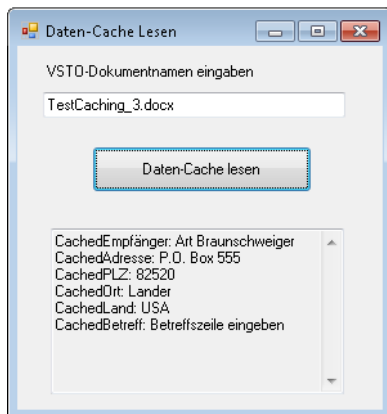
```

    CacheVariable = "Fehler beim Schreiben."
End Try
Else
    CacheVariable = "Inhaltssteuerelement nicht vorhanden."
End If
Return success
End Function

```

Von außerhalb des Dokuments greift die »ServerDocument«-Funktionalität des Namespaces `Microsoft.VisualStudio.Tools.Applications.ServerDocument` auf den Dokumentzwischenspeicher zu. Dieser Namespace ist *nicht* Teil der `Microsoft.Office.Tools`-DLL, die der Entwickler in einer VSTO-Dokumentlösung benutzt. Diese DLL kann auch auf einem Netzlaufwerk oder einem Webserver installiert werden.

Um die Funktionalität zu veranschaulichen, wurde eine kleine WinForms-Anwendung (siehe Abbildung 10.21) erstellt. Den zugehörigen Code finden Sie in Listing 10.24. Der Dateiname eines VSTO-Dokuments in einem im Code festgelegten Ordner (*Eigene Dokumente*) wird in das erste Textfeld eingegeben. Die Betätigung der Schaltfläche *Daten-Cache lesen* schreibt die zwischengespeicherten Informationen in das zweite Textfeld.

Abbildg. 10.21 Die Daten aus dem Zwischenspeicher eines geschlossenen VSTO-Dokuments anzeigen


Falls das Dokument im vorgegebenen Pfad vorhanden ist, wird ein neues `ServerDocument`-Objekt (`serverDok`) angelegt. Sein Zwischenspeicher wird angesprochen (`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 jedes *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 Inhaltssteuerelementen befindet. Das Ergebnis wird der Zeichenkette `cachedInhalt` hinzugefügt, die am Ende der Prozedur `btnDatenLesen_Click` in das zweite Textfeld geschrieben wird.

Listing 10.24 Daten aus dem Zwischenspeicher eines geschlossenen VSTO-Dokuments lesen

```
Imports VSTA = Microsoft.VisualStudio.Tools.Applications
Imports System.Xml.Serialization
Imports System.IO
Imports System.Windows.Forms

Public Class DatenCacheLesen
    Private Sub btnDatenCacheLesen_Click(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles btnDatenCacheLesen.Click
        Dim dokName As String = Me.txtDokumentName.Text
        Dim dokPfad As String
        If dokName.Length > 0 Then
            dokPfad = System.Environment.GetFolderPath( _
                Environment.SpecialFolder.MyDocuments) & "\" & dokName
        Else
            MessageBox.Show("Bitte einen Dateinamen eingeben.")
            Return
        End If

        If File.Exists(dokPfad) Then
            Try
                Dim serverDok As New VSTA.ServerDocument(dokPfad)
                Dim datenAusDemCache As VSTA.CachedData = serverDok.CachedData
                Dim datenHostItems As VSTA.CachedDataHostItemCollection = _
                    datenAusDemCache.HostItems
                Dim datenHostItem As VSTA.CachedDataHostItem
                Dim datenPunkt As VSTA.CachedDataItem
                Dim cachedInhalt As String = ""
                For Each datenHostItem In datenHostItems
                    For Each datenPunkt In datenHostItem.CachedData
                        'Schreibt das rohe XML
                        System.Diagnostics.Debug.Print(vbTab & datenPunkt.Id & ": " & datenPunkt.Xml)
                        cachedInhalt = cachedInhalt & datenPunkt.Id & ": " & _
                            ExtractTextAusXML(datenPunkt.Xml) & vbCrLf
                    Next
                Next
                'Zeigt den Dateninhalt im Fenster an
                Me.txtDaten.Text = cachedInhalt
                serverDok.Close()
            Catch ex As Exception
                MessageBox.Show(ex.Message)
            End Try
        Else
            MessageBox.Show("Das Dokument " & dokPfad & " konnte nicht gefunden werden.")
        End If
    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
        Return inhaltText
    End Function
End Class
```

VSTO-Lösung von einem Dokument abtrennen

Wie in der Einleitung zu diesem Abschnitt »Das VSTO-Dokument vorbereiten« ab Seite 555 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 10.25 ersichtlich.

VSTO schreibt Custom XML-Teile und Dokumenteigenschaften in ein Word 2007 bzw. Word 2010 Dokument, um die Lösung mit dem Dokument zu verbinden. Der »Microsoft Office Solution loader« achtet auf das Vorhandensein dieser Teile und, falls sie vorhanden sind, lädt die entsprechende Lösung.

HINWEIS

Der »Microsoft Office Solution loader« (auch als »Visual Studio Tools for Office loader« bekannt) ist Bestandteil der Office 2003-, Office 2007- und Office 2010-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 der Microsoft-Webseite <http://msdn.microsoft.com/en-us/library/bb608603.aspx>.

Die Methode `RemoveCustomization` entfernt die Custom XML-Teile aus dem Dokument. Es lässt jedoch die Dokumenteigenschaften bestehen für den Fall, dass die Lösung später mit dem Dokument wieder verbunden werden soll. Da unser Beispiel annimmt, dies wird nicht der Fall, entfernt der Code in Listing 10.25 explizit die drei von VSTO angelegten Dokumenteigenschaften `_AssemblyLocation`, `_AssemblyName` sowie `Solution ID`.

Das alles scheint ziemlich einfach zu sein und ist es auch, sofern kein dokumentspezifischer Aufgabenbereich vorhanden ist.

Wird das Beispieldokument 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 *Actions-Pane*. 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 immer noch eine Verbindung zur VSTO-Lösung. Durch Ausblendung der Gruppe im Menüband werden die mit der Lösung verbundenen Befehle dem Benutzer entzogen. Unter Umständen sollen Ereignisprozeduren testen, ob `RemoveCustomization` durchgeführt wurde, und, wenn ja, keine weiteren Handlungen durchführen.

Das Dokument einer VSTO-Vorlagenlösung bleibt weiterhin mit seiner Vorlage verbunden. Falls es mit einer anderen Vorlage, wie *Normal.dotm*, verbunden werden soll, geschieht dies am besten beim Schließen des Dokuments. Dafür wird das Ereignis `ThisApplication_DocumentBeforeClose` benutzt.

Listing 10.25 Die VSTO-Lösung von einem Dokument trennen

```

Private Sub btnVSTOEntfernen_Click(ByVal sender As System.Object,
    ByVal e As Microsoft.Office.Tools.Ribbon.RibbonControlEventArgs) _
    Handles btnVSTOEntfernen.Click

    Dim thisDoc As Tools.Document = Globals.ThisDocument
    Try
        'Entfernt die Custom XML-Teile...
        thisDoc.RemoveCustomization()
        '...und die Dokumenteigenschaften aus dem Dokument.
        Dim prop As Office.DocumentProperty
        For Each prop In Globals.ThisDocument.InnerObject.CustomDocumentProperties
            Select Case prop.Name
                Case "_AssemblyName", "_AssemblyLocation", "Solution ID"
                    prop.Delete()
                Case Else
            End Select
        Next
    Catch ex As Exception
        MessageBox.Show(ex.Message)
    Finally
        Me.grpBsp_VSTO.Visible = 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 bVstoProp As Boolean = False
    Dim prop As Office.DocumentProperty
    For Each prop In Me.InnerObject.CustomDocumentProperties
        Select Case prop.Name
            Case "_AssemblyName", "_AssemblyLocation", "Solution ID"
                bVstoProp = True
            Exit For
        Case Else
        End Select
    Next
    If Not bVstoProp Then Me.InnerObject.AttachedTemplate = _
        ThisApplication.NormalTemplate
End Sub

```

VSTO 2005-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 Menüband in die Lösung einzubinden; dies ist erst mit der nächsten VSTO-Version möglich.

Das Setupprogramm 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 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 (siehe den Hinweis im Abschnitt »VSTO COM-Add-Ins« ab Seite 533). Wir geben nur einen kurzen Überblick.

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 eine etwaige »Helfer *dll*« ein. Ab VSTO 2008 und Office 2007 erfolgt die Verteilung meistens über die ClickOnce-Funktionalität, die die Verantwortung für die Erfüllung dieser Voraussetzung übernimmt. Eine Lösung kann aber auch über eine MSI installiert werden.

Das Beispiel ausprobieren

Falls Sie Visual Studio 2008 oder 2010 Professional (oder höher) installiert haben, öffnen Sie das passende Beispielprojekt in Visual Studio. Falls Sie Visual Studio 2010 auf einem Rechner mit Office 2007 installiert haben, öffnen Sie das Projekt für Visual Studio 2008 und lassen es von Visual Studio 2010 konvertieren.

Durch Drücken der Taste **[F5]** wird die Dokumentlösung im Debugmodus in der Word-Anwendung gestartet, wodurch das Projekt auf Ihrem Rechner als vertrauenswürdig eingestuft wird.

Da es sich um eine Word-Dokumentvorlage handelt, wird ein auf der Vorlage basierendes Dokument erstellt. Sie können die Dateneingabe testen und das Dokument im Ordner *Eigene Dokumente* speichern, um später die Server Document-Funktionalität zu testen.

Falls Sie keine professionelle Version von Visual Studio 2008 oder 2010 haben, können Sie die Lösung aus dem Ordner *VSTO_Dok_VB* der ZIP-Datei *VSTO_Setup_Beispiele.zip* auf Ihren Rechner installieren, um die hier vorgestellte Funktionalität zu sehen. Die darin enthaltene *Setup.exe* führt eine ClickOnce-Installation auf dem Rechner aus.

CD-ROM

Die vorgestellte Lösung befindet sich im Ordner *VSTO_Dok_VB* der Datei *Bsp_VSTO_VB2008.zip* bzw. *Bsp_VSTO_VB2010.zip* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap10\VB_NET*.

HINWEIS

Immer interessant ist das Blog von Jens Häupel: <http://blogs.msdn.com/jensha/archive/tags/.NET+Dev+mit+Office/default.aspx>.

Die englischsprachigen MSDN-Seiten zum Thema VSTO starten auf der Seite [http://msdn.microsoft.com/en-us/library/d2tx7z6d\(VS.90\).aspx](http://msdn.microsoft.com/en-us/library/d2tx7z6d(VS.90).aspx) (VSTO 2008) bzw. <http://msdn.microsoft.com/en-us/library/d2tx7z6d.aspx> (VSTO 2010). Um die deutschen Übersetzungen zu sehen, ersetzen Sie »en-us« durch »de-de«

Es gibt zudem ein sehr nützliches Buch in englischer Sprache:

»Visual Studio Tools for Office 2007« von Eric Carter und Eric Lippert, herausgegeben von Addison-Wesley (ISBN-13: 978-0-321-53321-0). Die Codebeispiele darin sind in C#.

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 514) und wie Word effektiv ferngesteuert werden kann (Seite 518)
- Als weiterer Schwerpunkt wurde dargestellt, wie bereits erstellte Programmsequenzen gemeinsam genutzt werden können (Seite 522) 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 524)
- Zudem wurde veranschaulicht, wie Prozeduren in globalen Add-Ins von anderen Programmierumgebungen aus aufgerufen werden können (Seite 526)
- Es wurden die Grundlagen der Fernsteuerung der Word-Anwendung durch eine Visual Studio .NET-Anwendung erläutert (Seite 527)
- Darauf folgte eine Diskussion zum Thema Visual Studio Tools for Office (VSTO). Ab Seite 533 wurde an zwei Beispielen die Funktionalität der Add-In- sowie dokumentspezifischen Lösungen aufgezeigt.

Kapitel 11

Andere Programme von Word aus steuern

In diesem Kapitel:

Microsoft Excel fernsteuern	574
Microsoft PowerPoint fernsteuern	577
Microsoft Visio fernsteuern	582
Microsoft Outlook fernsteuern	584
Microsoft Access fernsteuern	592
Auf Datenbanken zugreifen	594
Zusammenfassung	604

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 zugrunde liegende 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 Compilerkonstante definiert, über deren Wert die gewünschte Funktionalität gesteuert wird. Das Thema »Early und Late Binding« sowie Compileranweisungen 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.xlsx"
```

Microsoft Excel fernsteuern



In diesem Abschnitt werden die verschiedenen Möglichkeiten zum Fernsteuern von Microsoft Excel erläutert, 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« ab Seite 601 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 sogenannte 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.xlsx"

    #Const EARLYBINDING = False 'oder True
    Dim intAntwort As Integer

    System.Cursor = wdCursorWait

    'Variablen und Objekte anlegen
    #If EARLYBINDING Then
        'Wird mit Early Binding gearbeitet, muss ein Verweis auf
        'die "Microsoft Excel 14.0 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)
                With xlsWks
                    'Arbeitsblatt ausdrucken
                    intAntwort = MsgBox("Soll der Ausdruck erstellt werden?", vbYesNo)
                    If intAntwort = vbYes Then
```

Listing 11.1 Ausdrucken eines Arbeitsblatts mittels Fernsteuerung von Excel *(Fortsetzung)*

```

        .PrintOut
            From:=1, _
            To:=1, _
            Copies:=1
        End If
    End With
    .Saved = True
    .Close
End With
.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, 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 14.0 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 kön-

nen.) 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 14.0 Object Library" aktiviert werden
    Dim xlsApp As Excel.Application
    Dim dblZahl As Double
    Dim dblBasis As Double
    Dim dblResultat As Double

    'Eingabewerte festlegen
    dblZahl = CDb1(InputBox("Anzahl Elemente eingeben", "Kombinationen berechnen", "64"))
    dblBasis = CDb1(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
```

CD-ROM Die oben dargestellten Beispiele finden Sie in der Beispieldatei *Bsp11_01.docm* 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.pptx"

    #Const EARLYBINDING = False 'oder True

    Dim intAntwort As Integer
    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 14.0 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
                'Präsentation ausdrucken
                intAntwort = MsgBox("Soll der Ausdruck erstellt werden?", vbYesNo)
                If intAntwort = vbYes Then
                    bBackground = .PrintOptions.PrintInBackground
                    .PrintOptions.PrintInBackground = False
                    .PrintOut _
                        From:=1, _
```

Listing 11.3 Ausdrucken einer Folie mittels Fernsteuerung von PowerPoint (Fortsetzung)

```

        To:=1,
        Copies:=1
        .PrintOptions.PrintInBackground = bBackground
    End If
    .Saved = True
    .Close
End With
.Quit
End With
Else
    MsgBox "Neue Instanz von PowerPoint konnte nicht erzeugt werden."
End If

Set pptPrs = Nothing
Set pptApp = Nothing
System.Cursor = wdCursorNormal
End Sub

```

Anhand der eingefügten Kommentarzeilen sollte das Listing 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 in älteren Programmversionen 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.

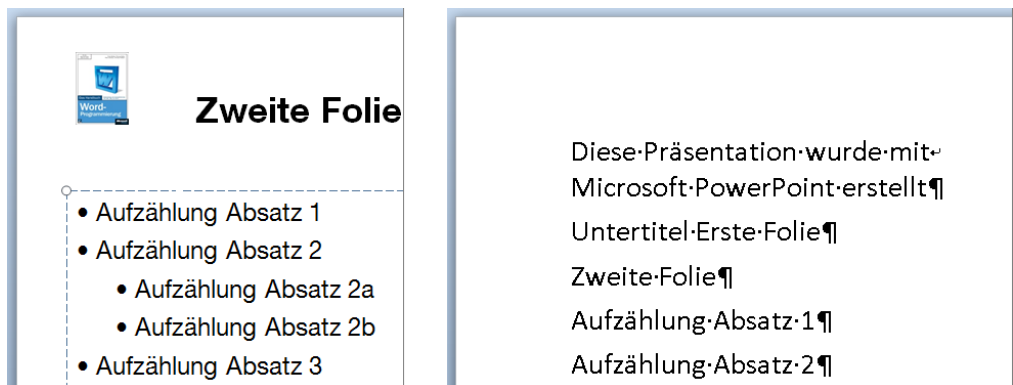
In der Variablen `bBackground` wird der aktuelle Status der Option *Drucken im Hintergrund* gespeichert, damit die veränderte Einstellung nach erfolgtem Ausdruck zurückgesetzt werden kann:

```
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 neben 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.pptx"

    #Const EARLYBINDING = False 'oder True

    'Variablen und Objekte anlegen
    #If EARLYBINDING Then
```


Listing 11.4 Einlesen der Texte aus den Platzhaltern und übertragen in ein Dokument (Fortsetzung)

```

'Wird mit Early Binding gearbeitet, muss ein Verweis auf
'die "Microsoft PowerPoint 14.0 Object Library" aktiviert werden
Dim pptApp As PowerPoint.Application
Dim pptPrs As PowerPoint.Presentation
Dim pptSld As PowerPoint.Slide
Dim pptShp As PowerPoint.Shape
Dim pptPara As PowerPoint.TextRange

'Neue Powerpoint-Instanz erzeugen
Set pptApp = New PowerPoint.Application
#Else
Dim pptApp As Object
Dim pptPrs As Object
Dim pptSld As Object
Dim pptShp As Object
Dim pptPara As Object

'Neue Powerpoint-Instanz erzeugen
Set pptApp = CreateObject("Powerpoint.Application")
#End If

Dim doc As Word.Document

Set doc = Documents.Add

'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

```

Listing 11.4 Einlesen der Texte aus den Platzhaltern und übertragen in ein Dokument (*Fortsetzung*)

```
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, 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
```

CD-ROM Die obigen Beispiele finden Sie in der Beispieldatei *Bsp11_01.docm* 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 sogenannte 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
    Dim intAntwort As Integer

    System.Cursor = wdCursorWait

    'Variablen und Objekte anlegen
    #If EARLYBINDING Then
        'Wird mit Early Binding gearbeitet, muss ein Verweis auf
        'die "Microsoft Visio 14.0 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
                    intAntwort = MsgBox("Soll der Ausdruck erstellt werden?", vbYesNo)
                    If intAntwort = vbYes Then
                        .Print
                    End If
                End With
                .Saved = True
                .Close
            End With
            .Quit
        End With
    Else
        MsgBox "Neue Instanz von Visio konnte nicht erzeugt werden."
```

Listing 11.5 Ausdrucken eines Zeichnungsblatts mittels Fernsteuerung von Visio (*Fortsetzung*)

```
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, 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.

CD-ROM

Das obige Beispiel finden Sie in der Beispieldatei *Bsp11_01.docm* 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
    Dim intAntwort As Integer

    #If EARLYBINDING Then
        'Wird mit Early Binding gearbeitet, muss der Verweis auf
        'die "Microsoft Outlook 14.0 Object Library" aktiviert werden.
        Dim olApp As Outlook.Application
        Dim olOrdner As Outlook.MAPIFolder
        Dim olKalenderEintrag As Object

        'Neue Outlook-Instanz erzeugen
        Set olApp = New Outlook.Application
    #Else
        Const olFolderCalendar As Integer = 9
        Dim olApp As Object
        Dim olOrdner As Object
        Dim olKalenderEintrag As Object

        'Neue Outlook-Instanz erzeugen
        Set olApp = CreateObject("Outlook.Application")
    #End If

    'Prüfen ob eine neue Instanz erzeugt werden konnte
    If Not (olApp Is Nothing) Then
        With olApp

            'Applikation am Bildschirm verbergen
            .Visible = False 'Eigenschaft nicht vorhanden

            'Kalender-Ordner setzen
            Set olOrdner = olApp.Session.GetDefaultFolder(olFolderCalendar)
            Set olKalenderEintrag = olOrdner.Items(1)

            'Kalendereintrag ausdrucken
            intAntwort = MsgBox("Soll der Ausdruck erstellt werden?", vbYesNo)
            If intAntwort = vbYes Then
                olKalenderEintrag.PrintOut
            End If
            .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, kann diese nur als versteckte Instanz ferngesteuert werden. Anders verhält es sich,

wenn eine bereits aktive Instanz gesteuert wird. Hier kann die `Visible`-Eigenschaft umgeschaltet werden.

```
'Visible = False    'Eigenschaft nicht vorhanden
```

Wird der Kalender nicht explizit sortiert, entspricht der erste Eintrag im Kalender nicht unbedingt dem Eintrag mit dem kleinsten Datum. Hier handelt es sich um den zuerst erfassten Kalendereintrag.

```
Set o1KalenderEintrag = o1Ordner.Items(1)
```

CD-ROM Das obige Beispiel finden Sie in der Beispieldatei *Bsp11_01.docm* 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« ab Seite 594 erläutert.

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.dotm* 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 »AutoMakros« 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 14.0 Object Library" aktiviert werden.
Dim olAnw As Outlook.Application
Dim olOrdner As Outlook.MAPIFolder
Dim olKontakt As Object
Dim olItem As Outlook.Items
Dim olItemFilter As Outlook.Items
#Else
Const olFolderContacts As Integer = 10
Const olContact As Integer = 40
Dim olAnw As Object
Dim olOrdner As Object
Dim olKontakt As Object
Dim olItem As Object
Dim olItemFilter As Object
#End If

Dim bOutlookNeuGestartet As Boolean

Public Function funcOL_Starten()
'Aktive Outlook-Instanz ermitteln ...
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

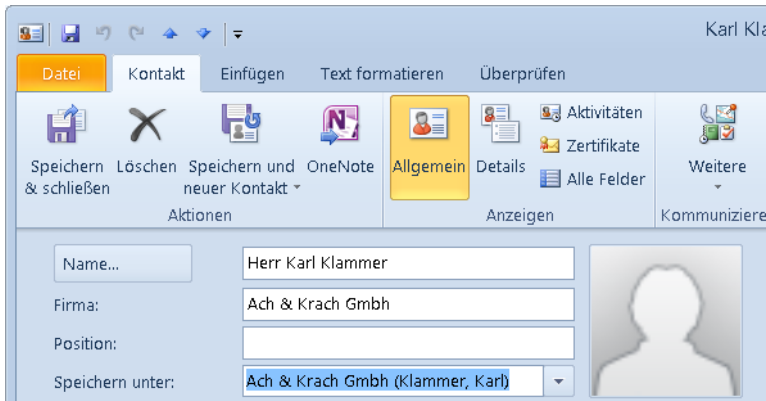
        'Kontakte-Ordner setzen und sortieren
        Set olOrdner = olAnw.Session.GetDefaultFolder(olFolderContacts)
        Set olItem = olOrdner.Items
        olItem.Sort "[FileAs]"

        '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 werden 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)
```

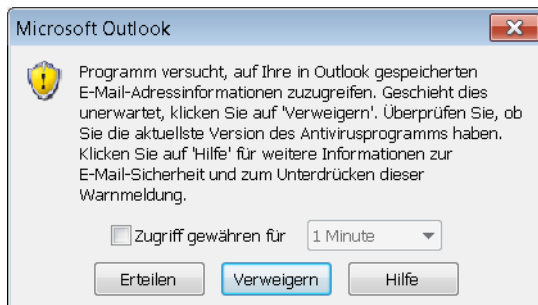
HINWEIS

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 (olItemFilter(1)) aus gefilterten Datensätzen eingelesen.

Im aktuellen Beispiel taucht die Sicherheitswarnung (siehe 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.

CD-ROM

Das dargestellte Beispiel finden Sie in der Beispieldatei *Bsp11_02a.dotm* 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« ab Seite 584 beschrieben.

Sobald die Verbindung zu dem Outlook-Objekt oder der Outlook-Instanz besteht, wird ein neues MailItem-Objekt erstellt, das eine neue Mail darstellt:

```
Set olEmailItem = olApp.CreateItem(olMailItem)
```

Ist das neue MailItem-Objekt erstellt, wird es, wie in Listing 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 Ihr 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 err_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
            bSaved = ActiveDocument.Saved
        ElseIf intRet = 0 Or ActiveDocument.Path = "" Then ' Abbruch gewählt
            MsgBox "Das Dokument wurde nicht gespeichert, das Makro wird beendet.", _
                vbCritical, c_Title
            GoTo end_Sub
        End If
    ElseIf ActiveDocument.Saved = False Then
        intRet = MsgBox("Das aktuelle Dokument muss gespeichert werden" & vbCrLf & _
            "Jetzt speichern?", vbInformation + vbYesNo, c_Title)
        If intRet = vbYes Then
            ActiveDocument.Save
        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
```

CD-ROM Das obige Beispiel finden Sie in der Beispieldatei *Bsp11_02b.docm* 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« ab Seite 594 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 Codemodule 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

    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 nach einer Rückfrage der Bericht gedruckt und die Access-Instanz anschließend wieder geschlossen.

Listing 11.14 Einen Access-Bericht ausdrucken

```
Sub AccessBerichtDrucken()
    Dim intAntwort As Integer
    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:=""
    'Report ausdrucken
    intAntwort = MsgBox("Soll der Ausdruck erstellt werden?", vbYesNo)
    If intAntwort = vbYes Then
        accApp.DoCmd.OpenReport ReportName="Katalog", View:=acViewNormal, _
            FilterName="", WhereCondition:=""
    End If
    accApp.CloseCurrentDatabase

Aussteigen:
    Set accApp = Nothing
Exit Sub
```

Listing 11.14 Einen Access-Bericht ausdrucken (*Fortsetzung*)

```
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
```

CD-ROM

Die Beispieldatei *Bsp11_03.docm* 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 Desktoprechner 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 letz-

ten Jahrhunderts wurde ADO (ActiveX Data Objects) entwickelt, und seit kurzem werden XML und ADO.NET – Teil von .NET Framework – 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.

Im Menüband 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

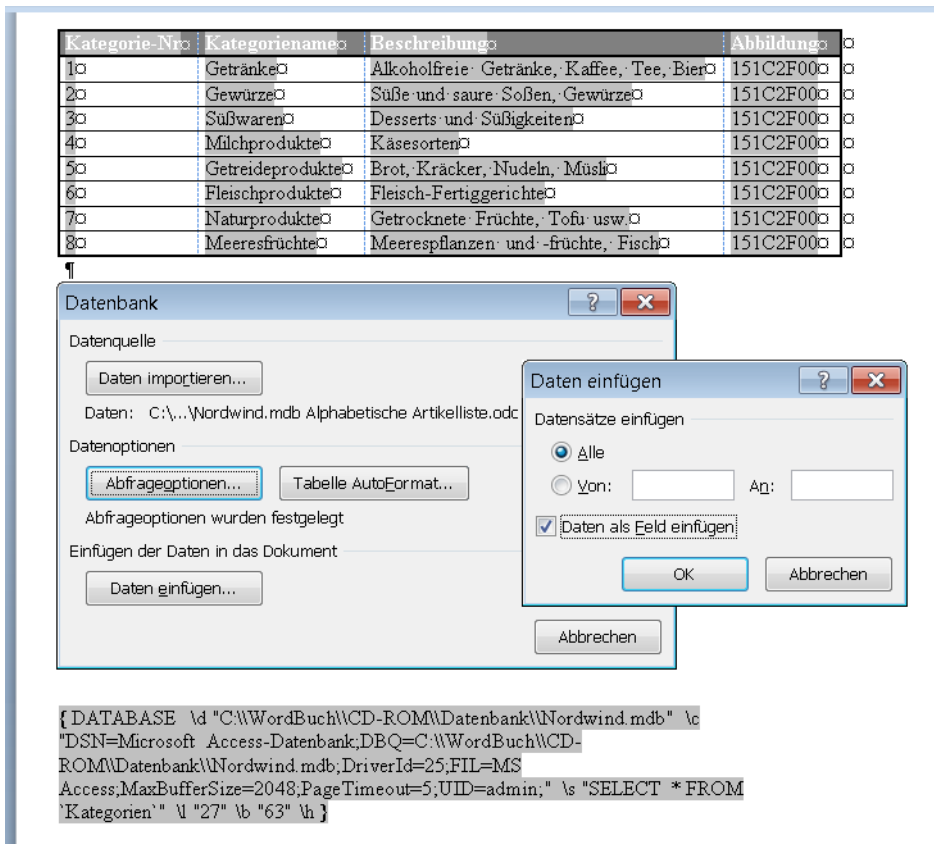
Wir empfehlen diesen Weg auch dem Entwickler, um die Syntax für die *Database*-Feldfunktion zu ermitteln. Das Word-Objektmodell stellt zwar eine *InsertDatabase*-Methode zur Verfügung, diese ist jedoch schwer zu bedienen. Mehr Informationen zu den Feldfunktionen sind in Kapitel 7 zusammengefasst.

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.

Die resultierende *Database*-Feldfunktion, die im Text-Argument der *Fields.Add*-Methode zu benutzen ist, erscheint unten in der Abbildung. Sie enthält die volle Pfadangabe zur Datenbank (relative Pfadangaben werden von der *Database*-Feldfunktion nicht unterstützt), die Verbindungsangabe für den ODBC-Treiber sowie eine *Select*-Anweisung, um die gewünschten Datensätze (in diesem Fall alle) festzulegen. Ferner bestimmen die Schalter *\l* und *\b* das Tabellen-AutoFormat und der Schalter *\h*, dass die Tabelle im HTML-Format von Word darzustellen ist.

Abbildg. 11.5 Daten aus einer Access-Datenbank mit der Feldfunktion *Database* in Word als Tabelle verknüpfen

CD-ROM

Die Beispieldatei *Bsp11_03.docm* 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 anwendungsspezifischen 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.connectionstrings.com/> sowie <http://www.carlprothman.net/Default.aspx?tabid=81>.

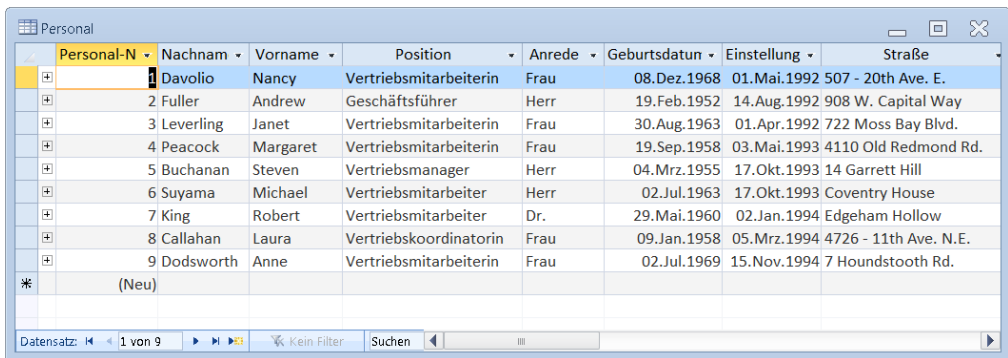
HINWEIS

Während der Arbeit an diesem Buch war die erwähnte Seite mit den Verbindungsinformationen zum neuen Access-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.dotm*) wird eine Liste der Namen erzeugen und zur Auswahl vorlegen, sodass der Anwender einen Brief an die ausgewählte Person schreiben kann. Dabei werden Adresse, Betreffzeile sowie Anrede vom Programm in das neue Dokument übernommen.

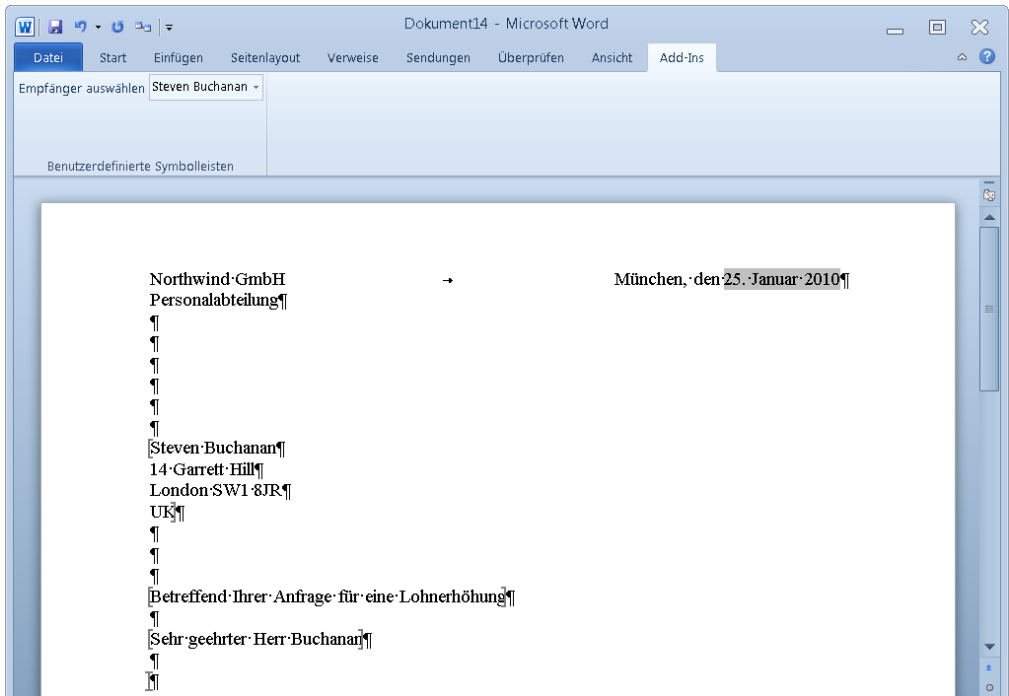
Abbildg. 11.6 Die *Personal*-Tabelle der Access-Datenbank *Nordwind.mdb*



Personal-N	Nachnam	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.
*	(Neu)						

Das Resultat sowie das Steuerelement mit der Liste in der Registerkarte *Add-Ins* sehen Sie in Abbildung 11.7. Um die eingefügten Daten sind Textmarkenklammern ersichtlich. Die Textmarken kennzeichnen die Zielbereiche und werden nach dem Einfügen der Daten wieder hergestellt (Textmarken wurden in Kapitel 6 näher vorgestellt).

Das Steuerelement wird beim Erstellen des Dokuments über das *CommandBars*-Objektmodell 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 Schaltfläche ist also temporär und wird beim Schließen des Dokuments gelöscht. 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

HINWEIS

Obwohl die neue Menüband-Technologie und das Menüband die Symbolleisten in der Office-Umgebung ersetzt haben, bleibt das CommandBars-Objektmodell für gewisse Aufgaben nützlich. Es ist die einzige Methode, Steuerelemente im Menüband zur Laufzeit dynamisch zu erzeugen und zu entfernen. Einziger Nachteil ist, dass diese zwingend in der Registerkarte *Add-Ins* erscheinen.

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 (ab 2.1) 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 *Brief schreiben* (das in der Registerkarte *Add-In* erscheint) 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 nächste Datensatz anzuwählen ist (RS.MoveNext).

Als letzte wichtige Handlung wird der Recordset geschlossen und die Variablen 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, welcher 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, sodass 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.
    Set RS.ActiveConnection = Nothing
```

Listing 11.15 Daten über ADO-Verbindungen und Recordsets lesen und in ein Word-Dokument schreiben (*Fortsetzung*)

```

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 & vbCrLf & RS.Fields("Straße").Value & _
        vbCrLf & RS.Fields("Ort").Value & " " & RS.Fields("PLZ").Value & _
        vbCrLf & RS.Fields("Land").Value, doc, False
    'Den Benutzer zur Eingabe der Betreffzeile auffordern.
    DatenSchreiben "Betreffzeile", InputBox("Betreffzeile eingeben"), doc, False
    'Den Ausdruck "Sehr geehrte(r)" dem Geschlecht des Empfängers anpassen.
    If RS.Fields("Anrede").Value = "Herr" Then
        strAnredeZusatz = "r"
    End If
    DatenSchreiben "Anrede", "Sehr geehrte" & strAnredeZusatz & " " & _
        & RS.Fields("Anrede").Value & " " & RS.Fields("Nachname").Value, doc, False
    DatenSchreiben "InhaltAnfang", "", doc, True
    RS.Close
    Set RS = Nothing
End Sub

'Falls die Textmarke nicht existiert, wird "Falsch" zurückgegeben.

```

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
```

CD-ROM Die Beispieldateien *Bsp11_04.dotm* und *Bsp11_04_AccDB.dotm* mit Code für die Verbindung zur *Nordwind.accdb* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap11*. Die Access 2010 Datenbank *Nordwind.accdb* befindet sich im Ordner *\Datenbank*.

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 Felder eines nach dem andern mit Daten zu bestücken, werden die Daten in Zeichenketten gesammelt und mit der SQL-Anweisung `INSERT INTO` der Datentabelle hinzugefügt.

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

Northwind-GmbH
Personalabteilung

München, den 23. August 2010

Herr [Vorname] [Nachname]
[Straße]
[PLZ] [Ort]

SUBMIT Adresse

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
```

Listing 11.16 Daten aus Word-Formularfeldern über eine ADO-Verbindung als neue Zeile einer Excel-Tabelle hinzufügen (Fortsetzung)

```

Dim ffld As Word.FormField
Dim doc As Word.Document
Dim strFeldListe As String
Dim strWertListe As String
Dim strSQL As String

Set doc = ActiveDocument
strSQL = ""
strFeldListe = "("
strWertListe = "("
For Each 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""

'Für XSLX-Dateien:
'conn.Open "Provider=Microsoft.ACE.OLEDB.12.0;" & _
    "Data Source=c:\WordBuch\Datenbank\Personalstamm.xlsx;" & _
    "Extended Properties=""Excel 12.0;HDR=Yes""

conn.Execute strSQL
conn.Close
Set conn = Nothing
End Sub

```

CD-ROM

Die Beispieldatei *Bsp11_05.dotm* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap11*. Die Excel-Arbeitsmappen *Personalstamm.xls* sowie *Personalstamm.xlsx* mit dem Arbeitsblatt *PersonalTabelle* sind 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 574 ff.) oder zur Berechnung von komplexen Funktionen (Seite 576) herangezogen werden kann
- Beim Steuern von PowerPoint (Seite 577 ff.) wurde verdeutlicht, wie der Inhalt einer Präsentation in ein Dokument eingelesen (Seite 580) werden kann
- Ein einfaches Beispiel zum Steuern von Visio wurde auf Seite 582 behandelt
- Beim Zugriff auf Outlook (Seite 584 ff.) wurde vermittelt, wie auf die Kontakte zugegriffen (Seite 586) oder ein Dokument als E-Mail versendet (Seite 591) werden kann
- Das Fernsteuern von Access (Seite 592 ff.) und der Zugriff auf Datenbanken (Seite 594 ff.) wurde ebenfalls behandelt

Kapitel 12

Eingebettete Objekte

In diesem Kapitel:

Excel-Tabellenobjekte	609
Office (Excel)-Diagramme	616
Microsoft Graph-Diagramme	623
WordArt	628
SmartArt	633
Zusammenfassung	646

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 sogenannte 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 sogenannte »In-place editing«: die Menüs 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-Serveranwendung 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, überprüfen Sie die automatisch geladenen Programme, wie Antiviren- und Desktopverwaltungs-Anwendungen, 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 Einstellung der Option *Bild einfügen als* (im Abschnitt *Ausschneiden, Kopieren und Einfügen* der Registerkarte *Erweitert in Datei/Optionen*) 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 97-Objekts aufgezeichnet:

```
Selection.InlineShapes.AddOLEObject ClassType:="Excel.Sheet.8", LinkToFile:=False, _
    DisplayAsIcon:=False
```

HINWEIS

Der OLE-Typ *Excel.Sheet.8* wurde mit Office 97 eingeführt und funktioniert mit allen späteren Versionen von Office. Mit Einführung des neuen OpenXML Dateiformats sowie die neuen grafischen Möglichkeiten in Office 2007 wurde ein neuer OLE-Typ fällig: *Excel.Sheet.12*. Im folgenden Text werden wir die Ausdrücke »Excel 97« und »Excel 2007« Kalkulationstabelle verwenden, um die zwei verschiedenen OLE-Typen zu unterscheiden.

Die Automatisierung von Excel-Tabellen in Word 2007 oder 2010 unterscheidet sich, trotz des neuen Dateiformats, nicht wesentlich von früheren Versionen. Tabellen im alten binären Dokumentformat sowie aus konvertierten Arbeitsmappen werden abgearbeitet. Dokumente mit gemisch-

ten 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 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 6 im Abschnitt zu Grafiken.

OLEFormat

Da OLE-Objekte in den `InlineShapes`- und `Shapes`-Auflistungen geführt werden, werden sie von Word offensichtlich als grafische Objekte betrachtet. Und tatsächlich stehen die üblichen Befehle zum Formatieren und Positionieren zur Verfügung. Aber wie lassen sich OLE-Objekte in ihrer Ursprungsumgebung öffnen?

Auch hier hilft der Makrorekorder. Der aufgezeichnete Code zeigt, dass ein Zugang zu den OLE-Fähigkeiten durch die Eigenschaft `OLEFormat` und die Aktivierung des Objekts über die Methode `DoVerb` erreicht wird:

```
Selection.InlineShapes(1).OLEFormat.DoVerb VerbIndex:=wdOLEVerbPrimary
```

DoVerb

Wie das OLE-Objekt durch die `DoVerb`-Methode zu aktivieren ist, wird über das Argument `VerbIndex` geregelt. Da die hierfür benötigten `WdOLEVerb`-Konstantenwerte 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`-Konstantenwerte unterstützt.

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 Objektvariablen 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
```

CD-ROM

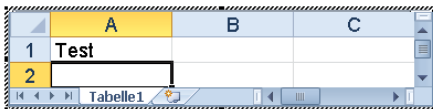
Die Beispieldatei *Bsp12_01.docm* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap12*.

Danach muss mit dem Objektmodell des OLE-Servers gearbeitet werden, wie die letzten Codezeilen veranschaulichen. Das Resultat sehen Sie in Abbildung 12.1.

PROFITIPP

Der Makrorekorder von Word kann in einer OLE-Serveranwendung nicht weiterhelfen, unter Umständen kann jedoch der Makrorekorder der OLE-Serveranwendung genutzt werden. 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 Objektbibliothek sucht. Muss aus irgendeinem Grund ein allgemeines Objekt wie *Selection* verwendet werden, sollte die Anwendung 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



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.

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.

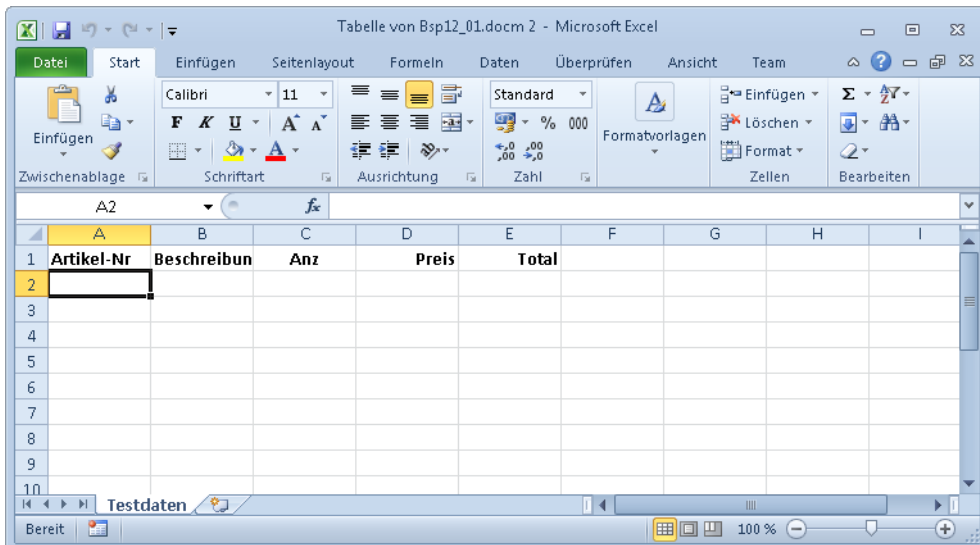
Excel-Tabellenobjekte

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.)

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 vom Typ `Object` 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.

Abbildg. 12.2 Ein in einem Word-Dokument eingebettetes und aktiviertes Excel-Arbeitsblatt



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.

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
    'Syntax für Excel 97 Kalkulationstabelle
    'Set o = ActiveDocument.InlineShapes.AddOLEObject( _
    '    "Excel.Sheet.8", Range:=rng).OLEFormat.Object
    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
```

HINWEIS Die Version von Excel, worauf ein eingebettetes Excel-Objekt basiert, spielt keine wesentliche Rolle, solange der Code keine versionsspezifische Funktionalität anspricht. Ein Dokument kann sogar Tabellen beider Art – Excel.Sheet.8 sowie Excel.Sheet.12 – enthalten.

Unter Umständen zeigt das Objekt mehr Spalten an 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 sichtbaren 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 und 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

    'Syntax für Excel 2003 und früher
    ' strArbeitsmappenPfad = ActiveDocument.Path & "\Bsp12_01.xls"
    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.
```

Listing 12.3 Eine Excel-Mappe erstellen und anschließend in Word importieren (Fortsetzung)

```
xlBlatt.UsedRange.Font.Bold = True
'Um sicher zu gehen, dass die zweite Zeile sichtbar ist,
'müssen Daten darin stehen.
xlBlatt.Range("A2").Value = "Start"
xlBlatt.Range("A2").Select
'Überflüssige Arbeitsblätter entfernen.
For lZaehler = xlMappe.Sheets.Count To 2 Step -1
    xlMappe.Sheets(lZaehler).Delete
Next
'Die Mappe speichern, schließen und Excel beenden.
xlMappe.Close SaveChanges:=True, FileName:=strArbeitsmappenPfad
xlApp.Quit
Set xlBlatt = Nothing
Set xlMappe = Nothing
Set xlApp = Nothing

'In Word das Tabellen-Objekt einfügen und aktivieren.
Set rng = ActiveDocument.Range
rng.Collapse wdCollapseEnd
Set ils = rng.InlineShapes.AddOLEObject(FileName:=strArbeitsmappenPfad, _
    LinkToFile:=False, DisplayAsIcon:=False)
ils.OLEFormat.DoVerb VerbIndex:=wdOLEVerbPrimary
End Sub
```


TIPP

Für .NET-Programmierer stellt sich die Frage, ob die Anpassung eines Word-Dokuments nicht besser durch die Bearbeitung des XML-Dateiformats zu erreichen ist. Dies erfolgt ohne die Einbeziehung von Word- und Excel-Anwendungen, mit den herkömmlichen .NET Framework Packaging- und XML-Klassen. Eine kurze Einführung in das Prinzip der Bearbeitung eines Word-Dokuments über das XML-Dateiformat wird im Kapitel 22 vorgestellt.

Listing 12.4 (.NET): 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;
        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 =
```


Listing 12.4 (.NET): 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)*

```

    xl.XlHAlign.xlHAlignRight;
    //Die Zahlenformatierung festlegen.
    xlBlatt.get_Range("D1", Type.Missing).EntireColumn.NumberFormat = "#,##0.00";
    xlBlatt.get_Range("E1", Type.Missing).EntireColumn.NumberFormat = "#,##0.00 ?";
    //Die Spaltenüberschriften eingeben ...
    xlBlatt.get_Range("A1", Type.Missing).Value2 = "Artikel-Nr";
    xlBlatt.get_Range("B1", Type.Missing).Value2 = "Beschreibung";
    xlBlatt.get_Range("C1", Type.Missing).Value2 = "Anz";
    xlBlatt.get_Range("D1", Type.Missing).Value2 = "Preis";
    xlBlatt.get_Range("E1", Type.Missing).Value2 = "Total";
    //... und formatieren.
    xlBlatt.UsedRange.Font.Bold = true;
    //Um sicher zu gehen, dass die zweite Zeile sichtbar ist,
    //müssen Daten darin stehen.
    xlBlatt.get_Range("A2", Type.Missing).Value2 = "Start";
    xlBlatt.get_Range("A2", Type.Missing).Select();
    //Überflüssige Arbeitsblätter entfernen.
    for (int zaehler = xlMappe.Sheets.Count; zaehler > 1; zaehler --)
    {
        xl.Worksheet xlVorigesBlatt = (xl.Worksheet) xlMappe.Worksheets[zaehler];
        xlVorigesBlatt.Delete();
        xlVorigesBlatt = null;
    }
    //Die Mappe speichern, schließen und Excel beenden.
    xlMappe.Close(true, arbeitsmappenPfad, Type.Missing);
    xlApp.Quit();
    xlBlatt = null;
    xlMappe = null;
    wdMarshal.ReleaseComObject(xlApp);
    xlApp = null;

    wd.Application wdApp = this.GetWordApp();

    //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 objRng);
    object oleVerbPrimary = wd.WdOLEVerb.wdOLEVerbPrimary;
    rng = null;
    ils.OLEFormat.DoVerb(ref oleVerbPrimary);
    //Das Word-Fenster zuvorderst bringen.
    wdApp.Visible = true;
    wdApp.Activate();
    wdMarshal.ReleaseComObject(wdApp);
    wdApp = null;
}

```

Listing 12.4 (.NET): 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*)

```
catch (System.Exception ex)
{
    MessageBox.Show("Fehlermeldung: " + ex.Message + " in " + ex.Source);
}
}
```

CD-ROM

Die Beispieldatei *Bsp12_01.docm* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap12*.

Bearbei-
ten

In Listing 12.1 (Seite 608) 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 *ExcelObjekt-Bearbeiten* 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
    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
    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
    End With
End Sub

```

Listing 12.5 Alle Excel-Tabellenobjekte eines Dokuments in einem Excel-Fenster öffnen und formatieren (Fortsetzung)

```
.Font.FontStyle = "Fett"
.Font.Color = RGB(255, 255, 255)
End With
lAnzZeilen = xlRange.Rows.Count
lAnzSpalten = xlRange.Columns.Count
'Die restlichen Zeilen, falls vorhanden, hellgrau hinterlegen
If lAnzZeilen > 1 Then
    lOffsetZeile = 2
    Set xlRange = xlRange.Range(xlRange.Cells(lOffsetZeile, 1), _
        xlRange.Cells(lAnzZeilen, lAnzSpalten))
    With xlRange
        .Interior.Color = RGB(200, 200, 200)
        .Interior.Pattern = xlSolid
    End With
End If
'Die Mappe schließen
xlMappe.Close False
Set xlMappe = Nothing
DoEvents
'Falls Excel vor der Ausführung nicht schon lief, beenden
If Not bExcel Then
    xlApp.Quit
End If
Set xlApp = Nothing
End Sub
```

TIPP

Auch wenn eine In-place-Aktivierung spezifiziert wird, öffnet Word unter gewissen Umständen ein OLE-Objekt in einem getrennten Anwendungsfenster, u.a.:

- Die Feldcodeanzeige ist für das Dokument aktiviert
- Das Objekt ist mit einer Datei verknüpft
- Word hat zu wenig Ressourcen, um die In-place-Aktivierung zu unterstützen

CD-ROM

Die Beispieldatei *Bsp12_01.docm* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap12*.

Office (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. Anders als bei den Tabellen ist der Übergang zwischen Office-Versionen nicht fließend.

Wie in Kapitel 7 erwähnt, erfolgte in Office 2007 ein Schnitt in der Herstellung und der Darstellung grafischer Objekte. Nicht nur die Grundlagen herkömmlicher Grafiken, sondern auch der Excel-Diagramme wurden gründlich überarbeitet. Das dazugehörige Objektmodell für die neuen Excel-Diagramme war in der ursprünglichen Version von Office 2007 nicht vorhanden – es war nicht möglich, diese neuen Diagramme programmtechnisch in Word zu erstellen. Inzwischen wurde dieses Manko

behalten. Das Objektmodell wurde mit dem Service Pack 2 geliefert und ist ebenfalls im Lieferumfang von Office 2010 enthalten.

In diesem Abschnitt werden wir den Umgang mit den neuen Office-Diagrammen in Word kurz vorstellen. Das neue Modell hat, gegenüber dem alten, einige Vorteile. Einer davon ist, dass es nicht mehr unbedingt notwendig ist, dem Projekt eine Referenz zum Excel-Objektmodell hinzuzufügen, um von IntelliSense profitieren zu können. Das Objektmodell für Diagramme wird automatisch durch das der Word-Anwendung zugänglich gemacht.

WICHTIG Eine logische Folge davon ist, dass diese Objekte mit dem Präfix »Word.« deklariert werden müssen, statt »Excel.« Der Office-Programmierer ist es gewöhnt, beispielsweise ein Diagramm als `Dim MeinDiagramm as Excel.Chart` zu deklarieren. Neu sieht die Deklaration folgendermaßen aus: `Dim MeinDiagramm as Word.Chart`.

Ein weiterer Unterschied besteht darin, dass die neuen Diagrammobjekte nicht durch Feldfunktionen verwaltet werden. Sie sind vollwertige grafische Objekte, mit allen im Kapitel 6 vorgestellten Vor- und Nachteilen (beispielsweise, die Verwaltung der Verknüpfungen). Früher wurden die Diagrammobjekte als OLE-Objekte eingefügt. Dies ist jetzt nicht mehr der Fall. Microsoft hat die Einbindung der neuen eingebetteten Objekte auf eine neue Stufe gesetzt. Die Einbindung und Verknüpfung erfolgen gänzlich im Hintergrund, transparent für den Entwickler sowie den Benutzer.

CD-ROM Eine Zeitlang werden wir in einer Welt arbeiten, die mit beiden Diagrammtypen umgehen muss – sogar im gleichen Dokument. Denn die Word-Anwendung unterstützt beide Arten von Diagrammen. Deshalb finden Sie das Material zu Diagrammen des Typs *Excel.Chart.8* aus der zweiten Auflage auf der CD zum Buch im Ordner *Beilagen/Kap12_XL8Diagramme*.

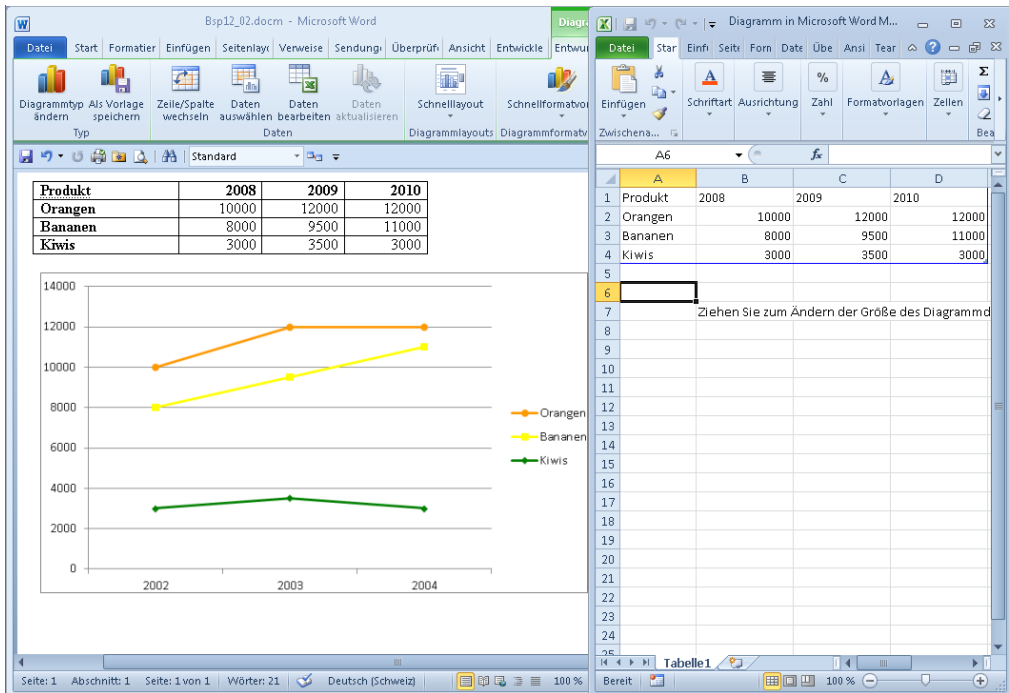
Der Makrorekorder verrät folgende Syntax, um ein Office-Diagramm einzufügen:

```
Selection.InlineShapes.AddChart Type:=51
'51 = XLChartType xlColumnClustered
```

Auffallend ist, dass es für die Bearbeitung der dahinterliegenden Daten keine »In-place-editing«-Schnittstelle mehr gibt: Die Daten werden in einem getrennten Excel-Fenster neben dem Word-Fenster mit dem Diagramm angezeigt. Es ist nicht möglich, dieses Verhalten zu unterbinden, auch nicht, wenn das Diagramm durch Automatisierung erstellt wird.

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. Der Beispielcode steht in Listing 12.6.

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.

Abbildg. 12.5 Das Excel-Tabellenblatt wird neben dem Word-Fenster mit dem Diagramm eingeblendet


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 bereitliegen, wird das Diagramm-Objekt eingefügt. Die Methode *AddChart* ist für die *InlineShapes*- sowie die *Shapes*-Auflistung vorhanden. Beide Argumente für ein *InlineShape*, *Type* sowie *Range*, sind fakultativ. Das erste legt die Diagrammart fest; das zweite, an welcher Stelle dieses im Dokument eingefügt werden soll.

Die sechs Argumente für ein Diagramm mit Textflussformatierung (ein *Shape*-Objekt also) sind ebenfalls fakultativ. Wie bei anderen grafischen Objekten, legen die Argumente zwei bis fünf die Position und Größe des Objekts fest, während mit dem ersten die Diagrammart und dem letzten der Verankerungsbereich angegeben werden: *AddChart*(*Type*, *Left*, *Top*, *Width*, *Height*, *Anchor*)

WICHTIG

Das *Type*-Argument verlangt ein Mitglied der Enumeration *xlChartType*. Ohne Verweis auf das Excel-Objektmodell werden diese im Word Objektkatalog nicht angezeigt, sie werden jedoch von Word akzeptiert. Dies ist jedoch nicht für alle Konstanten aus dem Excel-Objektmodell der Fall. Solche Objekte müssen als Datentyp *Object* deklariert und für deren Argumente die Ganzzahlwerte der Enumerationen verwendet werden.

Das eingefügte Objekt wird zuerst einer Objektvariablen des Typs *InlineShape* zugewiesen. Danach erfolgt der Test, ob dieses ein Diagramm enthält. Wenn ja, kann es einer Objektvariablen des Typs *Diagramm* zugewiesen werden:

```
Set ils = doc.InlineShapes.AddChart(xlColumnClustered, rng)
If ils.HasChart Then
    Set Diagramm = ils.Chart
```

Über dieses Objekt werden die dahinterliegende Arbeitsmappe sowie deren Arbeitsblatt angesprochen:

```
Set xlMappe = Diagramm.ChartData.Workbook
Set xlBlatt = xlMappe.Sheets(1)
```

Die Spaltenüberschriften, die als Diagrammlegende erscheinen, werden anhand der Feldnamen des ADO-Recordsets angelegt.

Die Datentabelle für diese neuartigen Diagramme wird in einem ListObject (Excel-Tabelle) verwaltet – das Diagramm ist damit verknüpft. Die Beispieldaten dürfen also nicht einfach gelöscht werden – der Code muss über das ListObject arbeiten, um nicht benötigte Spalten und Zeilen zu entfernen. Danach werden die Daten, dank Excels CopyFromRecordset-Methode, in einem Schritt eingefügt:

```
Set lo = xlBlatt.ListObjects(1)
'Nicht benötigte Zeilen in der Excel-Tabelle löschen
For i = rs.RecordCount + 1 To lo.ListRows.Count
    lo.ListRows(i).Delete
Next
'Nicht benötigte Spalten löschen
For i = rs.Fields.Count + 1 To lo.ListColumns.Count
    lo.ListColumns(i).Delete
Next
lo.DataBodyRange.CopyFromRecordset rs
```

Schließlich verpasst die Prozedur *DiagrammFormatieren* dem Diagramm den letzten Schliff. Der Diagrammtyp wird festgelegt. 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 Diagramm As Word.Chart
    Dim doc As Word.Document
    Dim rng As Word.Range
    Dim ils As Word.InlineShape
    Dim tbl As Word.Table

    Dim xlBlatt As Object ' Excel.Worksheet
    Dim xlMappe As Object ' Excel.Workbook
    Dim lo As Object ' Excel.ListObject

    Dim rs As ADODB.Recordset
    Dim fld As ADODB.Field
    Dim lZaehler As Long, i As Long

    Set doc = ActiveDocument
    Set tbl = doc.Tables(1)
    Set rng = tbl.Range
```

Listing 12.6 Ein Excel-Diagramm anhand einer Tabelle im Word-Dokument erstellen *(Fortsetzung)*

```

'Das Diagramm wird unter der Tabelle eingefügt.
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 ersten Datensatz angewählt werden.
rs.MoveFirst

'Das Diagramm im Dokument erstellen...
Set ils = doc.InlineShapes.AddChart(xlColumnClustered, rng)
If ils.HasChart Then
    Set Diagramm = ils.Chart
Else
    MsgBox "Das Diagramm konnte nicht erstellt werden."
    GoTo Cleanup
End If

'Die Arbeitsmappe und das Datenblatt Objektvariablen zuweisen.
Set xlMappe = Diagramm.ChartData.Workbook
Set xlBlatt = xlMappe.Sheets(1)

'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

'Diagramme basieren auf einer Tabelle (ListObject) in Excel;
'wir müssen mit diesem Bereich arbeiten.
Set lo = xlBlatt.ListObjects(1)
'Nicht benötigte Zeilen in der Excel-Tabelle löschen
For i = rs.RecordCount + 1 To lo.ListRows.Count
    lo.ListRows(i).Delete
Next
'Nicht benötigte Spalten löschen
For i = rs.Fields.Count + 1 To lo.ListColumns.Count
    lo.ListColumns(i).Delete
Next

'Die Daten einfügen.
lo.DataBodyRange.CopyFromRecordset rs
DiagrammFormatieren Diagramm, xlBlatt
'Die Anpassungen im Dokument speichern
'(wird dies nicht gemacht, können sie
'beim Schliessen der Excel-Mappe verloren gehen).
doc.Save
xlMappe.Close

Cleanup:
'Aufräumarbeiten

```


Listing 12.6 Ein Excel-Diagramm anhand einer Tabelle im Word-Dokument erstellen (Fortsetzung)

```

rs.Close
Set rs = Nothing
Set xlMappe = Nothing
Set xlBlatt = Nothing
Set Diagramm = Nothing
End Sub

Sub DatenLaden(ByRef rs As ADODB.Recordset, tbl As Word.Table)
    Dim row As Word.Row
    Dim cel As Word.Cell
    Dim rng As Word.Range
    Dim fld As ADODB.Field
    Dim lAnzFelder As Long
    Dim lZaehler As Long
    Dim lFeldTyp As Long
    Dim strZellenInhalt As String

    rs.CursorLocation = adUseClient
    rs.CursorType = adOpenDynamic
    rs.LockType = adLockOptimistic
    lAnzFelder = tbl.Columns.Count
    For Each row In tbl.Rows
        'Die erste Zeile (Spaltenüberschrift) enthält den Feldnamen.
        If row.Index = 1 Then
            'Die Felder des Recordsets definieren
            For Each cel In row.Cells
                Set rng = cel.Range
                'Sicherstellen, dass auch verborgener Text gelesen wird (Die erste Zelle enthält
                'eine Spaltenbezeichnung, weil ein Laufzeitfehler bei leerem Zelleninhalt
                'vorkommt)
                rng.TextRetrievalMode.IncludeHiddenText = True
                strZellenInhalt = TrimZelle(rng.Text) & ""
                'Der Feldtyp ist entweder eine Zeichenkette oder eine Zahl
                lFeldTyp = FeldTypErmitteln(strZellenInhalt)
                rs.Fields.Append Name:=strZellenInhalt, Type:=lFeldTyp, _
                DefinedSize:=30, Attrib:=adFldMayBeNull
                Set rng = Nothing
            Next
            rs.Open
        Else
            'Das Recordset mit Datensätzen füllen...
            rs.AddNew
            '... ein Datensatz pro Zeile
            For lZaehler = 1 To lAnzFelder
                rs.Fields(lZaehler - 1).Value = _
                TrimZelle(row.Cells(lZaehler).Range.Text)
            Next lZaehler
        End If
    Next
End Sub

Sub DiagrammFormatieren(Diagramm As Word.Chart, lo As Object) 'Excel.ListObject
    With Diagramm
        .ChartType = 4 'xlLine
        .Axes Type:=1 'xlCategory
    End With
End Sub

```

Listing 12.6 Ein Excel-Diagramm anhand einer Tabelle im Word-Dokument erstellen (Fortsetzung)

```

    'Orangen sind orange.
    DatenserieFormatieren .SeriesCollection(1), 8, _
        5, RGB(255, 153, 0) 'xlMarkerStyleCircle
    'Bananen sind gelb.
    DatenserieFormatieren .SeriesCollection(2), 1, _
        5, RGB(255, 255, 0) 'xlMarkerStyleSquare
    'Kiwis sind grün.
    DatenserieFormatieren .SeriesCollection(3), 2, _
        5, RGB(0, 128, 0) 'xlMarkerStyleDiamond
    .PlotBy = 1 'xlRows
End With
End Sub

Sub DatenserieFormatieren(DatenSerie As Word.Series, _
    xlDatenPunktStil As Long, xlDatenPunktGrösse As Long, _
    colorDatenPunktFarbe)

    With DatenSerie
        .MarkerStyle = xlDatenPunktStil
        .MarkerSize = xlDatenPunktGrösse
        .Border.Color = colorDatenPunktFarbe
        .MarkerBackgroundColor = colorDatenPunktFarbe
        .MarkerForegroundColor = colorDatenPunktFarbe
    End With
End Sub

Function TrimZelle(str)
    str = Mid(str, 1, Len(str) - 2)
    TrimZelle = str
End Function

Function FeldTypErmitteln(str) As Long
    If IsNumeric(str) Then
        FeldTypErmitteln = 3
    Else
        FeldTypErmitteln = 202
    End If
End Function

```

Es ist auch möglich, über einen Umweg, ein eingebettetes Office-Diagramm mit der alten OLE-Technologie im Dokument zu erstellen. Die OLE-Klasse des Objekts ist jedoch eine Excel-Kalkulationstabelle – *Excel.Sheet.12* – nicht ein Diagramm. In diesem Fall wird zuerst die Kalkulationstabelle erstellt. Anschließend wird ein darauf basierendes Diagrammblatt als erstes Blatt der Arbeitsmappe eingefügt, wie das Listing 12.7 veranschaulicht.

Listing 12.7 Ein Excel-Diagramm kann auf Basis eines bestehenden Excel-Tabellenobjekts eingefügt werden

```

Sub OLEDiagrammErstellen()
    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

```

Listing 12.7 Ein Excel-Diagramm kann auf Basis eines bestehenden Excel-Tabellenobjekts eingefügt werden (Fortsetzung)

```

Set doc = ActiveDocument
Set rng = doc.Content
'Das Tabellenobjekt 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 = "2007"
oSheet.Range("C1").Value = "2008"
oSheet.Range("D1").Value = "2009"
oSheet.Range("E1").Value = "2010"
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 x1Rng = oSheet.UsedRange
'Eine Diagrammblatt einfügen...
Set o = wb.Charts.Add
'...und mit dem Datenbereich verbinden.
o.SetSourceData x1Rng
'Als Linien-Diagramm festlegen
o.ChartType = xlLine
DiagrammFormatieren_2 o, oSheet 'Ruft eine Variation der Prozedur in Listing 12.6
End Sub

```

CD-ROM Die Beispieldatei *Bsp12_02.docm* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap12*.

Microsoft Graph-Diagramme



Die Funktionalität hinter Microsoft Graph ist die gleiche wie in Excel 2003 und früher. Nur die Schnittstelle für die Daten ist anders als in Excel, zudem bietet Microsoft Graph keine Unterstützung der Farbenpalette. Da Microsoft 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. Microsoft 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 Microsoft 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 im vorangehenden Abschnitt, um die Unterschiede und Gemeinsamkeiten der Objektmodelle 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 mit dem Ablauf für das Beispiel eines Diagramms des Typs `Excel.Chart.8`, auf der CD: 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 Microsoft 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 Microsoft 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 dies eine Eigenschaft des `Chart`-, nicht des `Application`-Objekts und daher für den erfahrenen Office-Entwickler entsprechend schwierig zu finden.

Im Gegensatz zum Excel-Beispiel im vorherigen Abschnitt, bei dem das Diagramm-Objekt am Schluss des Codes noch aktiv ist, wird das Microsoft Graph-Diagramm mit der `Quit`-Methode geschlossen. In Microsoft Graph funktioniert sie auch bei In-place-Aktivierung!

Listing 12.8 Ein Microsoft Graph-Diagramm einfügen und formatieren

```
Sub DiagrammEinfuegen()
    Dim grphDiagramm As Graph.Chart
    Dim grphDatenblatt As Graph.DataSheet
    Dim grphApp As Graph.Application
    Dim doc As Word.Document
    Dim rng As Word.Range
    Dim ils As Word.InlineShape
    Dim oleF As Word.OLEFormat
    Dim tbl As Word.Table
    Dim cel As Word.Cell
    Dim lZaehler As Long
    Dim lSpaltenZaehler As Long

    Set doc = ActiveDocument
    Set tbl = doc.Tables(1)
    Set rng = tbl.Range
    'Diagramm unter der Tabelle einfügen
    rng.Collapse Direction:=wdCollapseEnd
    rng.InsertAfter vbCrLf
```

Listing 12.8 Ein Microsoft Graph-Diagramm einfügen und formatieren (Fortsetzung)

```

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
        .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)

```

Listing 12.8 Ein Microsoft Graph-Diagramm einfügen und formatieren (Fortsetzung)

```

With DatenSerie
    .MarkerStyle = xlDatenPunktStil
    .MarkerSize = xlDatenPunktGrösse
    .Border.Color = colorDatenPunktFarbe
    .MarkerBackgroundColor = colorDatenPunktFarbe
    .MarkerForegroundColor = colorDatenPunktFarbe
End With
End Sub

```

Das Erstellen und Formatieren eines Microsoft Graph-Diagramms unterscheidet sich kaum von der Handhabung eines Excel-Diagramms des alten Typs. Aus Gründen der Rückwärtskompatibilität bleiben die meisten Befehle im Umgang mit Diagrammen die gleichen. Deshalb gilt der C#-Code in Listing 12.9 stellvertretend auch für Excel-Diagramme. Änderungen bezüglich des Word-Objektmodells entnehmen Sie der Diskussion im Abschnitt »Office (Excel)-Diagramme« ab Seite 616.

Listing 12.9 (.NET): Ein Microsoft Graph-Diagramm einfügen und formatieren (C#-Code)


```

private void DiagrammEinfuegen_CS()
{
    try
    {
        wd.Application wdApp = this.GetWordApp();
        if (wdApp != null)
        {
            wdApp.Visible = true;
            object docName = @"C:\WordBuch\Beispiele\Kap12\Bsp12_03.docm";
            wd.Document doc = wdApp.Documents.Open(ref docName, ref missing, ref missing,
                ref missing, ref missing, ref missing, ref missing, ref missing, ref missing,
                ref missing, ref missing, ref missing);
            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 objMissing,
                ref objRange).OLEFormat;
            //...und aktivieren
            oleF.DoVerb(ref objMissing);

            //Der Arbeitsmappe, dem Datenblatt sowie Diagrammblatt Objektvariablen zuweisen
            grph.Chart grphDiagramm = (grph.Chart) oleF.Object;
            grph.Application grphApp = grphDiagramm.Application;
            grph.DataSheet grphDatenblatt = grphApp.DataSheet;
            //Die vorhandenen Daten löschen.

```

Listing 12.9 (,.NET): Ein Microsoft Graph-Diagramm einfügen und formatieren (C#-Code) (Fortsetzung)

```

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;
}
else
{
    MessageBox.Show("Die Word-Anwendung konnte nicht gestartet werden.");
}
}
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 (.NET): Ein Microsoft 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;
    }

```

CD-ROM

Die Beispieldatei *Bsp12_03.docm* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap12*.

WordArt



Wie andere grafische Werkzeuge musste WordArt seit Office 2003 eine »Verschönerungskur« über sich ergehen lassen. In Office 2007 wurde die Änderung in Excel und PowerPoint vollzogen, diese wurden für Word erst in der Version 2010 zur Verfügung gestellt. Das neue WordArt besteht aus einem grafischen Textfeld, das mit den neuen grafischen Werkzeugen formatiert wurde. Dafür steht kein eigenes Objektmodell zur Verfügung.

Code aus Word 2003 und früher funktioniert weiterhin und wird hier kurz vorgestellt. Eine Benutzerschnittstelle für die Einfügung eines alten WordArt-Objekts steht in Word 2010 nicht zur Verfügung, die bearbeitenden Werkzeuge jedoch schon, wenn ein WordArt-Objekt markiert ist. Der Entwickler kann also durchaus dem Benutzer diese Funktionalität anbieten.

Statt WordArt als eine selbstständige Objektbibliothek zu führen, die allenfalls in das VBA-Projekt ausdrücklich eingebunden werden müsste, steht die Automatisierungsschnittstelle als Teil der Anwendung, über das gemeinsame Office-Objektmodell zur Verfügung.

Ein WordArt-Objekt wird durch die Methode `AddTextEffect` des `Shape`-Objekts erstellt:

```

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])
```

Alle Argumente außer Anchor sind erforderlich. Das erste Argument – PresetTextEffect – bestimmt die Grundform, die im WordArt-Katalog in der Benutzerschnittstelle ausgewählt wird (siehe Abbildung 12.6). Zudem werden Schriftart, -größe und -schnitt festgelegt sowie die Positionierung des Objekts relativ zum Verankerungsbereich. Wird kein Verankerungsbereich angegeben, erfolgt die Positionierung relativ zur linken, oberen Seitenecke.

PresetTextEffect erwartet einen MsoPreSetTextEffect-Konstantwert. Dieser ist von 0 bis 29 durchnummeriert, beschreibt den Effekt also nicht. Er entspricht jedoch den Abbildungen im Dialogfeld *WordArt-Katalog*, reihenweise von links oben nach rechts unten.

Abbildg. 12.6 Im ersten Schritt wird die Grundform des WordArt-Objekts aus dem *WordArt-Katalog* gewählt

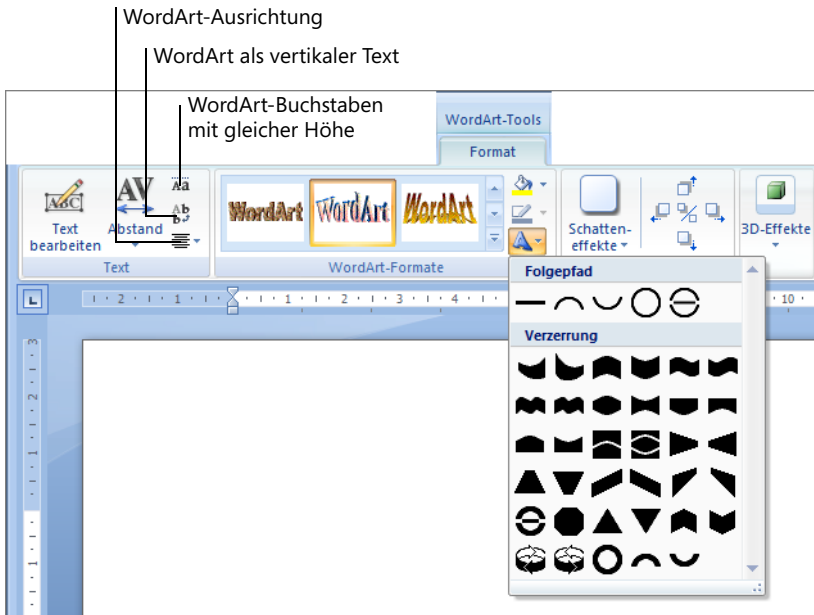


Diese Grundformen sind vordefinierte Zusammenstellungen der erweiterten Formatierungsmöglichkeiten, die die Registerkarte *WordArt-Tools/Format* (Abbildung 12.7) bereitstellt. Farben, Linien und Schattierungen werden mit den üblichen grafischen Werkzeugen geregelt.



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. 12.7 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 ein WordArt-Objekt ins Dokument eingefügt. 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 12.8).

Abbildg. 12.8 Wirkung der MsoPresetTextEffect- und MsoPresetTextEffectShape-Konstantwerte



Listing 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 vbCr
    Next

    'Dafür sorgen, dass genügend Absätze für alle MsoPresetTextEffectShape
    'Variationen vorhanden sind.
    rng.InsertAfter String(40 - doc.Paragraphs.Count, vbCr)
    For lEffectShape = 1 To 40
        Set rng = doc.Paragraphs(lEffectShape).Range
        Set shp = doc.Shapes.AddTextEffect(7, "Shape Effect" & CStr(lEffectShape), _
            "Arial", 20, msoFalse, msoFalse, 200, 0, rng)
        shp.TextEffect.PresetShape = lEffectShape
    Next
End Sub

```

Beachten Sie bitte im folgenden Listing 12.11, wie C# auf `Microsoft.Office.Core` verweist, um mit den WordArt-Eigenschaften zu arbeiten. Beachtenswert ist, wie der Ganzzahlwert einer Enumeration in den Enumerationswert konvertiert werden muss, bevor er mit `AddTextEffect` gebraucht werden kann.

Listing 12.11 (.NET): Die C#-Version für WordArt



```

private void AlleWordArtTextEffectsAuflisten_CS()
{
    Microsoft.Office.Core.MsoTriState offFalse =
        Microsoft.Office.Core.MsoTriState.msoFalse;
    wd.Application wdApp = this.GetWordApp();
    wd.Document doc = wdApp.ActiveDocument;
    wd.Range rng = doc.Content;
    rng.ParagraphFormat.LineSpacingRule = wd.WdLineSpacing.wdLineSpaceAtLeast;
    rng.ParagraphFormat.LineSpacing = 50f;
    for (int effect = 0; effect <= 29; effect++)
    {
        rng = doc.Paragraphs[effect + 1].Range;
        object objRange = rng;
        Microsoft.Office.Core.MsoPresetTextEffect objEffect =
            (Microsoft.Office.Core.MsoPresetTextEffect) effect;
        doc.Shapes.AddTextEffect(objEffect, "Effect" + effect.ToString(), "Arial", 20,
            offFalse, offFalse, 0, 0, ref objRange);
        rng.InsertAfter("\n");
    }
}

```

Listing 12.11 (.NET): Die C#-Version für WordArt (Fortsetzung)

```
//Dafür sorgen, dass genügend Absätze für alle MsoPresetTextEffectShape-
//Variationen vorhanden sind.
string newLine = "\n";
int i = 40 - doc.Paragraphs.Count;
char[] cNewLine = newLine.ToCharArray();
char c = cNewLine[0];
string s = new String(c, i);
rng.InsertAfter(s);
for (int effectShape = 1; effectShape <= 40; effectShape++)
{
    rng = doc.Paragraphs[effectShape].Range;
    object objRange = rng;
    Microsoft.Office.Core.MsoPresetTextEffect effect7 =
        Microsoft.Office.Core.MsoPresetTextEffect.msoTextEffect7;
    wd.Shape shp = doc.Shapes.AddTextEffect(effect7,
        "Shape Effect" + effectShape.ToString(),
        "Arial", 20, offFalse, offFalse, 200, 0, ref objRange);
    shp.TextEffect.PresetShape =
        (Microsoft.Office.Core.MsoPresetTextEffectShape) effectShape;
}
wdApp.Visible = true;
}
```

Die Benutzerschnittstelle für die Festlegung einiger Eigenschaften des TextEffectFormat-Objekts befindet sich im Dialogfeld *Text bearbeiten*: FontBold (Fett), FontItalic (Kursiv), FontName (Schriftart), FontSize (Schriftgröße), KernedPairs (Zeichenpaarkerning), NormalizedHeight (Höhe der Kleinbuchstaben). Die ersten und die letzten zwei erwarten einen msoTriState-Wert. Dies könnte Wahr (msoTrue) oder Falsch (msoFalse) sein. Über FontName wird die Schriftart, als Zeichenkette, festgelegt, während FontSize eine Single-Zahl akzeptiert, die die Schriftgröße in »Points« erwartet.

Die weiteren Eigenschaften des TextEffectFormat-Objekts entsprechen den Schaltflächen der Gruppe *WordArt-Formate*: Alignment (Ausrichtung, erwartet einen MsoTextEffectAlignment-Wert) und 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 einer Prozedur wie 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)
    End If
End Sub
```

Listing 12.12 Die Zeichen in einem WordArt-Objekt um 90° drehen (Fortsetzung)

```

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

```

CD-ROM Die Beispieldatei *Bsp12_04.docm* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap12*. Das entsprechende C#-Beispiel befindet sich in der Zip-Datei *Kap12_CS.zip* im Unterordner *C_Sharp*.

HINWEIS Ein weiteres Codebeispiel für die Automatisierung von WordArt finden Sie im Bonuskapitel VIII auf der CD-ROM zum Buch im Ordner *\Bonus\KapVIII*.

SmartArt

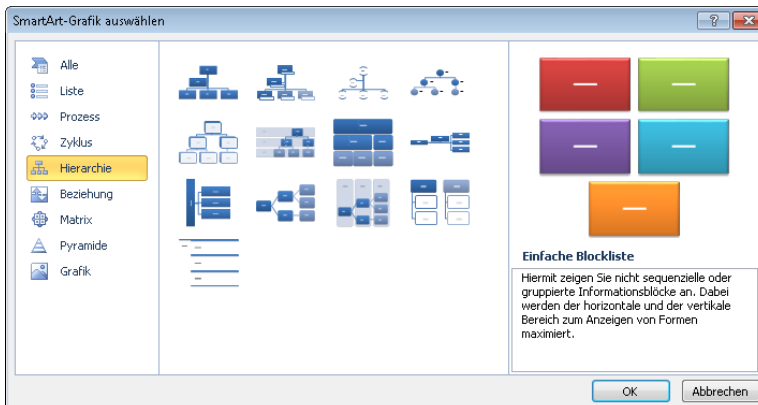
Als *SmartArt* wird der in Word 2007 eingeführte Ersatz für die bisherige Anwendung für Organisationsdiagramme bezeichnet. Diese neuen Grafiken bieten eine breite Palette von Diagrammen an, die modernen Ansprüchen entsprechen. Im Gegensatz zu den alten Werkzeugen bietet das neue ab Office 2010 eine Programmierschnittstelle für den Entwickler. Hier eine kurze Einführung in das zugehörige Objektmodell.

SmartArt-Grafiken auflisten



In der Benutzerschnittstelle wird eine SmartArt-Grafik über *Einfügen/Illustrationen/SmartArt* eingefügt. Das Dialogfeld *SmartArt-Grafik auswählen* bietet über 130 Grundformen von Diagrammen zur Auswahl an (Abbildung 12.9).

Abbildg. 12.9 Die Palette von SmartArt-Grafiken



Leider liefert der Makrorekorder überhaupt keine Informationen zu diesen Objekten – der Entwickler bleibt auf sich gestellt. Weil die angebotene Funktionalität erfreulich leistungsstark ist, ist das Objektmodell entsprechend komplex und somit nicht besonders intuitiv.

Das Listing 12.13 veranschaulicht, wie eine Liste der zur Verfügung stehenden SmartArt-Grafiktypen in einer Tabelle eines neuangelegten Dokuments erstellt werden kann (Abbildung 12.10). Daraus ist zu entnehmen, dass diese SmartArtLayouts-Objekte auf der Anwendungsebene (Application) angesiedelt sind. In Wirklichkeit gehören sie nicht der Word-Anwendung an, sondern sind Teil des gemeinsamen Office-Objektmodells. Wie bei WordArt oder Excel-Diagrammen stellen die einzelnen Office-Anwendungen sie über die eigene Programmierschnittstelle zur Verfügung, sodass kein Verweis gesetzt werden muss.

Abbildg. 12.10 Name, Beschreibung, Kategorie und ID-Eigenschaften einiger SmartArt-Grafiken

Anzahl Layouts: 134			
Einfache Blockliste	Hiermit zeigen Sie nicht sequenzielle oder gruppierte Informationsblöcke an. Dabei werden der horizontale und der vertikale Bereich zum Anzeigen von Formen maximiert.	list	urn:microsoft.com/office/officart/2005/8/layout/default
Alternierende Sechsecke	Verwendung: Darstellung einer Reihe miteinander verbundener Thesen. Text der Ebene 1 wird innerhalb der Sechsecke angezeigt. Text der Ebene 2 wird außerhalb der Formen angezeigt...	list	urn:microsoft.com/office/officart/2008/layout/Alternating Hexagons
Bildbeschriftungsliste	Hiermit zeigen Sie nicht sequenzielle oder gruppierte Informationsblöcke an. Die oberen Formen sind für Bilder vorgesehen, und Bilder werden gegenüber Text	list	urn:microsoft.com/office/officart/2005/8/layout/pList1

Listing 12.13 Zur Verfügung stehende SmartArt-Layouts auflisten

```

Sub SmartArtLayoutsAuflisten()
    Dim doc As Word.Document
    Dim rng As Word.Range
    Dim saLayout As Office.SmartArtLayout
    Dim l As Long
    Dim tbl As Word.Table

    Set doc = Documents.Add
    doc.PageSetup.Orientation = wdOrientLandscape
    Set rng = doc.Content
    l = 0
    For Each saLayout In Application.SmartArtLayouts
        rng.Text = saLayout.Name & vbTab & saLayout.Description & vbTab & _
            saLayout.Category & vbTab & saLayout.ID & vbCr
    Next saLayout
End Sub

```

Listing 12.13 Zur Verfügung stehende SmartArt-Layouts auflisten (Fortsetzung)

```

    rng.Collapse wdCollapseEnd
    l = l + 1
Next
rng.Text = "Anzahl Layouts: " & l & vbTab & vbTab
rng.Start = 1
Set tbl = rng.ConvertToTable(vbTab)
End Sub

```

Wie der Tabelle 12.1 zu entnehmen ist, hat ein `SmartArtLayout`-Objekt mehrere nützliche Eigenschaften. Die Eigenschaft `ID` identifiziert ein Mitglied der `SmartArtLayouts`-Auflistung eindeutig. Sie ist in der Form eines URN (»Uniform Resource Name« – siehe dazu http://de.wikipedia.org/wiki/Uniform_Resource_Name). Microsoft hat sich für einen URN statt eines traditionellen `Mso-Kons`-Konstantwertes entschieden. Dies untermauert die gegenwärtige Philosophie auf XML-Standarden zu bauen.

Tabelle 12.1 Wichtige Eigenschaften des `SmartArtLayout`-Objekts

Eigenschaft	Beschreibung
Name	Bezeichnung der SmartArt-Grafik in der Umgebungssprache der Installation
Description	Beschreibung des Zwecks der SmartArt-Grafik in der Umgebungssprache
Category	Kategorie, worin eine SmartArt-Grafik aufgelistet ist (englische Bezeichnung)
ID	Eindeutiger Identifikationswert in der Form eines URN; bleibt gleich für alle Sprachumgebungen

Um den URN (Indexwert) eines Bildes im Dialogfeld *SmartArt-Grafik auswählen* herauszufinden, stellen Sie den Mauszeiger über das Bild. Der Name erscheint neben dem Mauszeiger. Schlagen Sie den Namen in der erstellten Tabelle nach; der URN steht in der letzten Spalte.

CD-ROM Das Beispieldokument *Bsp_12_05_SA.docm* befindet sich auf der CD-ROM im Ordner *\Beispiele\Kap12*.

SmartArt-Grafiken einfügen

Wie bei allen grafischen Objekten ist ein SmartArt-Diagramm Mitglied der `InlineShapes`- oder der `Shapes`-Auflistung (mehr zu diesen Auflistungen lesen Sie im Kapitel 7). Ein neues SmartArt-Objekt wird über eine `Add`-Methode der `InlineShapes`- oder `Shapes`-Auflistungen eingefügt. In beiden Fällen heißt die Methode `AddSmartArt`.

Die Syntax für ein `InlineShape`-Objekt fordert ein Objekt des Typs `SmartArtLayout`, um die gewünschte SmartArt-Grafik festzulegen. Das Argument `Range` ist fakultativ und legt fest, wo das Objekt einzufügen ist.

```
AddSmartArt(Layout, [Range])
```

Auch die Methode des Shape-Objekts verlangt ein Objekt des Typs SmartArtLayout als erstes Argument. Die restlichen Argumente sind fakultativ und die gleichen, wie bei anderen Add-Methoden des Shape-Objekts, die dessen Position festlegen.

```
AddSmartArt(Layout, [Left], [Top], [Width], [Height], [Anchor])
```

Das Listing 12.14 veranschaulicht das Einfügen eines relativ einfachen Organisationsdiagramms in den zweiten Absatz eines neuen Dokuments (Abbildung 12.11). Wie daraus zu entnehmen ist, kann der ID-Wert (der URN) als Indexwert der SmartArtLayout-Auflistung dienen.

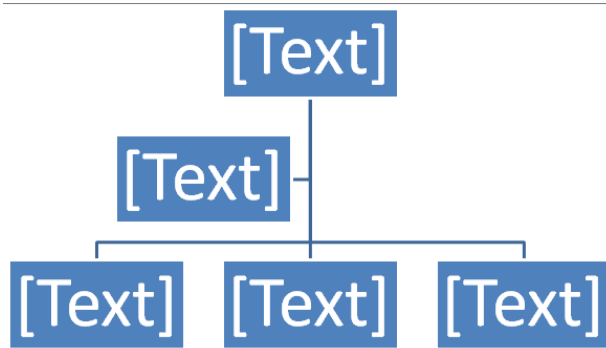
WICHTIG Im Gegensatz zu den meisten anderen Indexwerten in VBA-Objektmodellen *muss* auf die Großschreibung des URN eines SmartArtLayout-Objekts geachtet werden! Weil diese Angabe als XML behandelt wird, muss die Großschreibung genau übereinstimmen. Leider steht hier IntelliSense nicht zur Verfügung, wie dies bei traditionellen Konstanten der Fall ist. Wir haben den Wert aus der mit Listing 12.13 erstellten Tabelle kopiert und in den Code eingefügt.

Listing 12.14 Eine SmartArt-Grafik einfügen

```
Sub SmartArtGrafikEinfügen()
    Dim doc As Word.Document
    Dim i As Long
    Dim rng As Word.Range
    Dim ils As Word.InlineShape
    Dim sal As Office.SmartArtLayout

    Set doc = Documents.Add
    For i = 1 To doc.Paragraphs.Count
        If i < 2 Then
            doc.Content.InsertAfter vbCr
        End If
    Next i
    Set rng = doc.Paragraphs(2).Range
    Set sal = Application.SmartArtLayouts(
        "urn:microsoft.com/office/officeart/2005/8/layout/orgChart1")
    Set ils = doc.InlineShapes.AddSmartArt(sal, rng)
End Sub
```

Abbildg. 12.11 Ein SmartArt-Organigramm in seinem Ursprungszustand



CD-ROM Das Beispieldokument *Bsp_12_05_SA.docm* befindet sich auf der CD-ROM im Ordner *\Beispiele\Kap12*.

Auf vorhandene SmartArt-Grafik prüfen

Um festzustellen, ob eine bestimmte Grafik vom Typ SmartArt ist, wird das Resultat der Type-Eigenschaft des `InlineShape`- bzw. `Shape`-Objekts ausgewertet. Beträgt er `wdInlineShapeSmartArt` (entspricht der Ganzzahl 15) bzw. `msoSmartArt` (24), liegt eine SmartArt-Grafik vor.

TIPP Obwohl Word 2007 keinen Zugriff auf ein Objektmodell für SmartArt besitzt, steht ihm diese Eigenschaft zur Verfügung. Somit kann der Entwickler zumindest feststellen, ob er es mit einer SmartArt-Grafik zu tun hat.

Die SmartArt-API ergänzt das Word-Objektmodell um eine weitere Eigenschaft. Mit dieser lässt sich ebenfalls feststellen, ob es sich bei der Grafik um ein SmartArt handelt: `HasSmartArt`. Diese Eigenschaft gibt »Wahr« zurück, falls das `InlineShape`- oder `Shape`-Objekt eine SmartArt-Grafik ist.

Anpassung einer SmartArt-Grafik

Wir gelangen durch das `InlineShape`- bzw. `Shape`-Objekt an das dahinterliegende SmartArt-Objekt:

```
Dim sa as Office.SmartArt
Set sa = ils.SmartArt
```

Dieses Objekt stellt einige Eigenschaften und eine Methode zur Verfügung. Die nützlichsten sind Tabelle 12.2 zu entnehmen.

Tabelle 12.2 Die wichtigste Eigenschaften und Methoden des *SmartArt*-Objekts

Eigenschaft	Beschreibung
AllNodes	Ruft ein <code>SmartArtNodes</code> -Objekt mit allen Knoten im SmartArt-Diagramm ab. Schreibgeschützt.
Nodes	Ruft die untergeordneten Elemente eines angegebenen Knotenpunkts für das SmartArt-Diagramm ab. Schreibgeschützt. Um dem Diagramm Knotenpunkte hinzuzufügen, benutzen Sie die Methode <code>AddNode</code> des <code>SmartArtNode</code> -Objekts (Abschnitt »Struktur eines Diagramms« ab Seite 642).
Layout	Ruft das SmartArt-Layout ab, das auf die SmartArt-Grafik angewendet wird, oder legt es fest. Lese-/Schreibzugriff.
Color	Ruft die SmartArt-Farbformatvorlage ab, die auf die SmartArt-Grafik angewendet wird, oder legt sie fest. Lese-/Schreibzugriff. Diese basieren auf dem angehängten Design des Dokuments.
QuickStyle	Ruft die SmartArt-Schnellformatvorlage ab, die auf die SmartArt-Grafik angewendet wird, oder legt sie fest. Lese-/Schreibzugriff. Dies sind Paletten der Formkonturen- und -Effekte des grafischen Objektmodells.

Tabelle 12.2 Die wichtigsten Eigenschaften und Methoden des *SmartArt*-Objekts (Fortsetzung)

Eigenschaft	Beschreibung
Reverse	Ruft den Status des SmartArt-Diagramms im Hinblick auf LNR (links-nach-rechts) oder RNL (rechts-nach-links) ab oder legt ihn fest, wenn das Diagramm Umkehrung unterstützt. Lese-/Schreibzugriff.
Reset	Setzt die SmartArt-Grafik auf den ursprünglichen Zustand zurück

Durch die Nodes-Auflistungen sowie *SmartArtNode*-Objekte werden die einzelnen Elemente einer SmartArt-Grafik angesprochen sowie neue hinzugefügt. Die Eigenschaften *Color*, *Layout* und *Quickstyle* beeinflussen die Formatierung des gesamten Diagramms.

Farbe einer SmartArt-Grafik ändern

Wenn wir das Beispiel des Listing 12.14 weiterziehen, und nach dem Einfügen die SmartArt-Grafikfarbe ändern, sehen die entsprechenden Codezeilen so aus:

```
'Das eingefügte Diagramm formatieren
Dim sa As Office.SmartArt
Dim sac As Office.SmartArtColor

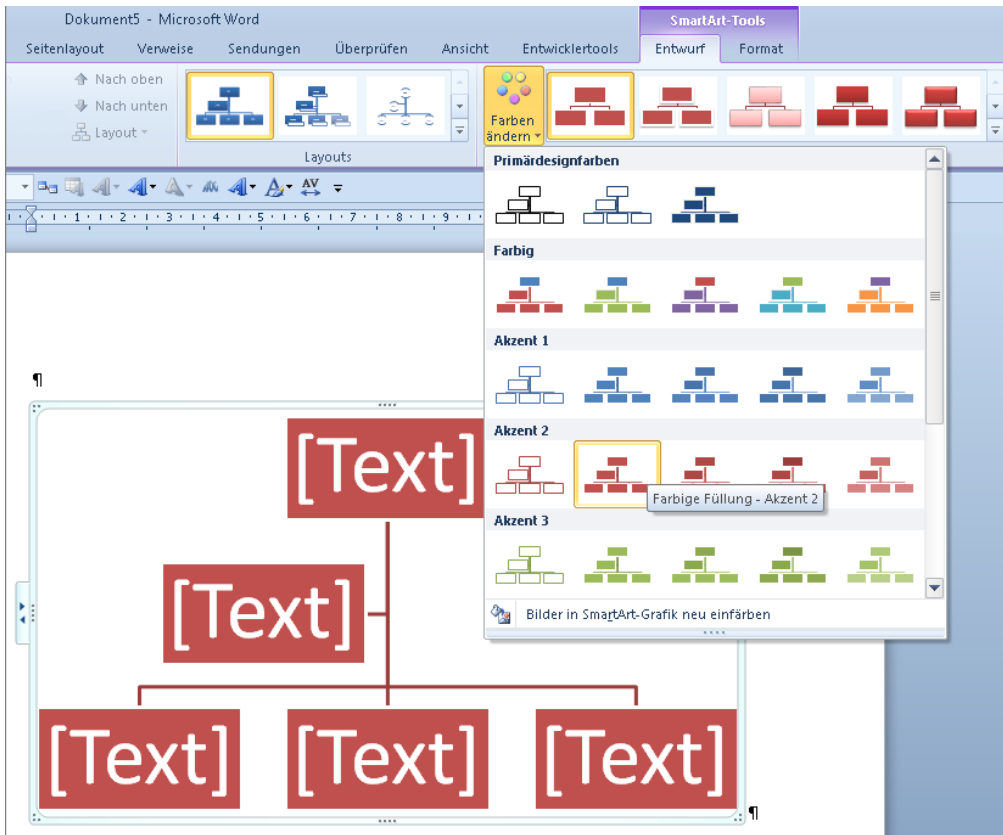
Set sa = ils.SmartArt
Set sac = Application.SmartArtColors(
    "urn:microsoft.com/office/officeart/2005/8/colors/accent2_2")
sa.Color = sac
```

Es ist ersichtlich, dass die Auflistung *SmartArtColors* ähnlich wie *SmartArtLayouts* gehandhabt wird. Das Objekt *SmartArtColor* hat die gleichen wichtigen Eigenschaften. Auch diese ist Mitglied des *Application*-Objekts und als Indexwert fungiert ein URN. Ähnlicher Code wie in Listing 12.13 zeigt auf, dass gegenwärtig 38 Farbpaletten zur Verfügung stehen. Auch hier entspricht der Name den Informationen einer Liste in der Word-Anwendung, in diesem Fall *SmartArt-Tools/Entwurf/SmartArt-Formatvorlagen/Farben ändern* (Abbildung 12.12).

HINWEIS

Die zur Verfügung stehenden Farben hängen vom aktuellen Design ab (*Seitenlayout/Designs*), welches mit dem Dokument verbunden ist. Wird ein anderes Design ausgewählt, ändern sich die Farben der *SmartArtColors*-Mitglieder entsprechend. Mehr über Designs erfahren Sie aus dem Buch » Microsoft Office 2007 – Das Profibuch: Fortgeschrittene Lösungen für Word, Excel und PowerPoint, die die neuen Möglichkeiten der Version 2007 nutzen«, erschienen bei Microsoft Press (ISBN-13: 978-3-86645-634-1).

Abbildg. 12.12 Die der SmartArtColors-Auflistung zugrunde liegende Farbpalette



Ähnlich verhält es sich bei den vierzehn zur Verfügung stehenden SmartArt-Formatvorlagen, (SmartArtQuickStyles-Auflistung) deren Steuerelement im Menüband in Abbildung 12.12 rechts, neben der Farbpalette zu sehen ist. Diese ändern die Formkonturen und -effekte der SmartArt-Grafik. Um den Elementen des Beispieldiagramms einen 3D-Look zu geben:

```
Dim saq As Office.SmartArtQuickStyle
Set saq = Application.SmartArtQuickStyles(
    "urn:microsoft.com/office/officeart/2005/8/quickstyle/3d1")
saq.QuickStyle = saq
```

CD-ROM Das Beispieldokument *Bsp_12_05_SA.docm* befindet sich auf der CD-ROM im Ordner *\Beispiele\Kap12*. Darin finden Sie die Prozeduren *SmartArtColorsAuflisten* und *SmartArtQuickStylesAuflisten*.

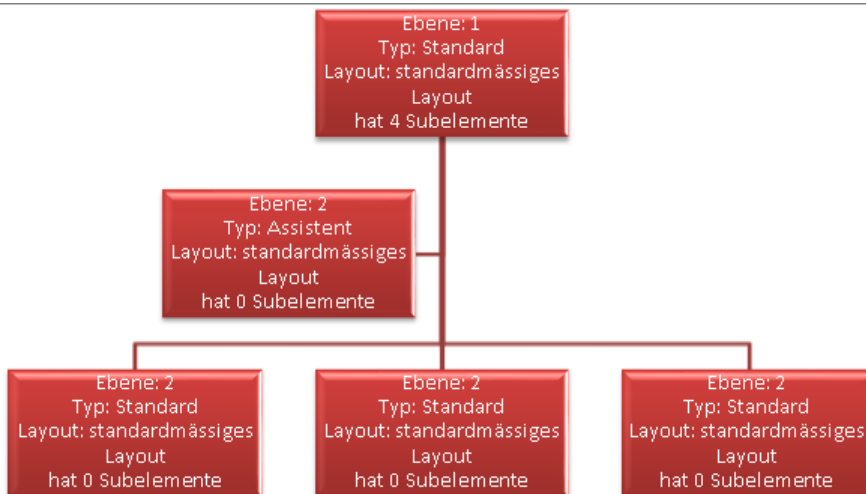
Elemente eines Diagramms durchlaufen und beschriften

Ein SmartArt-Diagramm ist hierarchisch aufgebaut: Jedes Element (Knotenpunkt/SmartArtNode) ist einer Ebene (Level) zugewiesen. Das Element ist entweder ein standardmäßiger Knotenpunkt oder ein Assistent, welches durch die Eigenschaft `Type` zurückgegeben wird. Die Anordnung der Unterelemente wird durch die Eigenschaft `OrgChartLayout` bestimmt. Auf den Text greifen Sie über die Eigenschaft `TextFrame2.TextRange.Text` zu. Diese Eigenschaft veranschaulicht das Codefragment in Listing 12.15. Das Resultat ist in Abbildung 12.13 ersichtlich.

Listing 12.15 Die Elemente eines Diagramms durchschleifen und mit den Eigenschaftswerten beschriften

```
'Die Elemente des Diagramms beschriften
Dim saNodes As Office.SmartArtNodes
Dim saNode As Office.SmartArtNode
Set saNodes = sa.AllNodes
For Each saNode In saNodes
    saNode.TextFrame2.TextRange.Text =
        "Ebene: " & saNode.Level & Chr(11) & "Typ: " &
        & SmartArtTypeAlsText(saNode.Type) & Chr(11) &
        & "Layout: " & OrgLayoutTypeAlsText(saNode.OrgChartLayout)
        & Chr(11) & "hat " & CStr(AnzahlUnterelemente(saNode)) & " Unterelemente"
Next
```

Abbildg. 12.13 Mit ihren Eigenschaften beschriftete Elemente eines Diagramms



Die wichtigsten Eigenschaften und Methoden des SmartArtNode-Objekts sind in der Tabelle 12.3 aufgelistet.

WICHTIG

Achten Sie auf den Unterschied der Eigenschaften `AllNodes` und `Nodes`. Die erste bezieht sich auf alle Knotenpunkte der SmartArt-Grafik. `Nodes` hingegen spricht einzeln die von einem Knotenpunkt direkt abhängigen Elemente an.

Tabelle 12.3 Die Eigenschaften und Methoden des SmartArtNode-Objekts

Eigenschaft oder Methode	Beschreibung
Level	Ruft die Ebene des Knotens in der Hierarchie ab. Schreibgeschützt.
Type	Ruft den Typ des SmartArt-Knotens ab und gibt einen MsoSmartArtNodeType -Konstantwert zurück. Diese haben entweder den Wert msoSmartArtNodeTypeDefault oder msoSmartArtNodeTypeAssistant . Schreibgeschützt.
TextFrame2	Gibt den Textrahmen der AutoForm zurück, der dem SmartArtNode -Objekt zugeordnet ist. Schreibgeschützt. Um darin enthaltenen Text zu lesen oder zu ändern, muss die TextRange.Text -Eigenschaft angesprochen werden.
OrgChartLayout	Ruft das MsoOrgChartLayoutType -Objekt auf oder legt es fest, das diesem Knoten (falls vorhanden) zugeordnet ist. Lese-/Schreibzugriff.
Shape	Gibt den Formbereich (Office.ShapeRange) zurück, der diesem SmartArtNode -Objekt zugeordnet ist. Schreibgeschützt. Durch diese Eigenschaft können viele Eigenschaften des Shape -Objekts gelesen werden, die meisten sind jedoch für SmartArt -Objekte schreibgeschützt. Es gibt im Objektmodell kein Gegenstück zur Funktionalität in der Benutzerschnittstelle, die es dem Benutzer erlaubt, einzelne Elemente auszuwählen und direkt zu formatieren.
Larger	Vergrößert den SmartArt-Knoten. Imitiert das Verhalten der Schaltfläche <i>Größer</i> auf der Registerkarte <i>Format</i> für SmartArt, die sich auf der Menüband-Benutzeroberfläche von Microsoft Office Fluent befindet.
Smaller	Verkleinert die SmartArt. Imitiert das Verhalten der Schaltfläche <i>Kleiner</i> auf der Registerkarte <i>Format</i> für SmartArt, die sich auf der Menüband-Benutzeroberfläche von Microsoft Office Fluent befindet.
ParentNode	Ruft das übergeordnete SmartArtNode -Objekt dieses SmartArtNode -Objekts ab. Schreibgeschützt.
AddNode	Fügt dem Datenmodell ein neues SmartArtNode -Objekt hinzu, dessen Position durch SmartArtNodePosition und dessen Typ durch SmartArtNodeType festgelegt wird
Delete	Entfernt den aktuellen SmartArt-Knoten
Demote	Stuft den aktuellen Knoten im Datenmodell eine einzelne Ebene tiefer
Promote	Stuft den aktuellen Knoten (und alle untergeordneten Elemente) im Datenmodell eine einzelne Ebene höher
ReorderDown	Vertauscht einen Knoten mit dem nächsten Knoten in der Aufzählung. Mit dieser Methode werden alle unter- und übergeordneten Elemente des Knotens neu angeordnet.
ReorderUp	Vertauscht einen Knoten mit dem vorherigen Knoten in der Aufzählung. Mit dieser Methode werden alle unter- und übergeordneten Elemente des Knotens neu angeordnet.

CD-ROM

Das Beispieldokument *Bsp_12_05_SA.docm* befindet sich auf der CD-ROM im Ordner *\Beispiele\Kap12*.

Struktur eines Diagramms

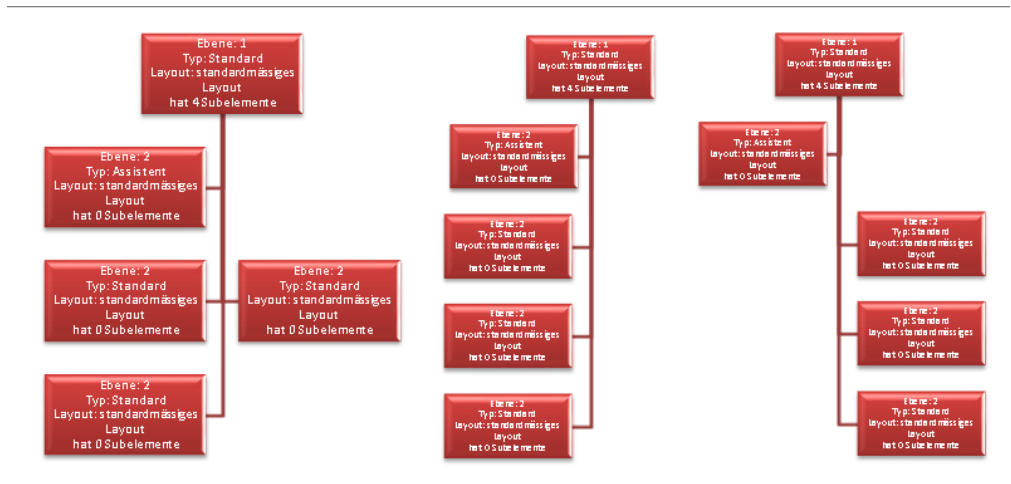
Es fällt auf, dass alle Elemente der Abbildung 12.13 das gleiche Layout haben, das Standardlayout. Das ist, weil das Diagramm erst eingefügt und die Struktur nicht geändert wurde. Es stehen weitere Layoutoptionen zur Verfügung, die durch `MsoOrgChartLayout`-Konstantenwerte festgelegt werden (Tabelle 12.4).

Tabelle 12.4 `MsoOrgChartLayout`-Konstantenwerte

<code>MsoOrgChartLayout</code> -Konstantenwert	Wert	Beschreibung
<code>msoOrgChartLayoutStandard</code>	1	Entspricht dem standardmäßigen Layout (Standard)
<code>msoOrgChartLayoutBothHanging</code>	2	Die Knotenpunkte werden beidseitig verteilt
<code>msoOrgChartLayoutLeftHanging</code>	3	Die Knotenpunkte sind auf der linken Seite hängend ausgerichtet
<code>msoOrgChartLayoutRightHanging</code>	4	Die Knotenpunkte sind auf der rechten Seite hängend ausgerichtet
<code>msoOrgChartLayoutDefault</code>	5	Das beim Einfügen standardmäßige Format. Dadurch kann festgestellt werden, ob das Layout jemals geändert wurde.
<code>msoOrgChartLayoutMixed</code>	-2	Gemischtes Layout

Sie können in Abbildung 12.14 die Wirkung der Konstantenwerte `msoOrgChartLayoutBothHanging`, `msoOrgChartLayoutLeftHanging` bzw. `msoOrgChartLayoutRightHanging` entnehmen. Der Konstantenwert `msoOrgChartLayoutDefault` ist schreibgeschützt und verursacht bei der Zuweisung einen Fehler; falls Sie zum ursprünglichen Layout zurückkommen möchten, weisen Sie dem Element den Wert `msoOrgChartLayoutStandard` zu.

Abbildg. 12.14 Die Wirkung der `MsoOrgChartLayout`-Konstantenwerte



Sie können einem Diagramm Elemente hinzufügen, löschen und in andere Ebenen versetzen, sowie deren Reihenfolge innerhalb der Ebene ändern. Listing 12.16 veranschaulicht die Bearbeitung einer

SmartArt-Grafik nach deren Einfügung. Alle »Assistent«-Elemente werden aus der AllNodes-Auflistung entfernt. Die erste Ebene wird beschriftet. Zudem stehen zwei Codezeilen im Beispiel, um zu veranschaulichen, wie das Shape-Objekt des SmartArt-Elements angesprochen wird.

Danach werden sukzessiv die Knotenpunkte der zweiten Ebene durchgearbeitet, um das Resultat in Abbildung 12.15 zu erzielen. Jeder muss beschriftet und als »rechts hängend« formatiert werden. Zudem wird jedes neue Element in einer dritten Ebene zugefügt. Auch diese Elemente müssen beschriftet werden.

Listing 12.16 Knotenpunkte einer SmartArt-Grafik löschen und hinzufügen

```
Sub SmartArtGrafikEinfügenFormatierenAnpassen()
    Dim doc As Word.Document
    Dim i As Long
    Dim rng As Word.Range
    Dim ils As Word.InlineShape
    Dim sal As Office.SmartArtLayout

    Set doc = Documents.Add
    For i = 1 To doc.Paragraphs.Count
        If i < 2 Then
            doc.Content.InsertAfter vbCr
        End If
    Next i
    Set rng = doc.Paragraphs(2).Range
    Set sal = Application.SmartArtLayouts(
        "urn:microsoft.com/office/officeart/2005/8/layout/orgChart1")
    Set ils = doc.InlineShapes.AddSmartArt(sal, rng)

    'Das eingefügte Diagramm formatieren..
    Dim sa As Office.SmartArt
    Dim sac As Office.SmartArtColor

    Set sa = ils.SmartArt
    Set sac = Application.SmartArtColors(
        "urn:microsoft.com/office/officeart/2005/8/colors/accent4_5")
    sa.Color = sac

    'und die Formkonturen ändern
    Dim saq As Office.SmartArtQuickStyle
    Set saq = Application.SmartArtQuickStyles(
        "urn:microsoft.com/office/officeart/2005/8/quickstyle/3d5")
    sa.QuickStyle = saq

    'Die Elemente des Diagramms bearbeiten
    Dim saNodes As Office.SmartArtNodes
    Dim saNode As Office.SmartArtNode
    Set saNodes = sa.AllNodes

    'Alle Knotenpunkte des Typs Assistent löschen
    For Each saNode In saNodes
        If saNode.Type = msoSmartArtNodeTypeAssistant Then
            saNode.Delete
        End If
    Next
End Sub
```

Listing 12.16 Knotenpunkte einer SmartArt-Grafik löschen und hinzufügen (Fortsetzung)

```

'Die erste Ebene bearbeiten
Dim saNode1 As Office.SmartArtNode
Dim shp As Office.Shape
Set saNode1 = sa.Nodes(1)
saNode1.TextFrame2.TextRange.Text = "Vorsitzender"
'Die Eigenschaften des Objekts als Word-Grafik abfragen
Set shp = saNodeOperationen.Shapes(1)
shp.Title = "Vorsitzender" 'Schreibbare Eigenschaft
'Lesbare Eigenschaften
Debug.Print "Hohe: " & shp.Height, "Breite: " & shp.Width

'Die Anzahl Knotenpunkte der 2. Ebene auf drei beschränken.
Dim saNodesLev2 As Office.SmartArtNodes
Dim iLev2 As Long
Set saNodesLev2 = saNode1.Nodes
For iLev2 = 4 To saNodesLev2.Count
    saNodesLev2(iLev2).Delete
Next

'Die zweite Ebene bearbeiten
'''Operativer Bereich
Dim saNodeOperationen As Office.SmartArtNode
Dim aNewNodes() As Variant
ReDim aNewNodes(2)
aNewNodes(0) = "Int'l Operationen"
aNewNodes(1) = "Marketing"
aNewNodes(2) = "Qualitätskontr."
Set saNodeOperationen = saNodesLev2(1)
NodeLev2Formatieren saNodeOperationen, "Operativer Bereich", _
    msoOrgChartLayoutRightHanging, aNewNodes()

'''Finanzieller Bereich
Dim saNodeFinanzen As Office.SmartArtNode
ReDim aNewNodes(1)
aNewNodes(0) = "Finz. Anlagen"
aNewNodes(1) = "Liegenschaften"
Set saNodeFinanzen = saNodesLev2(2)
NodeLev2Formatieren saNodeFinanzen, "Finanzbereich", _
    msoOrgChartLayoutRightHanging, aNewNodes()

'''Bereich Personal
Dim saNodePersonal As Office.SmartArtNode
ReDim aNewNodes(0)
aNewNodes(0) = "Rekrutierung"
Set saNodePersonal = saNodesLev2(3)
NodeLev2Formatieren saNodePersonal, "Personalbereich", _
    msoOrgChartLayoutRightHanging, aNewNodes()

End Sub

Sub NodeLev2Formatieren(saNode As Office.SmartArtNode, _
    Beschriftung As String, Layout As MsoOrgChartLayoutType, _
    Optional NeueKP As Variant)

    Dim i As Long

```


Listing 12.16 Knotenpunkte einer SmartArt-Grafik löschen und hinzufügen (Fortsetzung)

```

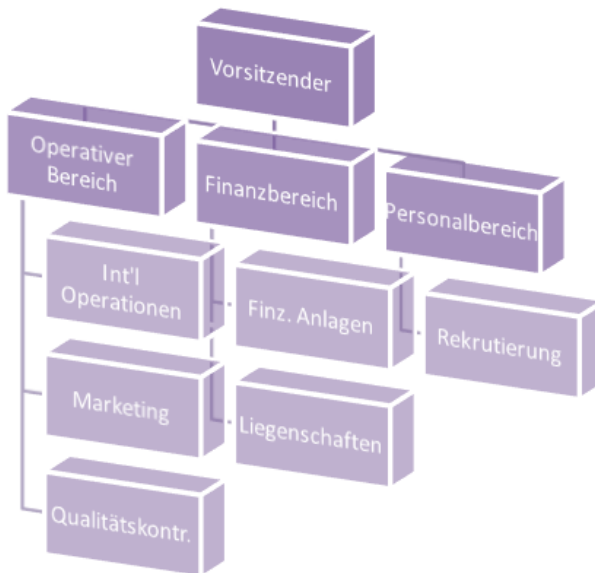
Dim subNodeBeschriftung As String
With saNode
    .TextFrame2.TextRange.Text = Beschriftung
    .OrgChartLayout = Layout
    If Not IsMissing(NeueKP) Then
        For i = LBound(NeueKP) To UBound(NeueKP)
            subNodeBeschriftung = NeueKP(i)
            KnotenpunktHinzu saNode, subNodeBeschriftung
        Next
    End If
End With
End Sub

Sub KnotenpunktHinzu(saNode As Office.SmartArtNode, _
    Beschriftung As String)

    Dim newNode As Office.SmartArtNode
    Set newNode = saNode.AddNode(msoSmartArtNodeBelow, _
        msoSmartArtNodeTypeDefault)
    NodeLev2Formatieren newNode, Beschriftung, msoOrgChartLayoutStandard
End Sub

```

Abbildg. 12.15 Knotenpunkte der SmartArt-Grafik löschen und hinzufügen



Was ist, wenn eine SmartArt-Grafik vorliegt, die bearbeitet werden muss? Das Element »Int'l Operationen« soll beispielsweise am Ende der Auflistung erscheinen. In diesem Fall wird das Diagramm über den `InlineShape`-Behälter angesprochen. Im Beispiel wird mit einer der `Reorder`-Methoden gearbeitet, um den Knotenpunkt zu verschieben, wie in Listing 12.17 ersichtlich.

Listing 12.17 Knotenpunkte einer SmartArt-Grafik neu anordnen

```
Sub SmartArtKnotenPunktVerschieben()  
    Dim doc As Word.Document  
    Dim sa As Office.SmartArt  
    Dim saNode As Office.SmartArtNode  
    Dim Beschriftung As String  
  
    Beschriftung = "Marketing"  
    Set doc = ActiveDocument  
    Set sa = doc.InlineShapes(1).SmartArt  
    For Each saNode In sa.AllNodes  
        If saNode.TextFrame2.TextRange.Text = Beschriftung Then  
            On Error Resume Next  
            saNode.ReorderDown  
            saNode.ReorderDown  
            Exit For  
        End If  
    Next  
End Sub
```

ACHTUNG

Es ist nicht möglich, die Position eines Knotenpunkts innerhalb einer Ebene direkt abzufragen. Ist es nicht möglich, einen Knotenpunkt nach unten oder oben zu verschieben, wird ein Laufzeitfehler ausgelöst: »Der Vorgang wird vom aktuellen Objekt nicht unterstützt«. On Error Resume Next unterbindet die Anzeige der Fehlermeldung.

Mit dieser Einführung in den Umgang mit dem SmartArt-Objektmodell haben Sie einen Anfangspunkt, aussagekräftige Diagramme programmtechnisch zu erstellen. Obwohl die Vorgehensweise etwas von herkömmlichen VBA-Objektmodellen abweicht, ist es durchaus logisch und entspricht der Zukunftsausrichtung der objektorientierten Programmiersprachen von Microsoft.

CD-ROM

Das Beispieldokument *Bsp_12_05_SA.docm* befindet sich auf der CD-ROM im Ordner *\Beispiele\Kap12*.

Zusammenfassung

In diesem Kapitel wurde auf OLE-Objekte zugegriffen, die in einem Dokument eingebettet sind. Diese Objekte wurden aus Word heraus ferngesteuert.

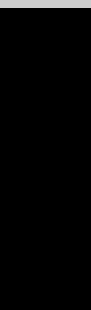
- In diesem Kapitel wurde zunächst gezeigt, wie eingebettete Excel-Tabellen (Seite 609) und Office-Diagramme (Seite 616) bearbeitet werden können
- Im Weiteren wurde erläutert, wie der Zugriff auf MS Graph-Diagramme (Seite 623) und Word-Art-Objekte (Seite 628) erfolgen kann
- Das neue Objektmodell für SmartArt wird am Schluss kurz vorgestellt (Seite 633 ff.)

Teil D

Optimierung der Benutzerschnittstelle

In diesem Teil:

Kapitel 13	Anwendungsoptionen	649
Kapitel 14	Speicherort der Anpassungen	675
Kapitel 15	Mit Dialogfeldern arbeiten	683
Kapitel 16	Die Office Fluent UI	727
Kapitel 17	Tastaturbelegungen	779
Kapitel 18	Word-Aufgabenbereiche	793
Kapitel 19	Interne Word-Befehle übersteuern	811
Kapitel 20	Zugriff auf den Visual Basic-Editor (VBE)	819



Kapitel 13

Anwendungsoptionen

In diesem Kapitel:

Dokumentvorlage <i>Normal.dotm</i> konfigurieren	650
Benutzereinstellungen abspeichern	660
Dokumenteinstellungen abspeichern	669
Zusammenfassung	674

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.dotm* – und deshalb ist es wichtig, dass die *Normal.dotm* als Basis entsprechend angepasst wird. Mehr über die Rolle der *Normal.dotm* erfahren Sie in Kapitel 1.

Dokumentvorlage *Normal.dotm* konfigurieren



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, sodass seine tägliche Arbeit erleichtert wird. Dazu gehören Punkte wie automatische Trennhilfe, Rechtschreibprüfung usw. Übergeordnet müssen die Vorgaben zur Darstellung von Dokumenten, dem sogenannten 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.dotm* 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 sogenannte »UrNormal.dotm« 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.dotm« erstellen

Damit die neue Standardvorlage die Bezeichnung »UrNormal.dotm« 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« ab Seite 652 erläutert
- Die Einstellungen zu den *Dokumenteigenschaften* werden im Abschnitt »Dokumenteigenschaften eintragen« ab Seite 652 behandelt

- Mögliche Einstellungen zur *Dokumentansicht* und *Zoomfaktor* werden im Abschnitt »Dokumentansicht und Zoomfaktor bestimmen« ab Seite 653 beschrieben
- Möglichkeiten zur Optimierung der *Formatvorlagen* werden im Abschnitt »Formatvorlagen definieren« ab Seite 653 dargestellt
- Die Einstellungen zur integrierten *Silbentrennung* werden im Abschnitt »Silbentrennung konfigurieren« ab Seite 654 festgelegt
- Die Einstellungen für das *Zeichnungsraster* sowie der Standard für die Autoformen werden im Abschnitt »Zeichnungselemente optimieren« ab Seite 655 den eigenen Bedürfnissen entsprechend angepasst
- Als letzter Arbeitsschritt werden die dokumentspezifischen *Optionen* im Abschnitt »Optionen bearbeiten« ab Seite 657 erläutert

WICHTIG Gemeinsam genutzte Makros, egal ob aus firmeninterner Quelle oder von Anbietern von Programmiererweiterungen, werden auf keinen Fall in der *Normal.dotm* 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 gezeigt werden, wie Sie für Ihre eigene Umgebung eine »UrNormal.dotm« 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 für das aktuelle Thema von Bedeutung sind.

Originalvorlage *Normal.dotm* durch Word erstellen lassen

Bevor Sie mit dem Konfigurieren und Anpassen der *Normal.dotm* beginnen können, muss sichergestellt werden, dass auf der Arbeitsstation eine originale *Normal.dotm* 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.dotm*.
3. Löschen Sie alle gefundenen Dateien oder benennen Sie diese um.

HINWEIS Beachten Sie, dass beim Löschen der *Normal.dotm* Ihre persönlichen Einstellungen für *Formatvorlagen*, *AutoTexte*, *Symbolleisten* sowie *Makroprojektelemente* gleichzeitig verloren gehen, soweit sie in dieser Datei gespeichert sind.

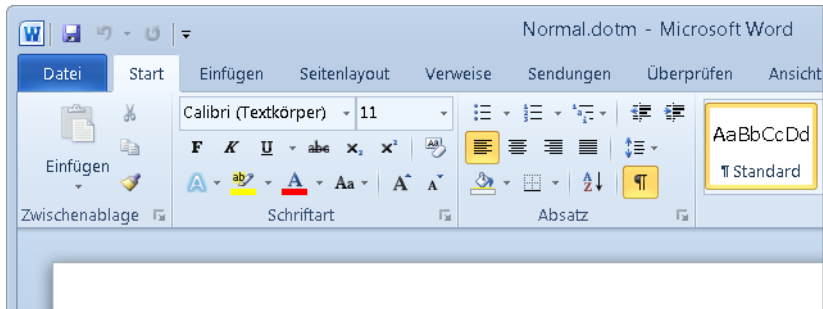
Wird die Datei nur umbenannt und in einem späteren Schritt vom System entfernt, können Sie diese Elemente nachträglich in die neue persönliche *Normal.dotm* übertragen. Stellen Sie sicher, dass die *Normal.dotm* und nicht die »UrNormal.dotm« verwendet wird.

Um einzelne oder gleich alle Elemente von der alten Datei in die neue *Normal.dotm* zu übertragen, wählen Sie im Menüband *Entwicklertools/Vorlagen/Dokumentvorlage/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.dotm* 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.dotm* bearbeiten. Beachten Sie dazu, dass der Dateiname wie in Abbildung 13.1 in der Titelleiste erscheint.

Abbildg. 13.1 Bearbeiten der geöffneten *Normal.dotm*



WICHTIG

Es ist enorm wichtig, dass die *Normal.dotm*, wie vorhin gezeigt, erstellt wurde. Das Abspeichern einer beliebigen Datei als Dokumentvorlage mit der Bezeichnung *Normal.dotm* macht aus dieser Datei noch keine *Normal.dotm*, wie wir sie benötigen.

Seitenlayout festlegen

In einem ersten Schritt wird das Seitenlayout der Standardvorlage festgelegt. Rufen Sie dazu den Befehl *Seitenlayout/Seitenränder/Benutzerdefinierte Seitenränder* 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 *Papier* wird das gewünschte *Papierformat* eingetragen
- Auf der Registerkarte *Layout* wird für die Kopf- und Fußzeilen der *Abstand vom Seitenrand* eingetragen

Dokumenteigenschaften eintragen

In einem zweiten Schritt werden die Eigenschaften des Dokuments festgelegt. Rufen Sie dazu den Befehl *Datei/Informationen/Eigenschaften/Erweiterte Eigenschaften* auf. Da wir uns mit der »UrNormal.dotm« beschäftigen, empfehlen wir bei allen Feldern die vorgeschlagenen Werte zu entfernen, damit später keine falschen Werte in die Dokumente übernommen werden.

ACHTUNG

Im Gegensatz zu allen anderen Dokumentvorlagen verhält sich die *Normal.dotm* 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 im Menüband den Befehl *Ansicht/Dokumentansichten/Seitenlayout*, um die vorgeschlagene Ansicht zu aktivieren
- Rufen Sie zusätzlich den Befehl *Ansicht/Zoom/Seitenbreite* auf, um einen dynamischen Zoomfaktor zu setzen

Formatvorlagen definieren

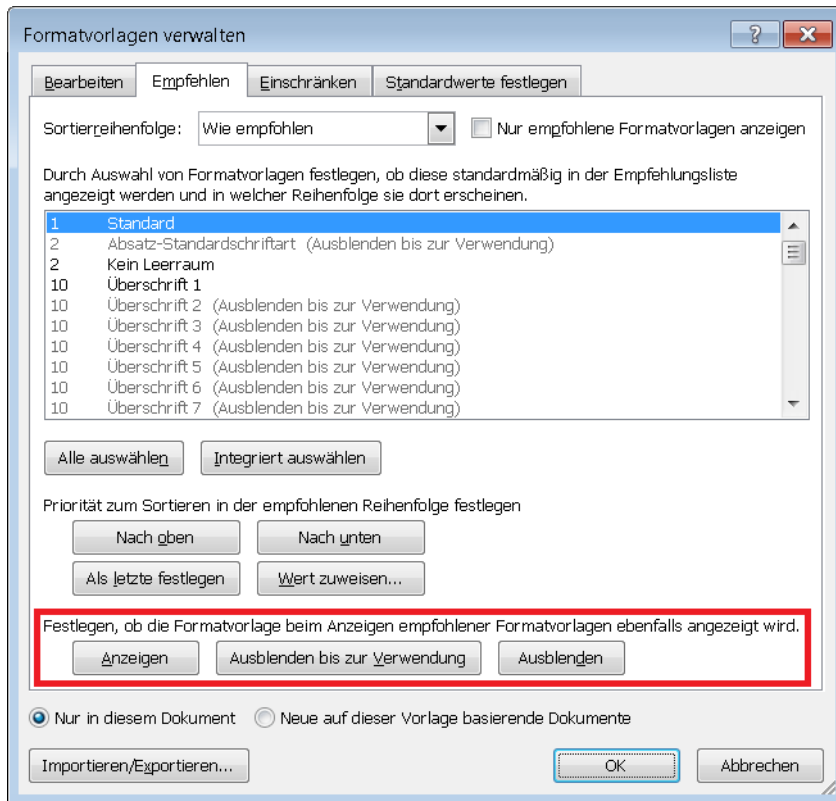
Im vierten Arbeitsschritt werden die benötigten Formatvorlagen definiert. Sie werden verstehen, dass wir hier keine Anleitung geben können, wie Sie die Formatvorlagen aufbauen müssen, da die Anforderungen sehr unterschiedlich sind. Grundsätzlich sollten Sie sich jedoch zu den nachstehenden Punkten Gedanken machen:

- Welche der vordefinierten Formatvorlagen müssen auf die eigenen Bedürfnisse hin angepasst werden? Aus unserer Sicht müssen mindestens die Eigenschaften der nachstehenden Formatvorlagen überprüft werden:
 - *Standard* und *Standardeinzug*
 - *Überschrift 1* bis *Überschrift 4* und *Verzeichnis 1* bis *Verzeichnis 3*
 - *Kopfzeilen* und *Fußzeilen*
 - *Normale Tabelle*
 - *Fußnotenzeichen* und *Fußnotentext*
 - *Endnotenzeichen* und *Endnotentext*
 - *Hyperlink* und *BesucherHyperlink*
- Welche Zeichen- und Absatzformate werden den benötigten Formatvorlagen zugewiesen?
- Welche Formatvorlagen müssen in der Liste der Schnellformatvorlagen sichtbar sein? Viele Formatvorlagen werden von Word automatisch zugewiesen und müssen deshalb nicht unbedingt angezeigt werden (beispielsweise die Formatvorlage *Kopfzeilen* innerhalb von Kopfzeilen).

Zum Bearbeiten und Konfigurieren der Formatvorlagen wählen Sie im Menüband *Start/Formatvorlagen* das Startprogramm für den Dialog (kleines Symbol in der rechten unteren Ecke in der Befehlsgruppe *Formatvorlagen*) an und betätigen die Schaltfläche *Formatvorlagen verwalten*. Definieren Sie im entsprechenden Dialogfeld die Eigenschaften aller bestehenden Formatvorlagen und fügen Sie gegebenenfalls eigene Formatvorlagen hinzu.

An dieser Stelle verweisen wir erneut auf das Buch »Office 2007 – Das Profibuch« in welchem dieses Thema detaillierter behandelt wird.

Abbildg. 13.2 Sichtbarkeit der Formatvorlagen festlegen



Silbentrennung konfigurieren

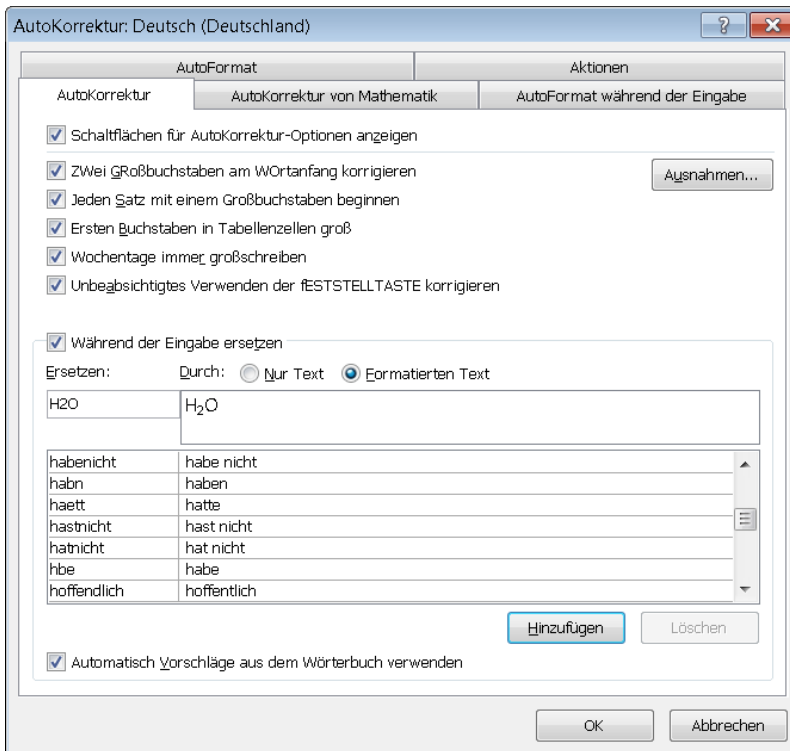
Im fünften Arbeitsschritt werden die Eigenschaften der Silbentrennung festgelegt. Um die automatische Silbentrennung zu aktivieren, rufen Sie den Befehl *Seitenlayout/Seite einrichten/Silbentrennung/Silbentrennungsoptionen* auf, aktivieren das Kontrollkästchen *Automatische Silbentrennung* und legen die weiteren Eigenschaften der Silbentrennung fest.

AutoKorrektur definieren

Im sechsten Arbeitsschritt wird die Liste der vorhandenen AutoKorrektur-Einträge überarbeitet und bei Bedarf mit eigenen Einträgen ergänzt. Eigene Einträge sind dann besonders sinnvoll, wenn innerhalb einer Firmenumgebung sichergestellt werden soll, dass einzelne Wörter von allen Anwendern immer gleich geschrieben und gleich formatiert werden sollen. Als Beispiel können Produktbezeichnungen, chemische Formeln, firmeninterne Terminologien usw. genannt werden.

Um die Liste der AutoKorrektur-Einträge zu bearbeiten, wählen Sie *Datei/Optionen*. In der Kategorie *Dokumentprüfung* befindet sich die Schaltfläche *AutoKorrektur-Optionen*. In der entsprechenden Liste werden überzählige Einträge entfernt und eigene Einträge hinzugefügt.

Abbildg. 13.3 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 Befehl *Datei/Optionen/Dokumentprüfung/AutoKorrektur-Optionen* auf. Aktivieren Sie im Dialogfeld *AutoKorrektur* die Option *Formatierten Text* und tragen Sie den zu ersetzenden Wert im Feld *Ersetzen* ein.

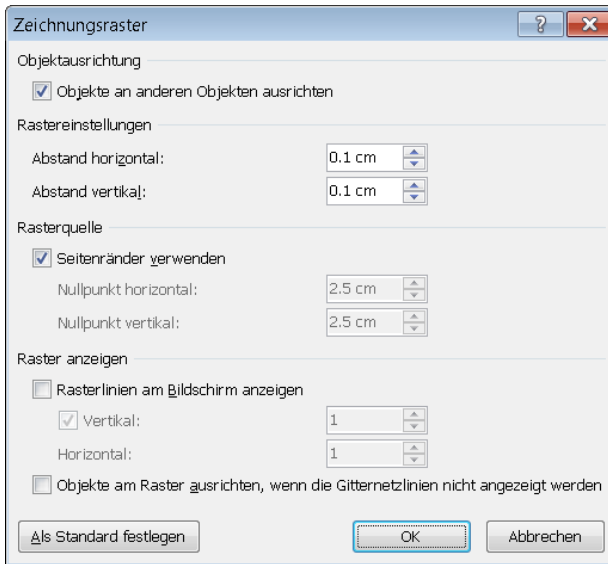
Zeichnungselemente optimieren

Der siebte Punkt, der beachtet werden sollte, widmet sich den Einstellungen für die Zeichnungselemente. Hier liegen uns die Konfiguration für das Zeichnungsrastrer und die Vorgabewerte für die Autoformen am Herzen.

Mit dem Zeichnungsrastrer 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.

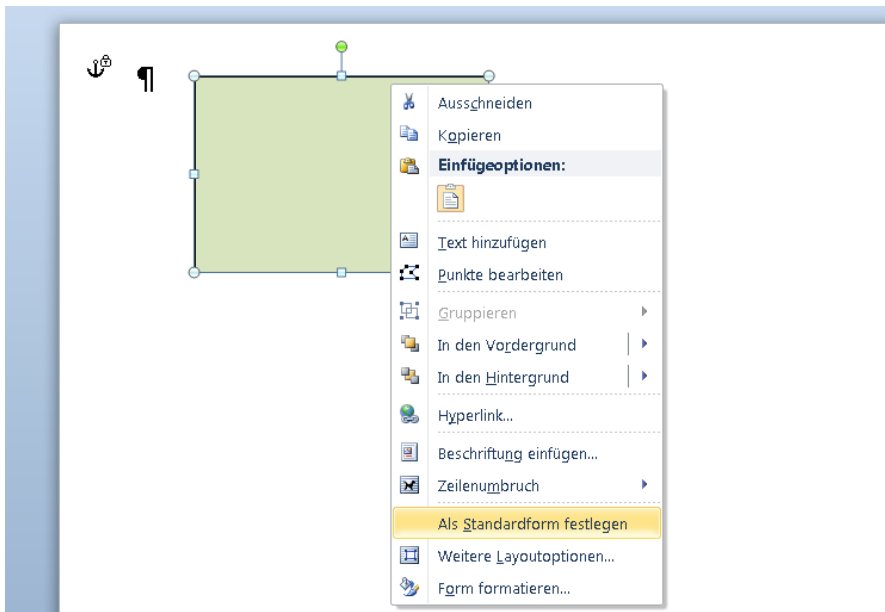
Die Einstellungen für das Zeichnungsrastrer können wie folgt festgelegt werden. Rufen Sie den Befehl *Seitenlayout/Anordnen/Ausrichten/Rastereinstellungen* auf und tragen Sie die gewünschten Einstellungen ein.

Abbildg. 13.4 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 Linienfarbe, Füllfarbe, Schatten usw.). Wählen Sie anschließend im Kontextmenü der Autoform den Befehl *Als Standardform festlegen* an. Die temporäre Autoform kann jetzt wieder entfernt werden.

Abbildg. 13.5 Standardwerte für Autoformen festlegen



Optionen bearbeiten

Im achten Schritt werden die Optionen von Word bearbeitet. Um die Einstellungen zu bearbeiten, wählen Sie im Menüband den Befehl *Datei/Optionen* an. Hier werden jetzt Schritt für Schritt die relevanten Kategorien und deren Abschnitte bearbeitet.

WICHTIG

Einige der Einstellungen, die hier definiert werden können, sind für die gesamte Programmumgebung gültig. Die anderen Einstellungen haben einen direkten Bezug auf das aktuelle Dokument oder die aktuelle Ansicht.

An dieser Stelle werden nur jene Einstellungen behandelt, deren Bezug sich auf das aktuelle Dokument, in diesem Falle auf die *Normal.dotm*, beziehen.

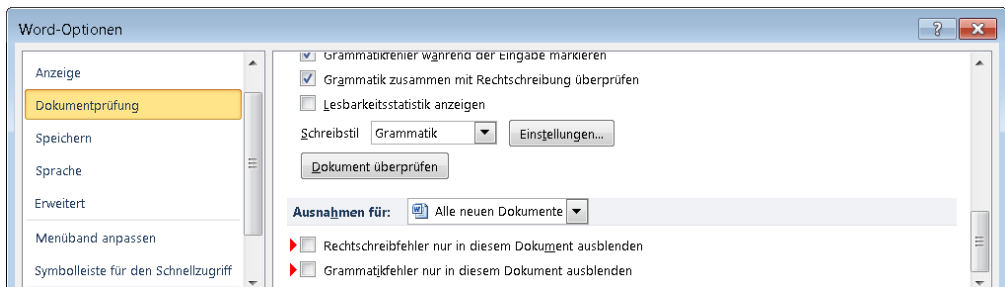
Wir verzichten darauf, jede einzelne Einstellung explizit mit den zugehörigen Werten aufzulisten. Stattdessen versuchen wir, die wichtigsten Punkte kurz zu erläutern. Alle eingestellten Optionen und die zugehörigen Vorschlagswerte sind in Abbildung 13.6 bis Abbildung 13.10 zusammengefasst. Zum Hervorheben der betroffenen Optionen wurde auf den zugehörigen Abbildungen der Text zusätzlich markiert. Die einzelnen Punkte werden in der gleichen Reihenfolge vorgestellt, wie die einzelnen Kategorien im Dialogfeld *Word-Optionen* aufgeführt werden.

Beachten Sie die nebenstehend gezeigte Markierung in den nachfolgenden Abbildungen!

Kategorie Dokumentprüfung

Aus Sicht der Autoren ist es angebracht, wenn dem Anwender die beiden Hilfsmittel als Standard zur Verfügung gestellt werden. Deshalb werden diese beiden Kontrollkästchen nicht aktiviert.

Abbildg. 13.6 Festlegen der nötigen Optionen in der Kategorie *Dokumentprüfung*

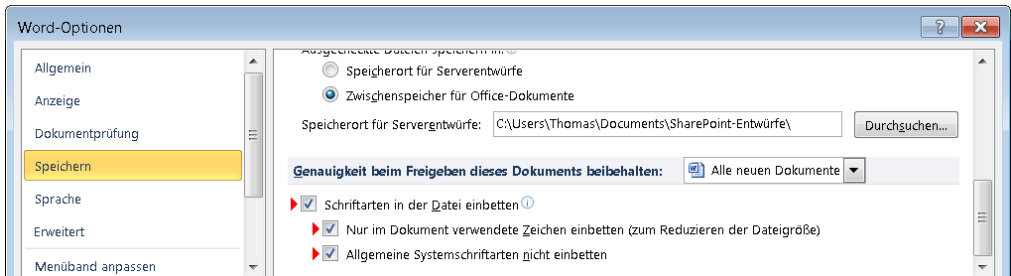


Kategorie Speichern

Kommt in einer Firmenumgebung eine spezielle Schriftart zum Einsatz, welche auf das Corporate Design abgestimmt ist, ist es mehr als sinnvoll, wenn die entsprechenden Schriftenarten in die einzelnen Dokumente eingebunden werden. So ist 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.

Abbildg. 13.7 Festlegen der nötigen Optionen in der Kategorie *Speichern*


Eigenschaften der TrueType-Schriften beachten



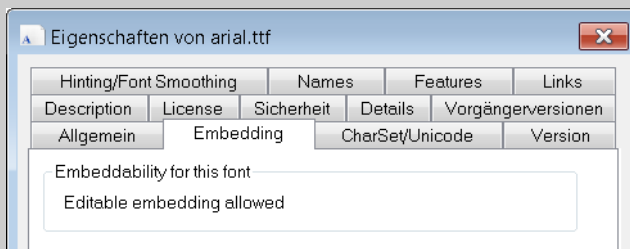
TrueType.ttf

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) erworben 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. 13.8 Die Eigenschaften der TrueType-Schrift prüfen


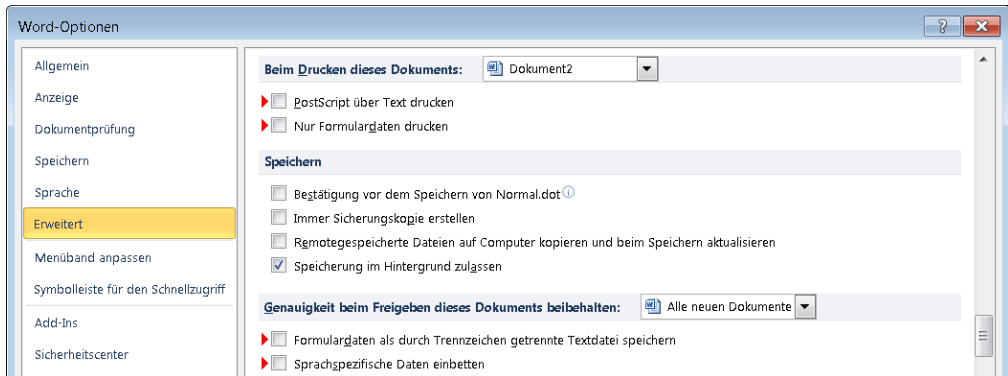
CD-ROM

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>.

Kategorie Erweitert

Das Aktivieren des Kontrollkästchens *Sprachspezifische Daten einbetten* scheint von der Bezeichnung her sinnvoll zu sein. Ein direkter Nutzen ist jedoch nicht vorhanden, da die entsprechende Schnittstelle (Erkennung von Spracheingabe und Handschrift) in der deutschen Version von Word nicht implementiert ist.

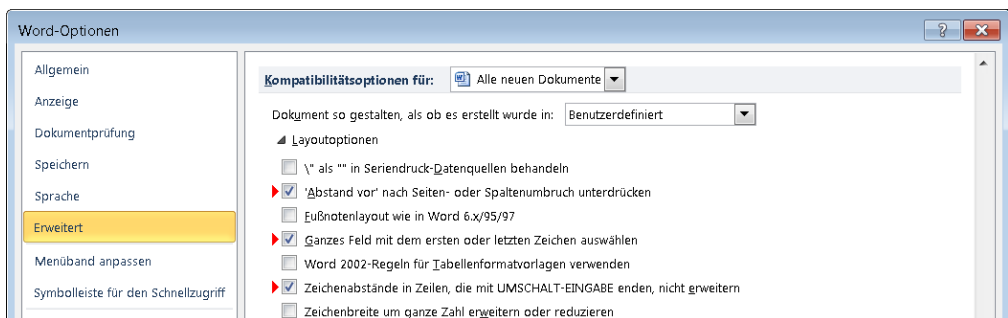
Abbildg. 13.9 Festlegen der nötigen Optionen in der Kategorie *Erweitert*



Von den möglichen Einstellungen von Word zur Kompatibilität mit älteren Programmversionen wurden drei Optionen ausgewählt, die den Anwender direkt bei der Arbeit unterstützen.

HINWEIS Möchten Sie erfahren, was die einzelnen Optionen bedeuten und welche Auswirkungen diese auf das Dokument haben, 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. 13.10 Festlegen der notwendigen Kompatibilitäts-Optionen in der Kategorie *Erweitert*



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.dotm* übernehmen möchten.

AutoTexte entfernen

In älteren Versionen von Word (Word 2003 und früher) war es zudem sinnvoll, wenn die Liste der ausgelieferten AutoTexte manuell ausgedünnt wurde. 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.

WICHTIG

Gemeinsam genutzte AutoTexte werden auf keinen Fall in der *Normal.dotm* 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 ab Version 2007 verweisen wir auf die Diskussion zu Bausteinen in dem bei Microsoft Press veröffentlichten Buch »Office 2007 – Das Profibuch«.

Vorlage *UrNormal.dotm* speichern

Als letzten Arbeitsschritt speichern Sie Ihre fertig konfigurierte *UrNormal.dotm* ab. Erstellen Sie mindestens eine Sicherheitskopie dieser wertvollen Datei.

Stellen Sie diese Datei im Netzwerk zur Verfügung, sodass für jeden Anwender vor dem ersten Start von Word eine Kopie im Ordner mit den *Benutzervorlagen* angelegt wird.

Benutzereinstellungen abspeichern

In der konfigurierten *Normal.dotm* 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 selbst werden unzählige Benutzerparameter und -einstellungen abgespeichert. Diese Werte werden entweder in der entsprechenden Dokumentvorlage, meistens ist dies die *Normal.dotm*, oder in der Windows-Registrierung abgelegt.

In den meisten Windows-Programmen ist es ebenfalls üblich, dass der Anwender persönliche Einstellungen, die das Verhalten des Programms beeinflussen, abspeichern kann. Oder das Programm speichert ohne das Zutun des Anwenders gewisse Einstellungen ab, um diese zu einem späteren Zeitpunkt erneut verwenden zu können.

Diese Funktionalität kann ebenfalls in die Makros integriert werden und unterstützt den Anwender zusätzlich bei seiner täglichen Arbeit. Besonders bei Makros, die mit einem Dialogfeld arbeiten, ist es oft sinnvoll, wenn der Status der Kontrollkästchen, Optionen usw. zwischengespeichert wird. Um spezielle Optionen des Anwenders zu speichern, sind zwei unterschiedliche Speicherorte möglich:

- Die Windows-Registrierung zum Abspeichern von unterschiedlichsten Daten, bezogen auf die aktuelle Arbeitsstation oder den aktuellen Anwender (Benutzerprofil)
- Das Dateisystem als Speicherort von sämtlichen Dateien, im aktuellen Fall in Form von Konfigurationsdateien

Für welche der beiden Arten Sie sich entscheiden, bleibt Ihnen überlassen, es ist durchaus sinnvoll im gleichen Makro beide Arten zu verwenden. Als Faustregel können wir Ihnen Folgendes empfehlen:

- Persönliche Einstellungen des Anwenders werden in der Windows-Registrierung abgespeichert (beispielsweise die Position des Dialogfelds)
- Allgemeine Konfigurationsparameter zum Makro werden in einer eigenen Konfigurationsdatei abgespeichert (beispielsweise die Definition eines zugehörigen Datenverzeichnisses)
- Allgemeine Daten zum Makro werden in einer Konfigurationsdatei gespeichert (beispielsweise die Aufstellung aller möglichen Einträge in einem Listefeld)

Im Befehlsumfang von Visual Basic for Applications sind zwei Funktionen und eine Methode integriert, welche das Schreiben und Lesen von Benutzereinstellungen unterstützen.

Was ist eine Konfigurationsdatei?



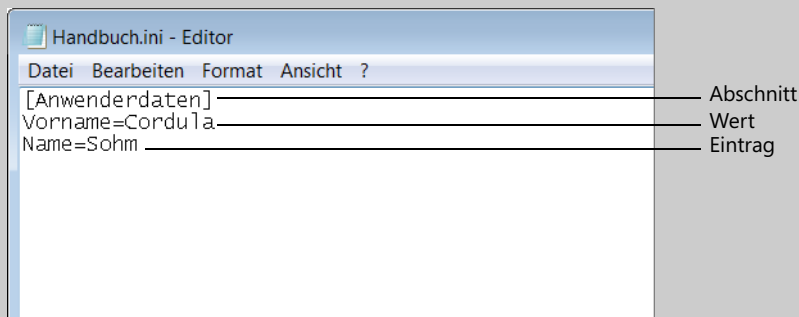
Handbuch.ini

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 (siehe Seite 664) wird ebenfalls eine *.ini*-Datei aufgebaut. Das Resultat der entsprechenden Programmsequenz ist in Abbildung 13.11 dargestellt.

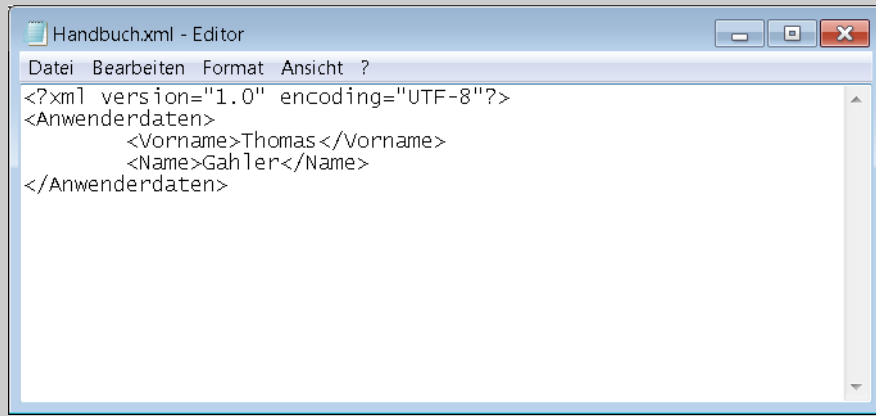
Abbildg. 13.11 Der Aufbau einer *.ini*-Datei



Die Möglichkeiten der Konfigurationsdateien standen vor allem in Microsoft Windows 3.11 und älteren Versionen im Vordergrund. Seit der Einführung von Microsoft Windows 95 steht die zentrale Windows-Registrierung dem Programmierer zur Verfügung. Viele Einstellungen, die früher in *.ini*-Dateien hinterlegt wurden, werden in den aktuellen Versionen von Windows in der Windows-Registrierung gespeichert.

Der allgemeine Trend in der Softwareentwicklung zeigt jedoch, dass Anwendungsdaten wieder vermehrt außerhalb der Windows-Registrierung abgespeichert werden. Der Grund dazu steht im Zusammenhang mit den Problemen, die entstehen können, wenn immer größere Datenmengen in dieser Datei abgelegt werden. Anstelle von *.ini*-Dateien verwenden Entwickler immer häufiger eine *.xml*-Datei zum Speichern solcher Daten.

Abbildg. 13.12 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")

    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 sogenannten 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 anstelle 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:\Temp\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

```

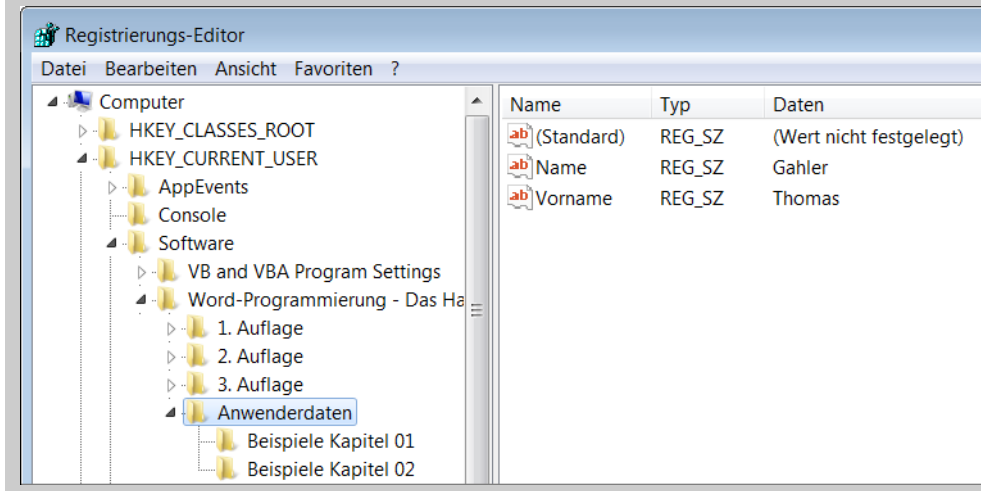
Daten in der Windows-Registrierung abspeichern

Werden persönliche Einstellungen des Anwenders in der Windows-Registrierung abgespeichert, 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. 13.13 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 & _
```

Listing 13.5 Auslesen der Anwenderdaten mittels *PrivateProfileString* aus der Windows-Registrierung (Fortsetzung)

```

    "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:\Temp\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 & _
        "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 Variablen 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, _

```

Listing 13.7 *PrivateProfileString*-Eigenschaft als eigene Funktion mit Standardrückgabewert erweitert (Fortsetzung)

```

    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

    '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 21 beschrieben. Mit den nachstehenden Programmzeilen soll erläutert werden, wie Anwenderdaten in die XML-Datei der Abbildung 13.12 (siehe Seite 662) 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:\Temp\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:\Temp\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
```


CD-ROM Das dargestellte Beispiel finden Sie in der Beispieldatei *Bsp13_01.docm* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap13*. Damit das Beispiel ausgeführt werden kann, muss im Ordner *C:\Temp* die Datei *Handbuch.xml* vorhanden sein.

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 näherbringen:

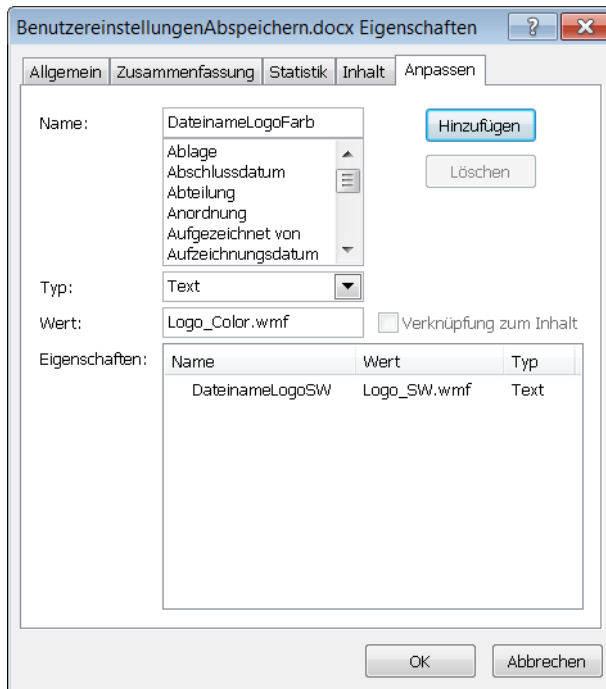
- Hinterlegen des Dateinamens für das Firmenlogo, welches beim Drucken auf einen Schwarz-weißdrucker bzw. bei einem Farbdrucker in das Dokument eingebunden wird
- Hinterlegen der Anzahl, welche als Vorgabewert im Dialogfeld für den Ausdruck vorgeschlagen wird (beispielsweise die Anzahl Personen gemäß Verteilerliste in einem Protokoll)
- Abspeichern des Passworts für geschützte Dokumente, damit zu einem späteren Zeitpunkt der Dokumentschutz aufgehoben bzw. dieser automatisch mit einem Makro bearbeitet werden kann

Zum Abspeichern dieser Werte stehen Ihnen zwei verschiedene Möglichkeiten zur Verfügung. Beide haben ihre Vor- und Nachteile.

Die erste Möglichkeit ist die Verwendung der *Dokumenteigenschaften*. Diese können sehr einfach auf die persönlichen Bedürfnisse hin angepasst und erweitert werden:

- Es können verschiedene Datentypen abgespeichert werden
- Alle gespeicherten Werte können jederzeit eingesehen werden
- Die Werte können auch von außerhalb, beispielsweise direkt im Windows-Explorer, eingesehen werden
- Die Werte können von jedem Anwender sehr einfach manipuliert werden
- Die Werte können manuell oder mit Codeprozeduren bearbeitet werden

Um eine Dokumenteigenschaft manuell zu bearbeiten, rufen Sie *Datei/Informationen/Eigenschaften/Erweiterte Eigenschaften* auf und wechseln zur Registerkarte *Anpassen*.

Abbildg. 13.14 Dokumenteigenschaft zum Verwalten des Firmenlogos erfassen


Die zweite Möglichkeit zum Abspeichern von Werten innerhalb eines Dokuments sind die *Dokumentvariablen*:

- Es stehen keine unterschiedlichen Datentypen zur Verfügung. Dokumentvariablen bestehen aus Zeichenketten. Sämtliche Werte müssen zuerst konvertiert werden.
- Die Werte können vom Anwender manuell nicht manipuliert werden
- Die Werte können nur mittels Codeprozeduren bearbeitet werden

Dokumenteigenschaften bearbeiten

Bei den Dokumenteigenschaften gibt es zwei Kategorien. Die erste Kategorie beinhaltet die *BuiltInDocumentProperties*, in welchen die internen Werte von Word (beispielsweise der Name des Autors, das letzte Druckdatum usw.) abgelegt werden. Bei der zweiten Kategorie handelt es sich um die *CustomDocumentProperties*. Hier kann der Anwender seine eigenen Werte hinterlegen.

Die *CustomDocumentProperties* ist eine Auflistung. Diese kann mit den Methoden *Add*, *Delete* usw. bearbeitet werden.

Listing 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.

CD-ROM

Die Datei *DsoFileSetup_KB224351_x86.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

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-ROM zum Buch vor. Die entsprechende Datei *Kap13_CS.zip* befindet sich im Ordner *\Beispiele\Kap13\C_Sharp*. 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 vrb1 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 vrb1 In docDemo.Variables
        MsgBox "Name" & vbTab & vrb1.Name & vbCr & _
            "Wert" & vbTab & CStr(vrb1.Value)
    Next vrb1

    '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.

CD-ROM

Das dargestellte Beispiel finden Sie in der Beispieldatei *Bsp13_01.docm* 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.dotm* explizit zu konfigurieren und welche Schritte dazu nötig sind (Seite 650 ff.)
- Im Weiteren wurde erklärt, wie Benutzereinstellungen innerhalb einer Konfigurationsdatei oder der Windows-Registrierung abgespeichert und von dort wieder gelesen werden können (Seite 660)
- Im dritten und letzten Abschnitt wurde erläutert, wie Dokumenteinstellungen abgespeichert und wieder gelesen werden können (Seite 669)

Kapitel 14

Speicherort der Anpassungen

In diesem Kapitel:

Der Speicherort einer Anpassung	676
Kontext für COM-Add-Ins	681
Hierarchie der Anpassungen	682
Zusammenfassung	682

Heutzutage stellen alle Office-Anwendungen ähnliche Benutzerschnittstellen zur Verfügung. Dazu gehören beispielsweise Dialogfelder, Menüband (oder auch Multifunktionsleiste bzw. »Ribbon«) 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 an 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 alle alten Befehle (inzwischen unterstützt Word etwa 1.800 einzelne Befehle).

Ab Office 2007 ersetzt die Multifunktionsleiste (bzw. das Menüband) zusammen mit der Symbolleiste für den Schnellzugriff (»QAT«) die Menüs und 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 zum Bearbeiten dieser 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* bzw. *FilePrint* (englische Befehle funktionieren allgemein, unabhängig von der installierten Sprachversionen von Word) 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.

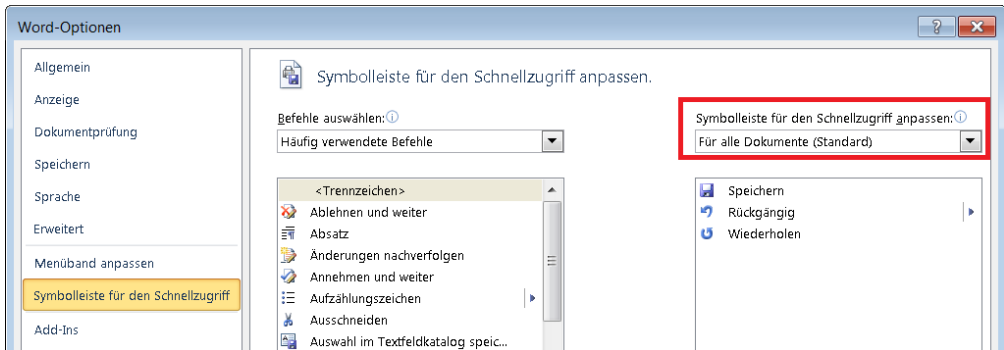
Schließlich kann jeder interne Befehl oder jedes Makro (`Public Sub`, ohne Argumente) einer beliebigen Tastenkombination zugewiesen werden (*Datei/Optionen/Menüband anpassen/Anpassen* in Word oder über die `KeyBindings`-Auflistung des Objektmodells, dies wird in Kapitel 18 detailliert beschrieben).

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

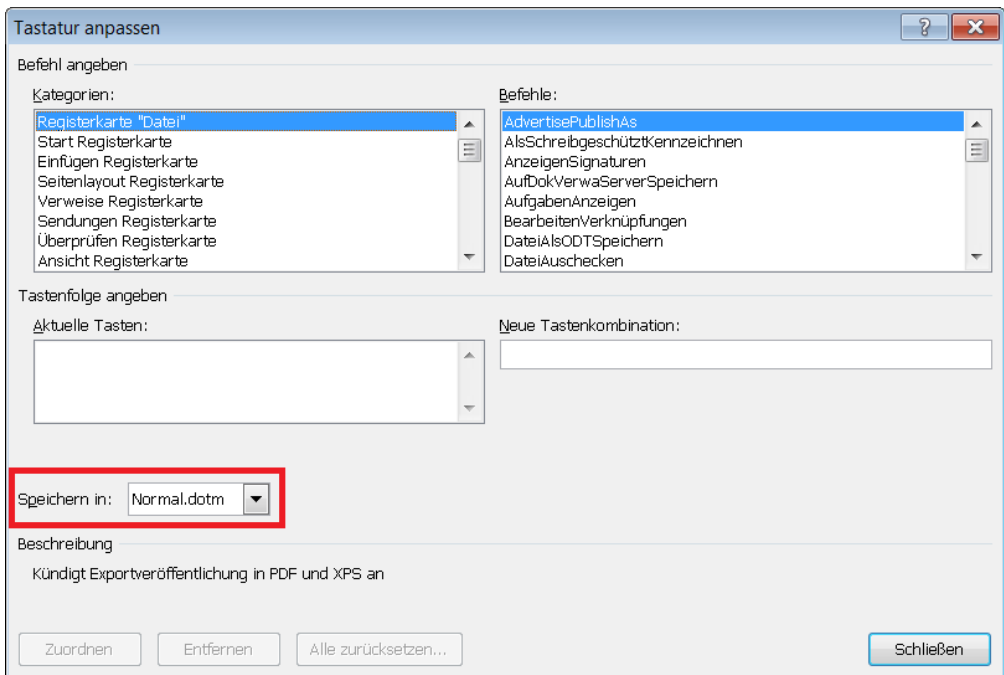
Im Gegensatz zu Excel oder PowerPoint bietet Word mehrere Speicherorte für Anpassungen, den sogenannten *Kontext*. Die Dialogfelder *Symbolleiste für den Schnellzugriff anpassen* (Abbildung 14.1) und *Tastatur anpassen* (Abbildung 14.2) verfügen über eine entsprechende Dropdownliste, diese enthält alle momentan gültigen Speicherorte.

Abbildg. 14.1 Bei Anpassungen an der Symbolleiste sollte die Auswahl für den Speicherort kontrolliert werden



Je nach gewählter Option stehen die *Normal.dotm* und das aktuelle Dokument zur Verfügung. Weiterhin können alle geladenen Vorlagen-Add-Ins sowie die Dokumentvorlage des aktuellen Dokuments, sofern es sich bei diesem nicht selbst um eine Dokumentvorlage handelt, aufgeführt werden.

Abbildg. 14.2 Beim Zuweisen einer Tastaturkombination sollte der Speicherort gezielt festgelegt werden



Makros, Symbolleiste für den Schnellzugriff und Tastenkombinationen können sowohl in einem Dokument als auch einer Dokumentvorlage gespeichert werden. AutoText- bzw. Bausteineinträge 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.dotm* sowie
- 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, QAT 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.dotm	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. Die 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.dotm	Persönlich genutzte Makros zur Optimierung des Arbeitsumfelds
Add-In	Steht allen Anwendern uneingeschränkt zur Verfügung

Grundsätzlich gilt:

- Sollen Tastaturbelegungen, AutoText bzw. Bausteine 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.dotm* gehört grundsätzlich dem Anwender. Hier haben Sie, als »Fremder«, nichts zu suchen. Mehr zum Thema *Normal.dotm* lesen Sie in Kapitel 13.

WICHTIG Hände weg von *Normal.dotm*



Normal.dotm

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

Wir Autoren müssen immer wieder feststellen, dass Hersteller von Programmiererweiterungen die entsprechenden Makros in der *Normal.dotm* abspeichern. So wird die Datei mit Makros und Symbolleisten, welche im Menüband auf der Registerkarte *Add-Ins* eingeblendet werden, erweitert oder – noch schlimmer – während der Programminstallation sogar vollkommen ersetzt.

Wir vertreten grundsätzlich die Meinung, dass die *Normal.dotm* 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.

Die Eigenschaftssyntax sieht wie folgt aus:

```
Application.CustomizationContext = [Dokument- oder Vorlagenobjekt]
```

Die Syntax für C# oder eine andere Programmierungsumgebung bleibt die gleiche. Jedoch wird eine Variable benötigt, der das `Application`-Objekt zugewiesen wurde:

```
wdApp.CustomizationContext = doc
```

Um die Anpassung in der standardmäßigen globalen Vorlage *Normal.dotm* 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.docx"
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, sodass 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.dotm"
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 Kontexts 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 muss der Kontext bei Bedarf ebenfalls festgelegt werden.

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 (*.dotm*) zur Verfügung steht, müssen die Anpassungen in der *Normal.dotm* 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.dotm* 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.

Falls Sie ein COM-Add-In für eine dokumentspezifische Aufgabe programmieren und Ihnen keine eigens dafür vorgesehene Vorlage zur Verfügung steht, fügen Sie nur eine Schaltfläche permanent in die *Normal.dotm* 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.dotm* 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.dotm* beispielsweise 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.dotm* kommen nur zur Geltung, wenn in keinem anderen Kontext eine gleichnamige Anpassung vorhanden ist.

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 676)
- Die bewusste Wahl des Speicherorts legt die Sichtbarkeit der Anpassung fest (Seite 678)
- Die Eigenschaft `CustomizationContext` wird gebraucht, um den Speicherort programmtechnisch festzulegen (Seite 680)
- Bei sogenannten COM-Add-Ins muss ein »externer« Speicherort verwendet werden, damit die erzeugten Anpassungen abgelegt werden können. Diese Änderungen sollten nur temporär erzeugt werden (Seite 681).
- Sind Anpassungen aus mehreren Kontexten gleichzeitig vorhanden, wird hierarchisch entschieden, welche sichtbar sind (Seite 682)

Kapitel 15

Mit Dialogfeldern arbeiten

In diesem Kapitel:

Benutzerdefinierte Dialogfelder	684
Interne Dialogfelder	708
<i>FileDialog</i> -Objekt	715
Zusammenfassung	726

Benötigt ein Makro Informationen, die zur Laufzeit durch den Anwender eingegeben oder ausgewählt werden müssen, 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.

Eine weitere Möglichkeit besteht in der Erstellung eines interaktiven Dialogfelds auf einer Registerkarte im Menüband. Dies kann jedoch nicht mehr ausschließlich in VBA erfolgen, sondern muss per XML vorbereitet werden. Nur die Auswertelogik (Callbacks) der Interaktionen erfolgt in VBA (ausführliche Informationen zur Erstellung und Bearbeitung des Menübands finden Sie in Kapitel 16, im Abschnitt »Code für den dynamischen Ablauf«).

Dieses Kapitel soll Ihnen nun den eher bekannten Weg zeigen, wie eine komplexere Kommunikation mit dem Anwender komplett in VBA aufgebaut werden kann und welche Möglichkeiten dazu zur Verfügung stehen:

- Der direkt folgende erste 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« ab Seite 708 aufgezeigt
- Im Abschnitt »FileDialog-Objekt« ab Seite 715 wird erläutert, welche internen Dialogfelder zusätzlich im Zusammenhang mit dem Dateisystem zur Verfügung stehen

Benutzerdefinierte Dialogfelder

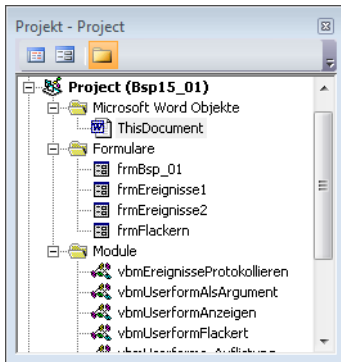


Für benutzerdefinierte Dialogfelder steht die Klasse `UserForm` aus der Bibliothek `MSForms` zur Verfügung. Mit dieser Klasse können Dialogfelder erstellt werden, welche die Schnittstelle zwischen dem eigentlichen Programm und dem Anwender bilden. Alle benötigten Eingabewerte für ein Makro werden in 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 *Listenfeld* 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.

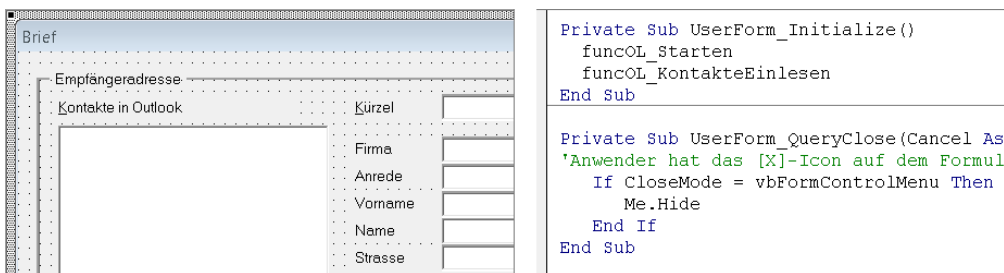
Abbildg. 15.1 Das UserForm-Objekt wird im Zweig *Formulare* zusammen mit dem Projekt gespeichert

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 Codefenster. In diesem Bereich werden alle nötigen Prozeduren und Funktionen hinterlegt, die für das einwandfreie Funktionieren des Dialogfelds 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 aber auch, 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 Codefenster (rechts)



Ein UserForm in verschiedenen Projekten nutzen

Das Erstellen eines benutzerdefinierten Dialogfelds ist mit einem nicht zu unterschätzenden zeitlichen Aufwand verbunden. Denn das eigentliche Design der Bildschirmmaske ist der kleinere Teil. Komplexer hingegen wird das Umsetzen der entsprechenden Logik zwischen den verschiedenen Komponenten und das Überprüfen der Eingaben, denn diese Schritte müssen, wie bereits erwähnt, manuell erstellt werden.

Export und Import

Damit nicht in jedem neuen Projekt ein bestehendes UserForm-Objekt komplett neu erstellt werden muss, kann dieses exportiert und im neuen Projekt importiert werden. Das Exportieren und Importieren eines UserForm-Objekts wurde bereits in Kapitel 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 gemeinsamen genutzten Objekten gearbeitet wird, wurde ebenfalls in Kapitel 10 beschrieben.

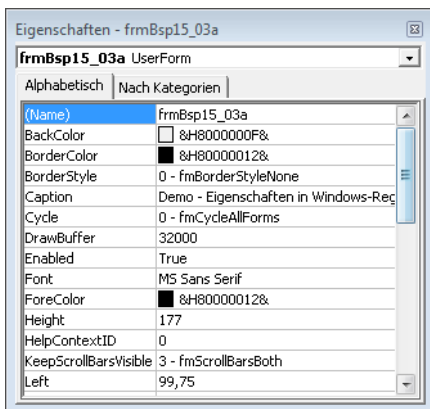
Das *UserForm*-Objekt

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 aussagekräftige 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 ergibt es durchaus einen 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 Dialogfelds 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 (in der Einheit Punkt) 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-Position 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 Top - und Left -Eigenschaft
1 (Fenstermitte)	Das Dialogfeld wird auf dem Element zentriert, von wo aus das UserForm -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.

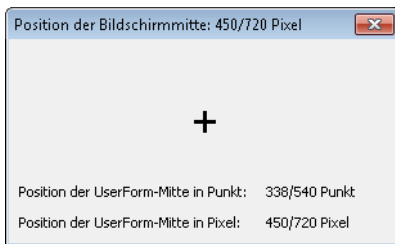
ACHTUNG Für verschiedene Objekt-Eigenschaften werden in VBA unterschiedliche Einheiten verwendet; wichtig an dieser Stelle sind die Einheiten: Pixel und Punkt (Points).

Während die Top- und Left-Eigenschaften des UserForm-Objekts bei der manuellen Startposition in Punkten angegeben werden müssen, verwendet das System-Objekt zur Ermittlung der Bildschirm-Auflösung die Einheit Pixel. Soll ein UserForm beispielsweise manuell in der Bildschirmmitte positioniert werden, können Sie die System.VerticalResolution- und System.HorizontalResolution-Eigenschaft heranziehen, müssen aber zum Setzen der Top- und Left-Eigenschaften des UserForms diese Werte erst in Punkte umrechnen.

Listing 15.1 Umrechnen der Bildschirmauflösung in Punkte zur Positionierung des UserForms auf dem Bildschirm

```
Me.StartupPosition = 0 'Manuell
' Setzen der UserForm-Mitte auf die Bildschirmmitte
Me.Caption = "Position der Bildschirmmitte: " &
    System.VerticalResolution / 2 & "/" & System.HorizontalResolution / 2 & " Pixel"
Me.Top = PixelsToPoints(System.VerticalResolution / 2) - (Me.Height / 2)
Me.Left = PixelsToPoints(System.HorizontalResolution / 2) - (Me.Width / 2)
```

Abbildg. 15.4 Umrechnen der Positions-Einheiten *Punkt* und *Pixel* zur Positionierung auf dem Bildschirm



Zur Umrechnung zwischen den verschiedenen Einheiten können Sie auf verschiedene Methoden des Global-Objekts oder Application-Objekts zurückgreifen. Zu diesen Methoden gehören u.a. Methoden zum Umrechnen von Maßeinheiten, z.B. CentimetersToPoints, InchesToPoints, PixelsToPoints, PointsToPixels.

Das Global-Objekt enthält wie das Application-Objekt Eigenschaften und Methoden der obersten Ebene, denen aber nicht die Application-Eigenschaft vorausgehen muss. Zusätzlich dürfen Sie auch das Global-Objekt nicht der Methode voranstellen, da anderenfalls eine Fehlermeldung ausgegeben wird. So sind folgende Schreibweisen identisch:

```
PointsToPixels(180, False) 'Methode des Global-Objekts
Application.PointsToPixels(180, False) 'Methode des Application-Objekts
```

Beide Anweisungen sind identisch und erzielen dasselbe Ergebnis.

Tag

Die Tag-Eigenschaft dient zum Abspeichern bzw. Zwischenspeichern zusätzlicher Informationen zum aktuellen Dialogfeld, ohne die Einstellungen oder Attribute anderer Eigenschaften zu verändern. Der Inhalt dieser Eigenschaft kann zur Laufzeit ausgewertet oder geändert werden.

Methoden

Alle Methoden werden explizit aus dem Programm heraus aufgerufen. Die Steuerung der Methode erfolgt durch Übergabe von entsprechenden Werten an die zugehörigen Argumente.

Load Um ein UserForm am Bildschirm darzustellen, muss diese vorab in den Speicher geladen werden. Dieser Schritt kann (wie in Listing 15.2 ersichtlich) durch den Aufruf der Load-Methode erfolgen.

Unload 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 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. 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 15.2 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 15.2 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 15.3 entsprechend angepasst.

Listing 15.3 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
```

CD-ROM

Ein Beispiel, wie Sie ein UserForm-Objekt auf die jeweilige Anzeigemethode überprüfen und ein modales Dialogfeld nicht-modal aufrufen können, finden Sie in der Beispieldatei *Bsp15_01.docm* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap15*.

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 wiederholten Neuzeichnen des Dialogfelds am Bildschirm. Aktuelle Rechner sind aber inzwischen so schnell, dass das Flackern nicht mehr so deutlich auffällt.

Diese störende Auswirkung am Bildschirm kann nicht behoben werden. Es besteht jedoch die Möglichkeit, nur einen Teil des Dialogfelds neu zu zeichnen, sofern dieses mittels eines Rahmens (Frame) unterteilt wurde. So kann anstelle von UserForm1.Repaint nur der Rahmen mittels Frame1.Repaint neu gezeichnet werden. Dies würde das Flackern auf den entsprechenden Teil des Dialogfelds einschränken.

CD-ROM

Ein Beispiel, welches das Problem mit dem flackernden Bildschirm aufzeigt, finden Sie in der Beispieldatei *Bsp15_01.docm* 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.

Initialize
Terminate

Jede Klasse verfügt über eine Initialisierungs- bzw. Terminierungssequenz, dies ist auch bei jedem UserForm-Objekt der Fall. Das Initialize-Ereignis tritt ein, wenn eine neue Instanz der Klasse erzeugt wird.

Während der Initialisierung werden in erster Linie Programmsequenzen abgearbeitet, die das benutzerdefinierte Dialogfeld abschließend konfigurieren, denn nicht alle benötigten Eigenschaften können zur Entwurfszeit abschließend gesetzt werden. Dazu gehören unter anderem:

- Eintragen von abgespeicherten Werten der einzelnen Komponenten, um den Zustand des Dialogfelds wie beim letzten Aufruf wieder herzustellen (beispielsweise Position am Bildschirm, Status der Kontrollkästchen usw.). Ein entsprechendes Beispiel ist im Abschnitt »Setzen von Eigenschaften« ab Seite 704 aufgeführt.
- Befüllen von Listefeldern mit Werten aus externen Datenquellen (beispielsweise aus einer Textdatei). Das zugehörige Beispiel ist im Abschnitt »Eigenschaften in .txt-Dateien auslagern« ab Seite 706 beschrieben.
- Aktivieren bzw. Deaktivieren von Komponenten in Abhängigkeit zu anderen Komponenten

Das Terminate-Ereignis entspricht dem Gegenteil. Dieses Ereignis tritt ein, wenn die entsprechende Instanz der Klasse zerstört wird. Während der Terminierung werden vor allem Programmzeilen zum Zwischenspeichern des Status des UserForm-Objekts eingefügt.

Activate
Deacti-
vate
Layout

Im Zusammenhang mit der Bildschirmdarstellung treten drei Ereignisse in den Vordergrund. Das Activate-Ereignis tritt ein, wenn das Dialogfeld aktiviert wird und den so genannten Fokus bekommt. Dies ist bei dessen erster Anzeige sowie beim Wechseln zwischen zwei benutzerdefinierten Dialogfeldern der Fall.

Wird zwischen zwei eigenständigen UserForm-Objekten hin und her gewechselt, tritt im ersten UserForm-Objekt (dasjenige, das den Fokus abgibt) das Deactivate-Ereignis ein.

Beim Verschieben des Dialogfelds am Bildschirm, aber auch bei einer Größenänderung muss das Dialogfeld neu gezeichnet. Bei jedem Neuzeichnen wird das Layout-Ereignis ausgelöst.

HINWEIS

Die einfachste Art, um herauszufinden, in welcher Reihenfolge die einzelnen Ereignisse ausgeführt werden, ist in Listing 15.4 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 Programmierumgebung, wie dies in Abbildung 15.5 ersichtlich ist.

Um den Direktbereich als Fenster innerhalb der Programmierumgebung anzuzeigen, rufen Sie den Menübefehl *Ansicht/Direktfenster* auf.

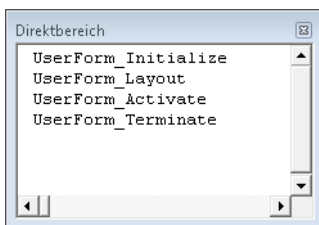
Listing 15.4

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



Query-
Close

Das QueryClose-Ereignis tritt auf, bevor das Dialogfeld geschlossen wird. Dieses Ereignis dient zum Überprüfen der erfassten Daten im Dialogfeld. Wurden die Daten noch nicht gespeichert, kann beispielsweise das Schließen des Dialogfelds verhindert werden.

Dem Ereignis sind zwei Argumente zugeordnet. Wird das erste Argument (Cancel) auf True gesetzt, 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 <i>Unload</i> -Anweisung aufgerufen
vbAppWindows	2	Die aktuelle Windows-Umgebung wird beendet
vbAppTaskManager	3	Die Anwendung wird vom Windows Task-Manager geschlossen

In Listing 15.5 wird als mögliches Beispiel gezeigt, wie verhindert werden kann, dass der Anwender durch Betätigen des in der Titelleiste vorhandenen Schließen-Symbols das benutzerdefinierte Dialogfeld schließen kann.

Listing 15.5 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
```

CD-ROM Das dargestellte Beispiel finden Sie in der Beispieldatei *Bsp15_01.docm* 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 befindliche UserForm-Objekte angesprochen werden.

Anhand des Beispiels in Listing 15.6 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 15.6 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« ab Seite 689 gezeigt, 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 15.7 ist ein Ausschnitt aus dem allgemeinen Programm sowie die Prozedur zur Bearbeitung einer bestimmten Instanz des UserForm-Objekts dargestellt.

Listing 15.7 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 *UserformAnpassen* in Listing 15.7 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)
```

CD-ROM Das vollständige Beispiel finden Sie in der Beispieldatei *Bsp15_01.docm* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap15*.

Steuerelemente einbinden

Das UserForm-Objekt ist ein sogenanntes Containerobjekt. Dies bedeutet, dass es weitere Objekte aufnehmen und darstellen kann. Diese Objekte werden Steuerelemente genannt, denn damit kann der Anwender das Programm über ein Dialogfeld steuern.

Zum Umfang von VBA gehört die *Microsoft Forms 2.0 Object Library*, welche die wichtigsten Steuerelemente zur Verfügung stellt. Grundsätzlich ist es jedoch möglich, weitere Steuerelemente einzubinden, sofern diese als ActiveX-Komponente zur Verfügung steht.

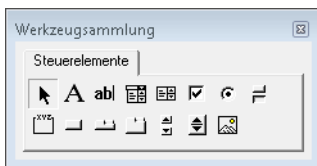
HINWEIS

ActiveX-Steuerelemente von Drittherstellern werden von Office 2010 auf einem 64-Bit System nicht unterstützt.

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 vorgegeben Menge von Werten ein einzelner Wert ausgewählt werden. Es kann definiert werden, ob neben den vorgegebenen Werten auch ein zusätzlicher Wert manuell eingetragen werden kann. Nach erfolgter Auswahl ist nur der gewählte Wert am Bildschirm sichtbar.



Im *Listenfeld* (ListBox) kann aus einer vorgegeben Menge von Werten ein oder mehrere Einträge ausgewählt werden. Es kann definiert werden, ob nur ein Wert oder mehrere ausgewählt werden können. Nach der erfolgten Auswahl bleibt die Liste weiterhin im Dialogfeld sichtbar.



Das *Kontrollkästchen* (CheckBox) dient zum Aktivieren bzw. Deaktivieren einer einzelnen Option. Es kann drei mögliche Zustände annehmen (aktiviert, deaktiviert, unbekannt).



Mit dem *Optionsfeld* (OptionButton) wird eine Auswahl aus einer Gruppe von Optionen zur Verfügung gestellt. Innerhalb der gleichen Gruppe kann nur ein Mitglied gleichzeitig aktiv sein.



Das *Umschaltfeld* (ToggleButton) wird ähnlich wie in den Symbolleisten zur Darstellung der aktuellen Auswahl verwendet. Es wird meistens in kleinen Gruppen zusammen eingesetzt. Es kann jedoch

auch als einzelne Schaltfläche eingesetzt werden. In den meisten Fällen wird zugunsten einer kleinen Grafik auf eine Beschriftung verzeichnet.



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 Containerobjekt 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 Containerobjekt. 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 Containerobjekt.



Die *Bildlaufleiste* (ScrollBar) wird für die Wahl eines Werts 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 Bilds 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 am Bildschirm aktiv sein. Das gewünschte Element wird in der Werkzeugsammlung (ggf. über das Menü *Ansicht/Werkzeugsammlung* aufrufen) angewählt 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.

Nachdem das Steuerelement eingefügt wurde, können die entsprechend zugehörigen Eigenschaften im Eigenschaftenfenster (wird über **F4** eingeblendet) festgelegt werden.

HINWEIS

Grundsätzlich kann ein UserForm-Objekt (oder einzelne Steuerelemente) während der Laufzeit des Makros erzeugt werden. Wie dies funktioniert, ist in Kapitel 20 detailliert beschrieben.

Steuerelemente und ihre Besonderheiten

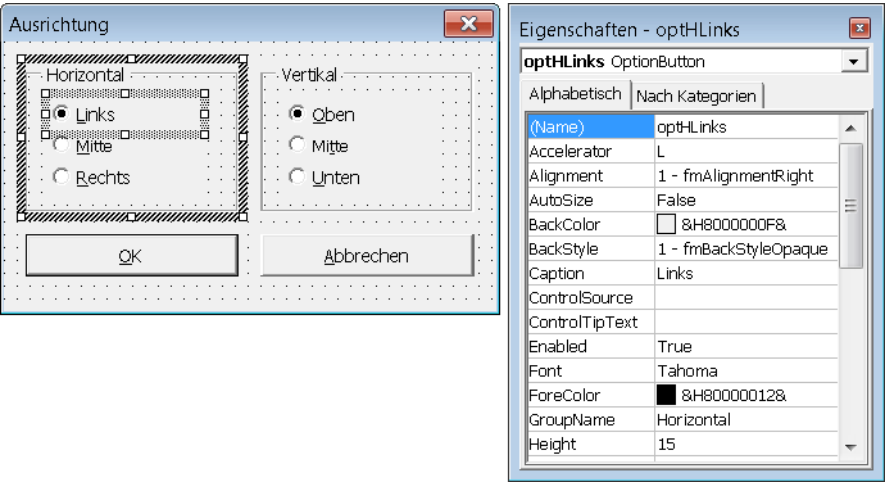
Das Verhalten von einzelnen Steuerelementen aus der »Microsoft Forms 2.0 Object Library« unterscheidet sich teilweise vom bekannten Verhalten aus anderen Entwicklungsumgebungen. Aus diesem Grunde werden eine paar dieser Besonderheiten hier erläutert.

Option (OptionButton)

Auf einem UserForm-Objekt können mehrere *OptionButtons* eingefügt werden. Zusammengehörende Optionen werden, wie in Abbildung 15.7 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 Dialogfelds festgelegt werden, welche Optionen zusammen in einer Gruppe gehören. Um eine Reihe von Optionen zu einer Gruppe zusammenzufassen, muss deren *GroupName*-Eigenschaft mit einer gemeinsamen und eindeutigen Bezeichnung belegt werden.

Abbildg. 15.7 Zusammengehörende Optionen werden in Rahmen mit gleichem *GroupName* 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 15.8 dargestellt, die Auswertung der gewählten Option einzeln erfolgen und kann beispielsweise mit einer *Select Case*-Anweisung erfolgen.

Listing 15.8 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
```

Listing 15.8 Auswerten der gewählten Ausrichtung in den beiden *OptionButton*-Gruppen (Fortsetzung)

```

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:" & vbCr & _
    "Horizontal" & vbTab & strHorizontal & vbCr & _
    "Vertikal" & vbTab & vbTab & 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 füllen 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 + List 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

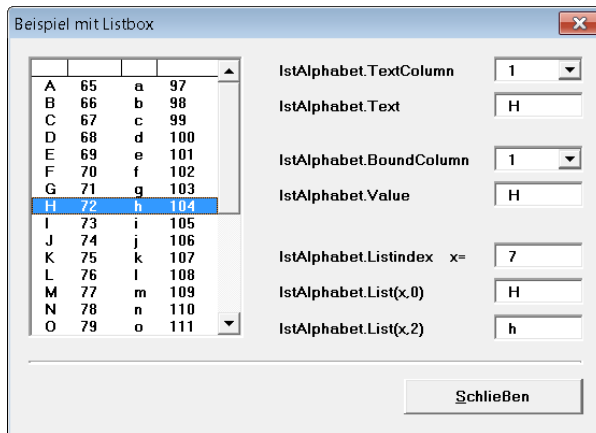
```

Zum Auslesen des gewählten Werts sind in Abbildung 15.8 die verschiedenen Eigenschaften und deren Abhängigkeiten dargestellt. Grundsätzlich stehen drei Eigenschaften zur Verfügung, um den gewählten Eintrag im Listenfeld zu ermitteln.

Text TextColumn	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, 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 ListIndex	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. 15.8 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 des Steuerelements haben die Entwickler leider bis heute nicht implementiert (siehe Abbildung 15.8).

CD-ROM

Die dargestellten Beispiele finden Sie in der Beispieldatei *Bsp15_02.docm* 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) und 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 Dialogfelds.
- 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, wird für die gleichen Bezeichnungen in verschiedenen Dialogfeldern die gleiche Tastaturkombination zugewiesen. Diese sollten möglichst mit jenen aus den internen Dialogfeldern korrespondieren.

Abbildg. 15.9

Komplexes benutzerdefiniertes Dialogfeld, das die Anforderungen an ein Dialogfeld abdeckt

Drucken mit Papierschachtsteuerung

Drucker

Name: HP LaserJet 2300 Series PS Eigenschaften

Typ: HP LaserJet 2300 Series PS

Ort: LPT1: Lokaler Drucker ☐ Ausgabe in Datei umleiten

Papierschacht: Logopapier = 'Briefkopf', Mekulaturpapier = 'Fach 2' i

☐ Drucker 'HP LaserJet 2300 Series PS' nach dem Ausdruck als Standarddrucker setzen

Seitenbereich

☒ Alles ☐ Aktuelle Seite ☐ Markierung

Seiten:

Einzelseiten müssen durch Semikola und Seitenbereiche durch Bindestriche getrennt werden, wie z.B.: 1; 3; 5-12 oder P2S2-P3S5

Exemplare

Anzahl:

☐ Beidseitiger Druck mittels Duplexer

☒ Sortieren Blattreihenfolge: 1 2 3; 1 2 3; 1 2 3

Seiten pro Blatt:

Papierzufuhr | Wasserzeichen | Allgemein | Dokument- und Druckerinformationen

☐ Bestehende Einstellungen für die Papierzufuhr beibehalten 0 / 0

Papierzufuhr für die ersten Exemplare festlegen

Diese Einstellungen haben Gültigkeit für die Exemplare 1 bis

Erste Seite: ☐ Firmenlogo in Kopfzeile einfügen

Übrige Seiten: ☒ Nur erste Seite 'Logopapier' verwenden

Papierzufuhr für die übrigen Exemplare festlegen

Diese Einstellungen haben Gültigkeit für die Exemplare 3 bis 4

Erste Seite: ☐ Firmenlogo in Kopfzeile einfügen

Übrige Seiten: ☐ Nur erste Seite 'Logopapier' verwenden

Optionen... Jokerdruck... 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 soll. 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 jeder Eigenschaft alle benötigten Standardwerte zu.
- Fügen Sie ein erstes Steuerelement dem Dialogfeld hinzu. Weisen Sie dem Steuerelement im entsprechenden Eigenschaftenfenster jeder Eigenschaft 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 hinzugefügt 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 Onlinehilfe zu VBA unter dem Thema »Erstellen eines benutzerdefinierten Dialogfelds« 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, da sie ansonsten verloren gehen.

In Listing 15.9 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 15.10).

Listing 15.9 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

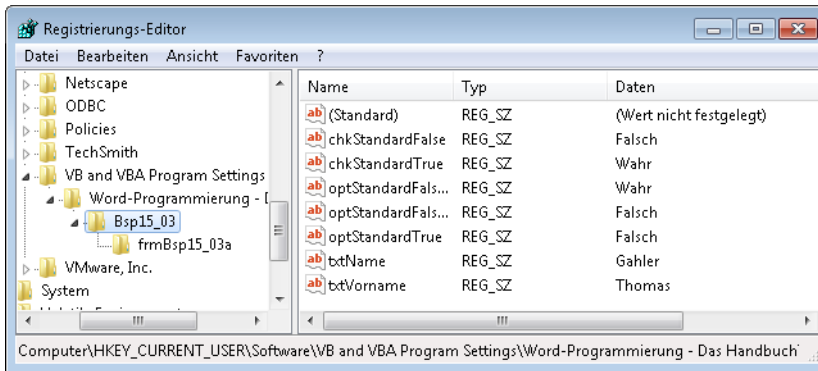
SaveSetting REG_APP, REG_SEC, "chkStandardTrue", chkStandardTrue.Value
SaveSetting REG_APP, REG_SEC, "chkStandardFalse", chkStandardFalse.Value

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. 15.10 Abgespeicherte Werte und Eigenschaften in der Windows-Registrierung


Setzen von Eigenschaften

Wurden die Eigenschaften und Werte aus dem UserForm-Objekt erst einmal an einem zentralen Ort abgespeichert, können diese bei der nächsten Verwendung des Dialogfelds wieder eingetragen werden.

Wird eine neue Instanz des UserForm-Objekts angelegt, wird automatisch das Initialize-Ereignis ausgelöst. Frühestes zu diesem Zeitpunkt können die gewünschten Werte eingetragen werden.

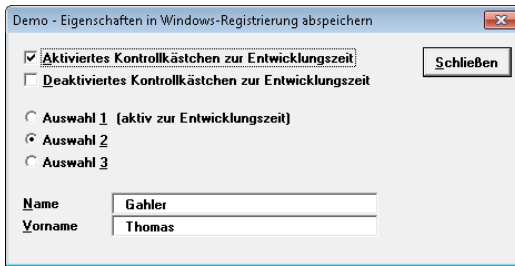
In Listing 15.10 wird gezeigt, wie alle Optionen und die Position des Dialogfelds aus der Windows-Registrierung ausgelesen und den einzelnen Steuerelementen wieder zugewiesen werden. Dies hat zur Folge, dass das Dialogfeld wieder genauso wie beim letzten Aufruf dargestellt wird. Die während der Entwurfszeit eingetragenen Standardwerte werden übersteuert.

Listing 15.10 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. 15.11 Die Werte des Dialogfelds wurden aus der Windows-Registrierung übernommen



Eigenschaften der Dialogfelder in Dateien auslagern

Die meisten Eigenschaften eines benutzerdefinierten Dialogfelds oder der zugehörigen Steuerelemente werden beim Erstellen desselben festgelegt und müssen zur Laufzeit des Programms auch nicht angepasst werden.

Dennoch besteht manchmal das Bedürfnis, einzelne Eigenschaften von Steuerfeldern dynamisch zu gestalten. Dabei kann es sich beispielsweise um eine sprachspezifische Bezeichnung der Steuerelemente oder um einen geänderten Inhalt eines Listenfelds oder Ähnliches handeln.

In beiden Fällen müsste der Programmcode angepasst werden, um diese Änderungen vorzunehmen. Es besteht jedoch die Möglichkeit, diese Werte vom eigentlichen Programm zu trennen und in eine externe Datenquelle auszulagern.

Je nach Anforderung kann als Datenquelle eine Konfigurationsdatei oder eine Datenbank verwendet werden. In den nachstehenden Beispielen werden die Daten jeweils aus unterschiedlichen Konfigurationsdateien eingelesen. Dabei handelt es sich um sogenannte *.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 15.11 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 15.11 Einlesen der Steuerelement-Bezeichnungen aus der Konfigurationsdatei

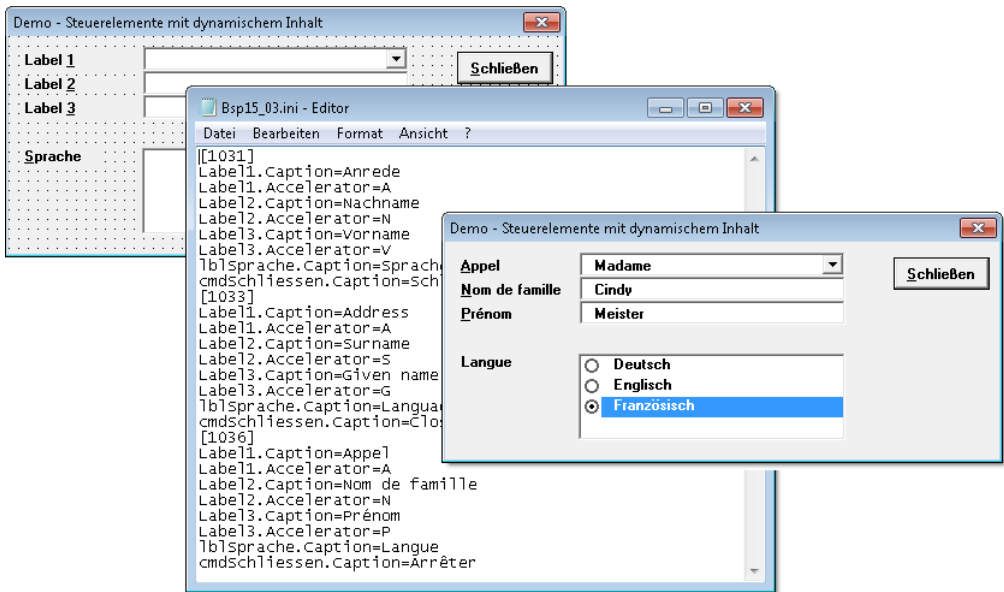
```
'Werte aus .ini-Datei auslesen
strDatei = ThisDocument.Path & "\" & FILE INI
Label1.Caption = fktPrivateProfileString(strDatei, "Allgemein", _
    "Label1.Caption", Label1.Caption)
Label1.Accelerator = fktPrivateProfileString(strDatei, "Allgemein", _
    "Label1.Accelerator", Label1.Accelerator)
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)
```

Anstelle der `PrivateProfileString`-Methode des System-Objekts wird die benutzerdefinierte Funktion `fktPrivateProfileString` verwendet. Diese Funktion wurde bereits in Kapitel 13 vorgestellt.

HINWEIS

In Abbildung 15.12 ist ersichtlich, wie in der `.ini`-Datei `Bsp15_03.ini` für alle Sprachen die entsprechenden Texte hinterlegt sind. Der Sprachcode wurde zusammen mit der Sprache in das Listenfeld eingelesen. Da aber für das Listenfeld über die `ColumnCount`-Eigenschaft eingestellt wurde, dass nur eine Spalte angezeigt werden soll, stehen die Sprachcodes unsichtbar zur Verfügung.

Abbildg. 15.12 Benutzerdefiniertes Dialogfeld zur Entwicklungszeit und zur Laufzeit mit zugehöriger `.ini`-Datei



Eigenschaften in `.txt`-Dateien auslagern

In Listing 15.12 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.

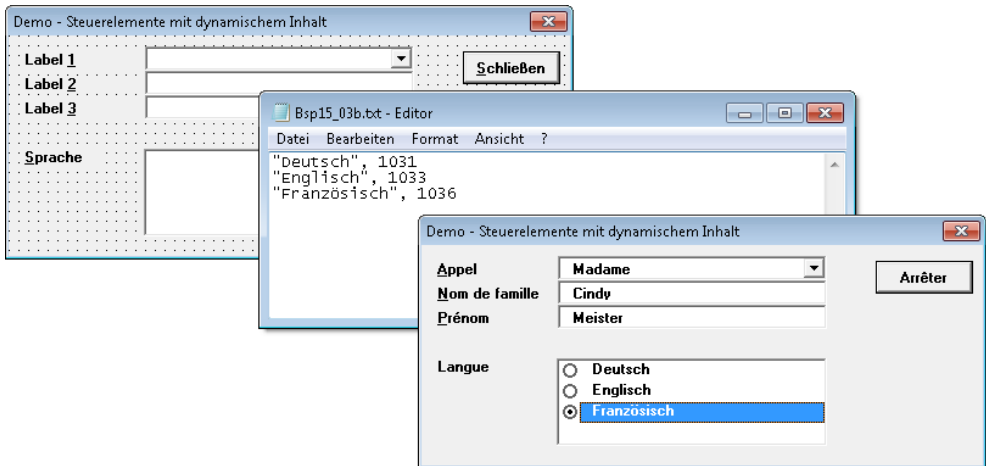
Listing 15.12 Einlesen des Inhalts des Listenfelds aus der Textdatei

```
'Werte aus .txt-Datei auslesen
strDatei = MacroContainer.Path & "\" & FILE_TXT_B
If fktExistiertDatei(strDatei) Then
    Open strDatei For Input As #1
    Do While Not EOF(1)
        Input #1, strEintrag, strINI
        lstSprache.AddItem strEintrag
        lstSprache.List(lstSprache.ListCount - 1, 1) = strINI
    Loop
    Close #1
```

Listing 15.12 Einlesen des Inhalts des Listenfelds aus der Textdatei (Fortsetzung)

```
Else
    lstSprache.AddItem "Deutsch"
End If
```

Abbildg. 15.13 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 15.13 und Listing 15.14 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 Formularobjekt 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 Anwender 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 15.13 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 (MacroContainer.Path & "\" & FILE_XML_B)
    If xmlDoc Is Nothing Then
        MsgBox "Die XML-Datei konnte nicht gefunden werden."
    End If
    If xmlDoc.parseError <> 0 Then
```

Listing 15.13 Daten in eine XML-Datei speichern (Fortsetzung)

```
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.Text
    xmlDoc.SelectSingleNode("//Sprache").Text = frm.lstSprache.Text
    xmlDoc.SelectSingleNode("//Sprache/@index").Text = frm.lstSprache.ListIndex
    xmlDoc.SelectSingleNode("//Sprache/@ID").Text =
frm.lstSprache.List(frm.lstSprache.ListIndex, 1)
End If
xmlDoc.Save ThisDocument.Path & "\" & FILE_XML_B
Set xmlDoc = Nothing
End Sub
```

Listing 15.14 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
```

CD-ROM

Das dargestellte Beispiel finden Sie in der Beispieldatei *Bsp15_03.docm* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap15*.

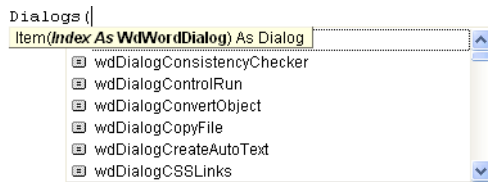
Interne Dialogfelder

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.

Dia-Logs-Auflistung Im Word-Objektmodell 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-Objekts müssen Sie entweder den Index oder den Namen der dem Dialogfeld zugeordneten `WdWordDialog`-Konstante angeben. Laut Count-Eigenschaft der Auflistung stehen in Word 2007 fast 300 Dialogfelder und in Word 2010 304 Dialogfelder zur Verfügung.

Alle benannten Konstanten werden Ihnen im Visual Basic-Editor nach Eingabe der ersten Klammer »(« in einer Auswahlliste angezeigt.

Abbildg. 15.14 Übersicht über die benannten `WdWordDialog`-Konstanten



Dialogfelder anzeigen, anzeigen und ausführen oder ausführen

Nach Auswahl eines Dialogfelds stehen Ihnen nur wenige allgemeine Methoden des Dialogs-Objekts 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 rufen Sie das Dialogfeld zum Speichern eines Dokuments unter einem anderen Namen einmal mit der `Show`-Methode auf:

```
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 <i>OK</i> bzw. die jeweilige Aktion des Dialogfelds beendet (z.B. <code>wdDialogFilePrint</code>). Bei Verwendung der <i>Show</i> -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:

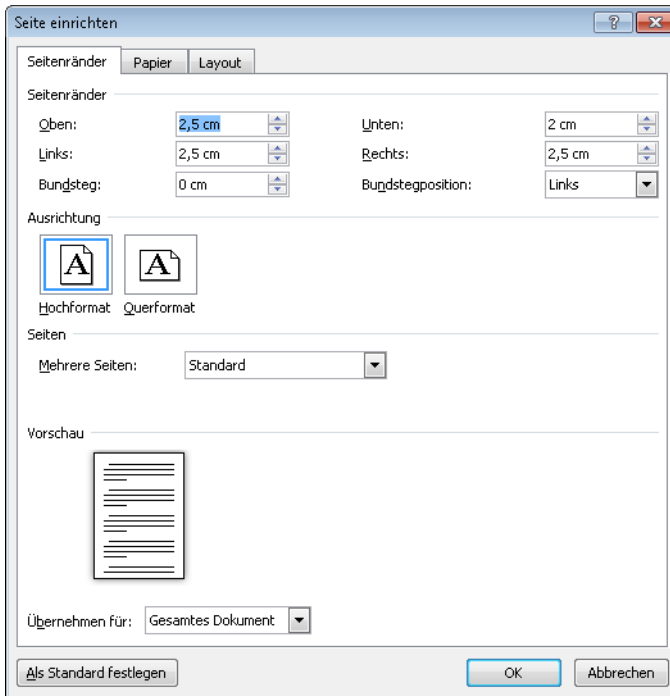
```
Dialogs(wdDialogFileNew).Execute
```

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. 15.15 Anzeige des Dialogfelds zur Seiteneinrichtung *wdDialogFileDocumentLayout*

Wenn die Dokumentseite wie in Abbildung 15.15 im Hochformat eingerichtet ist und Sie anschließend die Seitenorientierung ändern, führt die erneute Anzeige dieses Dialogfelds 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 sämtliche Eigenschaften des Objekts mitgespeichert sind; in diesem Fall ist es die Einstellung zur Seitenorientierung. Diese Einstellungsänderung an der Seite wird jedoch nicht automatisch in den Objektverweis aufgenommen, wenn das Dialogfeld erneut angezeigt wird und stimmt somit mehr mit der gespeicherten Einstellung der Seite überein.

Um alle Änderungen in einem Dialogfeld auch in die erneute Anzeige zu übernehmen, müssen Sie die `Update`-Methode anwenden.

Listing 15.15 Aktualisieren eines per Objektverweis referenzierten Dialogfelds

```
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 nicht alle Argumente unterstützt werden und diese nicht gekennzeichnet sind. So läuft das Ermitteln und Anwenden der Argumente mit ihrer (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 *Wrd-basic.hlp* befindet sich auf der CD-ROM zum Buch im Ordner *\Beilagen\Word95 Wordbasic Hilfe*. Bevor diese alte Hilfedatei genutzt werden kann, muss die zur Windows-Version passende Datei *WinHlp32.exe* heruntergeladen werden (<http://support.microsoft.com/kb/917607>).

In Listing 15.16 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.

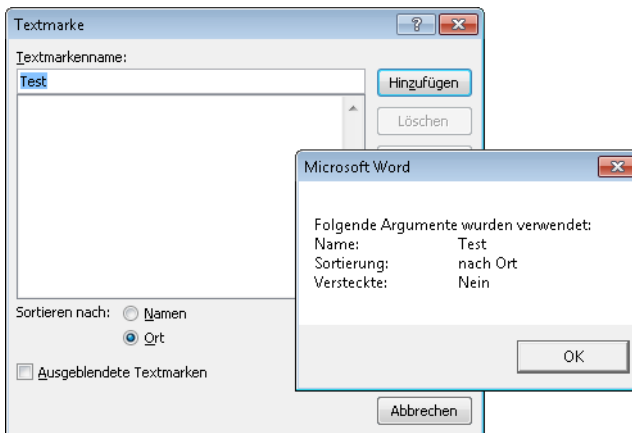
Listing 15.16 Konfigurieren des Dialogfelds *wdDialogInsertBookmark* mittels Argumente und Anzeige der verwendeten Argumente

```
Sub subDialogArguments()
    Dim dlg As Dialog
    Dim strMSG As String
    Set dlg = Dialogs(wdDialogInsertBookmark)
    With dlg
```

Listing 15.16 Konfigurieren des Dialogfelds *wdDialogInsertBookmark* mittels Argumente und Anzeige der verwendeten Argumente (Fortsetzung)

```
.Update
.Name = "Test"
' .SortBy = wdSortByName '0
.SortBy = wdSortByLocation '1
.Hidden = 0 ' wird ignoriert
.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. 15.16 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 15.17 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 15.17 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
```

ACHTUNG Beim Durchlaufen der *For Each...Next*-Anweisung werden Dialoge angesprochen, die im aktuellen Kontext nicht angesprochen werden können (z.B. wenn keine Tabelle markiert ist und das Dialogfeld Zellen in eine Tabelle einfügen würde). Diese Aufrufe können mitunter schwerwiegende interne Fehler auslösen, die in Abstürzen von Word resultieren können.

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/Informationen/Eigenschaften/Erweiterte Eigenschaften* nur über den Index aufrufen:

```
On Error Resume Next ' ohne Fehlerbehandlung kommt es zur Fehlermeldung
Dialogs(750).Display
On Error GoTo 0
```

Die Ermittlung aller über den Index erreichbaren Dialogfelder kann nur anhand einer Schleife erfolgen, die alle möglichen Index-Werte ausprobiert und den Namen des Dialogfelds zurückliefert.

CD-ROM In der Datei *Bsp15_04.docm* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap15* finden Sie das Makro *subFindDialogs*, das Ihnen eine Liste aller Dialogfelder bis zum Indexwert 5000 in einem neuen Dokument auflistet. Dabei werden mitunter Fehlermeldungen oder Hinweismeldungen bezüglich Outlook-Zugriffe ausgegeben, die von den verwendeten Dialogfeldverweisen stammen und sich nicht vermeiden lassen.

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).Execute
```

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.

CD-ROM

Die dargestellten Beispiele finden Sie in der Beispieldatei *Bsp15_04.docm* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap15*.

Sie finden eine vollständige Übersicht auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap15\wdDialogsÜbersicht_Word2010.docx*. Eine Übersicht über die `WdWordDialog`-Konstanten finden Sie in der Datei *\Beispiele\Kap15\wdDialogsKonstanten-Word2010.docx*.

Die internen Dialogfelder von Word sind, neben vielen zusätzlichen Informationen, auch in der Datei *Interne Word-Befehle.xlsx* bzw. *Interne Word-Befehle.pdf* aufgeführt. Diese Dateien befinden sich im Ordner *\Beilagen\Interne Word-Befehle*.

FileDialog-Objekt

Neben den im Abschnitt »Interne Dialogfelder« ab Seite 708 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 Objekts können Sie den Typ und somit die spezifischen Funktionen dieses Objekts auslesen und begrenzt auch festlegen. Der jeweilige Typ des `FileDialog`-Objekts 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
<code>AllowMultiSelect</code>	<code>True/False</code>	Lesen Schreiben	Legt fest, ob mehrere Dateien ausgewählt werden können
<code>Application</code>		Lesen	Gibt die Containeranwendung, in der das FileDialog -Objekt aufgerufen wurde, zurück
<code>ButtonName</code>	»Bezeichnung«	Lesen Schreiben	Legt den Beschriftungstext der auslösenden Aktionsschaltfläche fest oder gibt ihn zurück
<code>Creator</code>		Lesen	Eine 32-Bit-Zahl, die die Containeranwendung repräsentiert
<code>DialogType</code>	<code>msoFileDialogOpen</code> <code>msoFileDialogSaveAs</code> <code>msoFileDialogFilePicker</code> <code>msoFileDialogFolderPicker</code>	Lesen Schreiben	Der durch die MsoFileDialogType -Konstante festgelegte FileDialog -Typ
<code>Filters</code>	<code>FileDialogFilters</code> -Objekt		Gibt eine FileDialogFilters -Auflistung zurück, über die Sie Dateiauswahlfilter definieren können (siehe den Abschnitt »Definieren von Dateiauswahlfilter« ab Seite 718)
<code>FilterIndex</code>	»Index«	Lesen Schreiben	Legt den zu verwendenden Dateifilter der FileDialogFilters -Auflistung fest oder liefert ihn zurück
<code>InitialFileName</code>		Lesen Schreiben	Legt den vorgegebenen Dateinamen inkl. Pfad im Dialogfeld fest oder liefert ihn zurück. Dabei können Sie im Dateinamen die Platzhalter »*« und »?« verwenden.

Tabelle 15.4 Übersicht über die Eigenschaften der *FileDialog*-Objekte (Fortsetzung)

Eigenschaft	Parameter	Zugriff	Beschreibung
InitialView	msoFileDialogViewDetails msoFileDialogViewLargeIcons msoFileDialogViewList msoFileDialogViewPreview msoFileDialogViewProperties msoFileDialogViewSmallIcons msoFileDialogViewThumbnail	Lesen Schreiben	Legt über die MsoFileDialogView -Konstante die Anzeigeeoption der Dateien und Ordner im Dialogfeld fest oder liefert sie zurück
Item		Lesen	Repräsentiert den aktuellen FileDialog -Typ und liefert die Bezeichnung zurück
Parent		Lesen	Gibt das Parent -Objekt »Microsoft Word« des FileDialog -Objekts zurück
SelectedItems		Lesen	Gibt eine FileDialogSelectedItems -Auflistung zurück, in der alle im Dialogfeld markierten Dateien und Ordner enthalten sind, wenn das Dialogfeld mit der Show -Methode angezeigt wird
Title		Lesen Schreiben	Legt den Titel des Dialogfelds fest oder liefert ihn zurück

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 Show -Methode wird die Aktion dabei nicht ausgeführt.
0	Das Dialogfeld wurde über die Schaltfläche <i>Abbrechen</i> , über das Schließen-Symbol in der Titelleiste oder über die Tastenkombination [Alt] + [F4] beendet

Definieren von Dateiauswahlfilter

Filters
FileDia-
logFilters

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

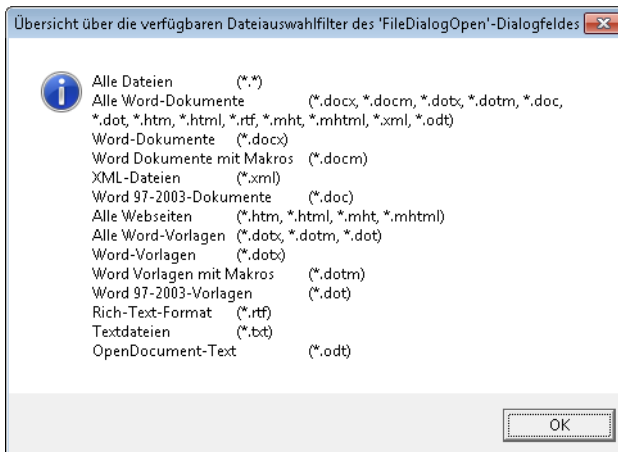
Leider lassen sich diese Filter nicht für die Dialogfelder `msoFileDialogSaveAs` und `msoFileDialogFolderPicker` ändern und an eigene Bedürfnisse anpassen.

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-Objekts.

Listing 15.18 Auflistung aller aktiven Dateiauswahlfilter

```
Sub subListFileDialogOpenFilters()
    Dim strMSG As String
    Dim dlg As FileDialog
    Dim dlgFlt As FileDialogFilter
    Set dlg = Application.FileDialog(msoFileDialogOpen)
    For Each dlgFlt In dlg.Filters
        strMSG = strMSG & dlgFlt.Description & vbCrLf & "(" & dlgFlt.Extensions & ")" & _
            vbCrLf
    Next dlgFlt
    MsgBox strMSG, vbInformation, "Übersicht über die verfügbaren Dateiauswahlfilter " & _
        "des 'FileDialogOpen'-Dialogfeldes"
End Sub
```

Abbildg. 15.17 Übersicht über die Dateiauswahlfilter des *FileDialogOpen*-Objekts



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 15.19 werden zuerst alle Filter des msoFileDialogOpen-Dialogfelds 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 15.19 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"
    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.

Allow-
Multi-
Select

Zuerst müssen Sie beim Einsatz dieses Dialogfelds überlegen, ob Sie eine Mehrfachauswahl von Dateien erlauben möchten und auch weiterverarbeiten können.

Im Beispiel in Listing 15.20 wird die Mehrfachauswahl durch die Angabe der Eigenschaft erlaubt:

```
.AllowMultiSelect = True
```

Als nächsten 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, Textdateien 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

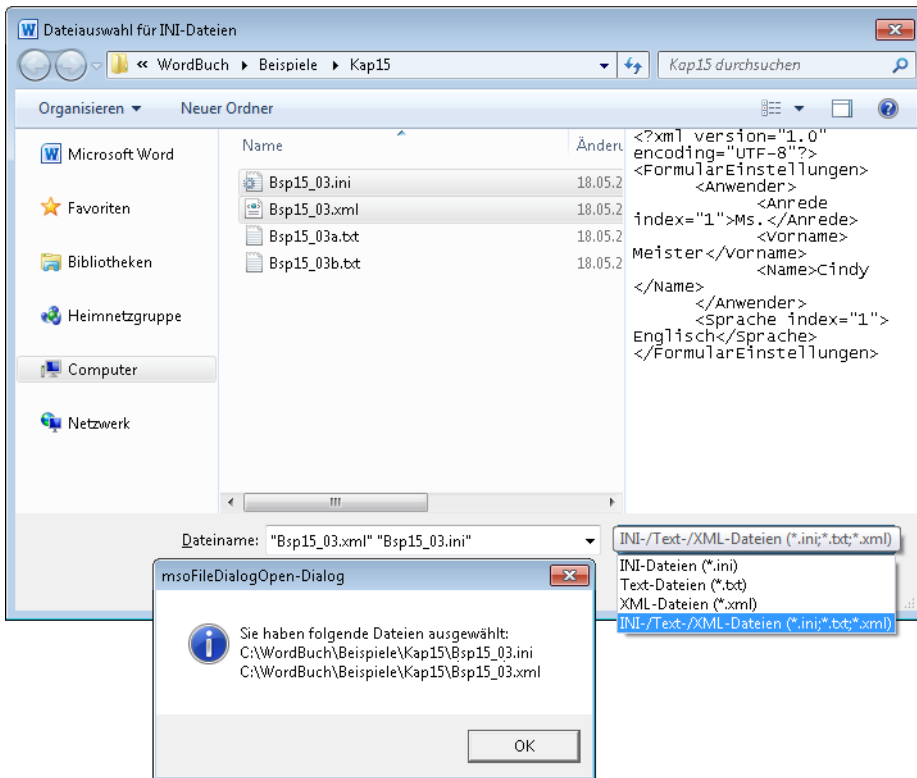
Die weiteren Eigenschaften des Dialogfelds legen den Titel und den Anzeigetyp der Dateiinformationen fest und ändern den Namen der ausführenden Aktionsschaltfläche *Öffnen* in *Dateien öffnen*.

Anschließend wird das Dialogfeld angezeigt. Nur wenn das Dialogfeld über die Aktionsschaltfläche beendet wird, der Rückgabewert des Dialogfelds »-1« ist, werden alle markierten Dateinamen inklusive Pfad gesammelt und ausgegeben.

Listing 15.20 Konfiguration des *msoFileDialogOpen*-Dialogfelds zur Auswahl von INI- und Textdateien

```
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, intFile As Integer
Set dlg = Application.FileDialog(msoFileDialogOpen)
fktStoreFilters dlg
With dlg
    .ButtonName = "Dateien öffnen"
    .Title = "Dateiauswahl für INI-Dateien"
    .InitialView = msoFileDialogViewDetails
    .AllowMultiSelect = True
    .InitialFileName = "C:\WordBuch\Beispiele\Kap15\"
    .Filters.Clear
    .Filters.Add "INI-Dateien", "*.ini"
    .Filters.Add "Text-Dateien", "*.txt"
    .Filters.Add "XML-Dateien", "*.xml"
    .Filters.Add "INI-/Text-/XML-Dateien", "*.ini;*.txt;*.xml"
    .FilterIndex = 4
    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. 15.18 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 15.21 wird neben der Festlegung des Titels und dem Ansichtstyp die Aktionsschaltfläche in *Datei speichern* geändert.

Damit der Dateiname des aktuellen Dokuments 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 Dateiendung ».doc« im Dateinamen vorkommt. Ist dies der Fall, wird diese durch die Endung .xml ersetzt, andernfalls wird die Endung an den Dateinamen angehängt.

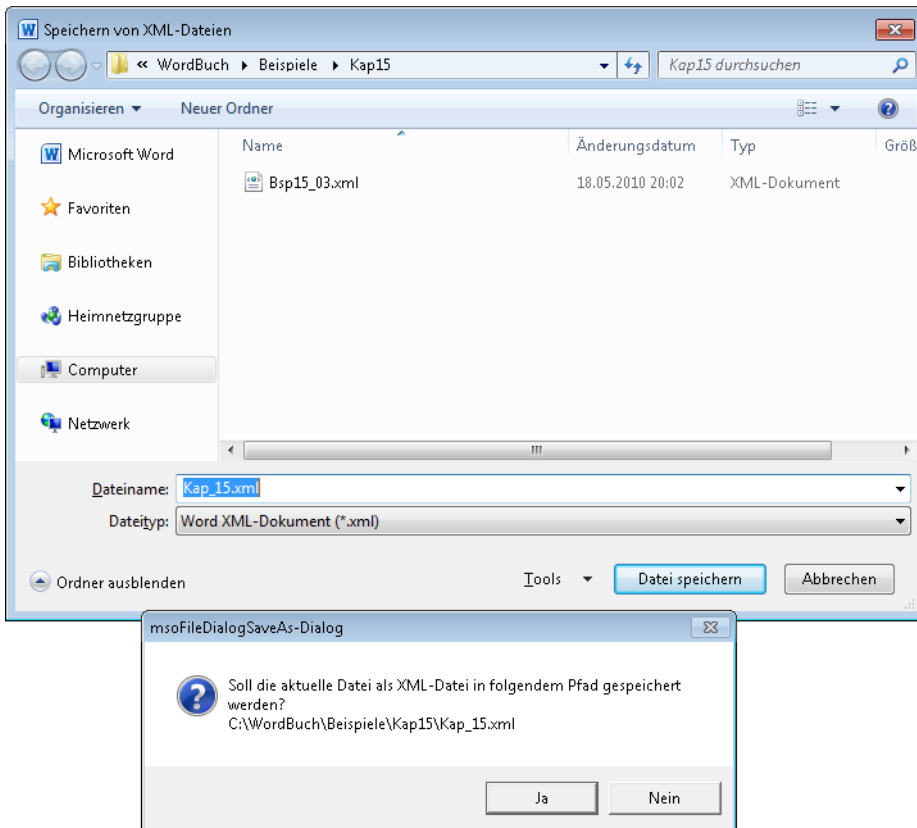
Im nächsten Schritt werden die vorhandenen Dateifilter dahingehend überprüft, ob ein geeigneter Filter für XML-Dateien vorhanden ist. Ist dies der Fall, wird er als verwendeter Dateityp festgelegt, andernfalls wird der Speicher-Vorgang beendet.

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 15.21 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:\WordBuch\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. 15.19 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`-Dialogfelds (siehe den Abschnitt »Anwendung des *FileDialog*-Typs *msoFileDialogOpen*« ab Seite 720), weshalb an dieser Stelle nicht weiter auf diesen `FileDialog`-Typ eingegangen wird.

Anwendung des *FileDialog*-Typs *msoFileDialogFolderPicker*

Das Dialogfeld `msoFileDialogFolderPicker` erlaubt nur die Auswahl eines Ordners. Wie das `msoFileDialogOpen`-Dialogfeld führt auch dieses keine Aktion aus, sondern liefert nur den ausgewählten Ordnersnamen inklusive Pfad über die `Show`-Methode zurück. Für dieses Dialogfeld können auch weder Filter festgelegt noch mehrere Ordner gleichzeitig ausgewählt werden.

Im Beispiel in Listing 15.22 wird nach Festlegung des Titels und Ansichtstyps die Aktionsschaltfläche in *Ordner einlesen* geändert. Denn nach Bestätigung der Auswahl über diese Schaltfläche werden in der Funktion `fktReadDirFiles` alle im ausgewählten Ordner enthaltenen Dateien ermittelt und mit ihren Dateiattributen, wie z.B. *Ordner* oder *Schreibgeschützt*, angezeigt.

Dir Dazu durchläuft die Funktion `fktReadDirFiles` mithilfe der `Dir`-Funktion (siehe Kapitel 2) alle Dateien des ausgewählten Ordners, ermittelt die Dateiattribute und speichert die Daten in einem Array. Dieses Array wird dann über den Funktionsnamen an die aufrufende Prozedur zurückgeliefert, wo die Einträge im Array ausgelesen und in einem Dialogfeld angezeigt werden.

Listing 15.22 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, strMSG 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:\WordBuch\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
```

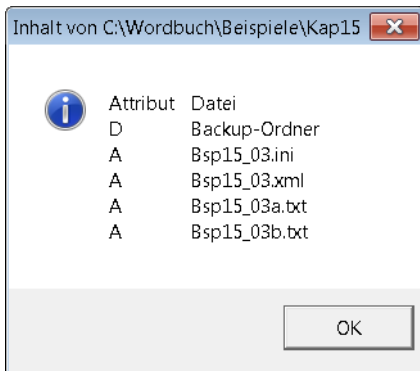

Listing 15.22 Auswahl eines Ordners und Anzeige aller im Ordner enthaltenen Dateien mit Attributen (Fortsetzung)

```

' Attribute in Kürzel umsetzen
If (GetAttr(strDir & strFile) And vbReadOnly) = vbReadOnly Then
    strAttr = strAttr & "R"
End If
If (GetAttr(strDir & strFile) And vbHidden) = vbHidden Then
    strAttr = strAttr & "H"
End If
If (GetAttr(strDir & strFile) And vbSystem) = vbSystem Then
    strAttr = strAttr & "S"
End If
If (GetAttr(strDir & strFile) And vbDirectory) = vbDirectory Then
    strAttr = strAttr & "D"
End If
If (GetAttr(strDir & strFile) And vbArchive) = vbArchive Then
    strAttr = strAttr & "A"
End If
If (GetAttr(strDir & strFile) And vbAlias) = vbAlias Then
    strAttr = strAttr & "L"
End If
strFiles(1, UBound(strFiles(), 2) - 1) = strAttr
End If
strFile = Dir    ' Nächsten Eintrag abrufen.
Loop
fktReadDirFiles = strFiles()
End Function

```

Abbildg. 15.20 Anzeige aller Dateien im ausgewählten Ordner mit ihren Dateiattributen



ACHTUNG Die Anzeige von versteckten Dateien oder Systemdateien hängt maßgeblich von den Einstellungen in Windows ab. So können Sie die Anzeige von Systemdateien oder versteckten Dateien und Ordnern für jeden Ordner in den jeweiligen Ordneroptionen einstellen.

CD-ROM Die dargestellten Beispiele finden Sie in der Beispieldatei *Bsp15_04.docm* 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 685 ff.)
- Es wurden die wichtigsten Eigenschaften (Seite 686 ff.), Methoden (Seite 689 ff.) und Ereignisse (Seite 691 ff.) des UserForm-Objekt vorgestellt
- Es wurden die allgemeinen Steuerelemente aus der »Microsoft Forms 2.0 Object Library« vorgestellt (Seite 695 ff.) und erklärt, wie zusätzliche Steuerelemente eingebunden werden können (Seite 696 ff.)
- Eine Zusammenfassung, welche Anforderungen der Anwender an ein Dialogfeld hat (Seite 696 ff.) und wie ein benutzerdefiniertes Dialogfeld zur Laufzeit beeinflusst werden kann (Seite 702 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 708 ff.) und über die zugehörigen Argumente konfigurieren können (Seite 712). Dabei wurde hervorgehoben, welche Unterschiede in den verschiedenen Aufrufmethoden bestehen.
- Der letzte Abschnitt stellte die verschiedenen FileDialog-Dialogfelder vor (Seite 715). Anhand von Beispielen wurden die Einstellmöglichkeiten und Einschränkungen aufgezeigt (Seite 720 ff.).

Kapitel 16

Die Office Fluent UI

In diesem Kapitel:

Was ist das Ribbon?	728
Einführung in die Menüband-Erweiterung	732
Erweiterte Funktionalität	738
Die Steuerelemente	755
Kontextmenüs definieren	771
Backstage	774
Zusammenfassung	777

In Office 2007 hatte das Microsoft Office-Team eine neue Benutzerschnittstelle eingeführt. Sie ersetzte die bisher bekannten Menüs und Symbolleisten. In der deutschen Sprache wurde sie »Multifunktionsleiste« getauft; in Office 2010 wurde sie in »Menüband« umbenannt. In Englisch, sowie in der Entwicklerdokumentation, hieß sie zuerst »Ribbon« oder auch »RibbonX« (das »X« steht für Extensibility, was so viel wie Erweiterung bedeutet). Bald jedoch wurde der Begriff »Office Fluent UI« eingeführt, was in etwa »fließende Benutzerschnittstelle« bedeutet. Damit kündigte Microsoft weitere Änderungen an, die nicht ausgeblieben sind.

Diese wurden in Office 2010 realisiert. Zum Menüband gesellt sich die Backstage-Ansicht. Diese ist auf ähnliche Art und Weise anpassbar wie das Ribbon. Ebenso können die Kontextmenüs in Office 2010 in der Menüband-XML-Datei definiert werden.

In diesem Kapitel werden wir zuerst und hauptsächlich das Menüband unter die Lupe nehmen. Nach einer kurzen Einführung der Grundlagen wird die erweiterte Funktionalität präsentiert, gefolgt von einem Überblick der verschiedenen zur Verfügung stehenden Steuerelemente.

Am Schluss des Kapitels wird die Anpassung der Backstage-Ansicht und der Kontextmenüs kurz vorgestellt. Diese ist auf den gleichen XML-Prinzipien wie das Menüband aufgebaut und in der gleichen Datei enthalten.

Die Seitenzahl für dieses Buch ist leider beschränkt und so können wir uns hier mit dieser neuen und spannenden Technologie nicht in ihrem vollen Umfang befassen. Dieses Thema füllt selber ein ganzes Buch. Mit den Angaben in diesem Kapitel werden Ihnen die nötigen Puzzle-Teile in die Hände gegeben, selbstdefinierte Benutzerschnittstellen in Word 2007 und Word 2010 zu erstellen.

HINWEIS

Eine ausführliche Behandlung der Menüband-Funktionalität füllt ein ganzes Buch, wie beispielsweise das Nachschlagwerk »Menüband-Programmierung für Office 2007«, erschienen bei Addison-Wesley (ISBN-13: 978-3-8273-2738-3). Hier in unserem Buch werden ausschließlich die Grundlagen für einen guten Anfang gelegt.

Was ist das Ribbon?

Im Gegensatz zu den Symbolleisten bleibt das Menüband statisch am oberen Fensterrand haften und kann nicht verschoben werden. In Office 2007 konnte der Benutzer diese innerhalb der Anwendungsoberfläche nicht anpassen (ihm steht jedoch eine andere, kleinere Leiste zur Verfügung, die im Abschnitt »Die neue Terminologie« ab Seite 730 kurz vorgestellt wird).



Offensichtlich war der Aufschrei über die fehlenden Anpassungsmöglichkeiten vom Entwicklerteam in Redmond gehört worden: Das Dialogfeld *Optionen* des Office 2010 wurde um den Menüpunkt *Menüband anpassen* erweitert. Der Benutzer kann nun im Menüband eigene Registerkarten anlegen. Diesen Registerkarten können Word-eigene oder benutzerdefinierte Gruppen hinzugefügt werden. Die Gruppen werden anschließend mit Schaltflächen bestückt. Diese Anpassungen werden zentral gespeichert und stehen auf Anwendungsebene zur Verfügung. Mehr darüber lesen Sie im Kapitel 1.

Statt über ein Objektmodell wie *CommandBars* wird der Inhalt des Menübands 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 Menüband-Elemente ansprechen, haben die Anpassungen der zuletzt geladenen Vorrang.

Alle Steuerelemente müssen bereits in der XML-Datei definiert sein. Es ist nicht möglich, zur Laufzeit Steuerelemente zu erstellen oder diese zu löschen. Diese können jedoch durch eigenen Code

ein- und ausgeblendet oder gesperrt werden. Andere Eigenschaften sind ebenfalls dynamisch anpassbar. Außenstehender Code hat keinen Zugriff auf die Steuerelemente.

Einige Überlegungen stehen hinter der Entscheidung, diese neue Benutzerschnittstelle einzuführen. Dazu gehören unter anderen:

- Der Vielfalt an Funktionalität in Office-Anwendungen wie Word ist inzwischen so reichhaltig geworden, dass dem Benutzer viele Möglichkeiten entgangen sind. Sie sind im Menüband schneller auffindbar.
- Immer häufiger störten COM-Add-Ins von Drittanbietern den Benutzer bei seiner Arbeit: Sie blendeten die Symbolleistenanpassungen anderer (inklusive diejenigen des Benutzers) aus oder entfernten ihre Befehle aus den Menüs. Dazu kommen die Speicheraufforderungen für die Vorlage *Normal.dotm*, die nicht nur störend sind, sondern den Benutzer auch verunsichern. Die Arbeitsweise des Menübands 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 Menübands 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 Menüband-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 Menüband-Erweiterung ist jedoch nicht besonders kompliziert. Kenntnisse von XSLT, XPath oder XMLDOM sind nicht erforderlich.

Obwohl der Entwurf einer Menüband-Erweiterung etwas mühsam ist, da es nur in Visual Studio .NET Professional einen grafischen Entwurfsmodus gibt, ermöglicht sie 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 an den Symbolleisten sind in dieser Beziehung ebenfalls kein Thema mehr.

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:

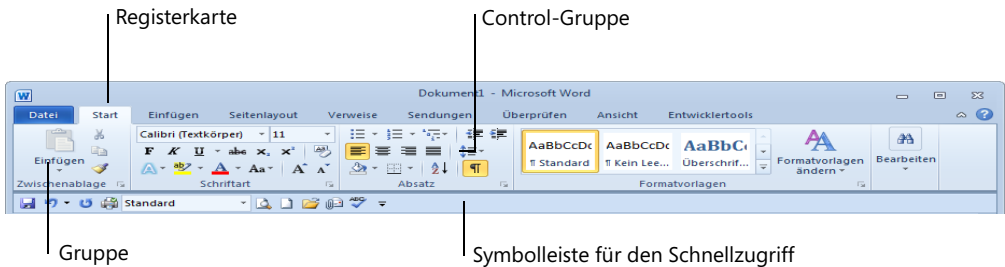
- **Toolbar Toggle** Ermöglicht die Erstellung von eigenen »Symbolleisten«. Diese können überall auf dem Bildschirm positioniert werden und unterstützen sowohl Word-eigene als auch Makrobefehle. Dieses Werkzeug ist erhältlich unter <http://www.toolbartoggle.com>.

- **RibbonCustomizer** Unterstützt ausschließlich Anpassungen im Menüband. Auch dieses Werkzeug unterstützt Word-eigene sowie Makrobefehle. Es ist in einer deutschen Version erhältlich unter <http://pschmid.net/office2007/ribboncustomizer/indexGerman.php>.

Die neue Terminologie

Die neue Schnittstelle bringt auch neue Bezeichnungen mit sich. Einige werden in Abbildung 16.1 vorgestellt.

Abbildg. 16.1 Das Menüband als Benutzerschnittstelle



- **Ribbon** Der »Behälter« aller übrigen Elemente. Es umfasst das, was wir früher als die Menüleiste kannten, sowie den Teil darunter. Der untere Teil kann über die rechts im Menüband befindliche Schaltfläche *Menüband 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 Menü, sowie die dazu gehörenden Befehle. In Abbildung 16.1 ist die Registerkarte *Start* aktiviert.
- **Gruppe** Die viereckigen Felder innerhalb einer Registerkarte, welche die Befehle gruppieren. Jede Gruppe ist an ihrem 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
- **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 des Menübands und Auswahl des Eintrags *Zu Symbolleiste für den Schnellzugriff hinzufügen* kann ein Befehl der QAT zugefügt werden. Sie kann ebenfalls über die Anwendungsoptionen in der Registerkarte *Symbolleiste für den Schnellzugriff* 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 dem Menüband positioniert werden, wie dies Abbildung 16.1 veranschaulicht.

HINWEIS Mehr über die Anpassung der Symbolleiste für den Schnellzugriff steht im Kapitel 1 beschrieben.

Die Rolle von *CommandBars*

Das *CommandBars*-Objektmodell existiert noch, hat aber an Bedeutung verloren. Wichtig bleibt es in Office 2007 für die Arbeit mit Kontextmenüs; diese können ab Office 2010 auch über Menüband-XML definiert werden. Die über den früheren Menübefehl *Extras/Anpassen/Symbolleiste* erreichbare Schnittstelle wurde ersatzlos gestrichen, sodass Kontextmenüs nur programmtechnisch erstellt und angepasst werden können.

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.

Neue *CommandBars*-Methoden für das Ribbon

Obwohl das *CommandBars*-Objektmodell von verschwindender Bedeutung ist, wurde es gleichwohl in Bezug auf das Menüband um einige wichtigen Methoden erweitert. Diese sind in Tabelle 16.1 aufgelistet und ermöglichen die Ausführung der anwendungseigenen Befehle sowie die Abfrage gewisser Eigenschaften anwendungseigner Steuerelemente.

HINWEIS Mehr zum Thema *idMso* erfahren Sie im Abschnitt »Word-eigene Schaltflächen benutzen« ab Seite 740.

Tabelle 16.1 Neue Methoden des *CommandBars*-Objekts

Methode	Beschreibung	Datentyp des Rückgabewerts
<code>ExecuteMso(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>GetScreenTipMso(idMso)</code>	Gibt den Screentip (QuickInfo) zurück	Zeichenkette
<code>GetSuperTipMso(idMso)</code>	Gibt den Text des Supertips zurück	Zeichenkette
<code>GetVisibleMso(idMso)</code>	Ermittelt, ob das Steuerelement sichtbar ist	Boolean

Einführung in die Menüband-Erweiterung

Wie eingangs erwähnt, wird das Menüband 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 21.

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 in Office keinen grafischen Entwurfsmodus für die Menüband-Erweiterung.

Jeder Teil der Menüband-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 bedingt. Von bedeutender Hilfe ist ein geeignetes Werkzeug, um die XML-Datei zu erstellen. Wer sich weniger mit dem Codeschreiben 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 21 im Abschnitt »XML-Namensräume und Schemas« aufgeführt. Sie übernehmen rein administrative Arbeiten wie das Einfügen von Einzügen, Abschlusselementetags und Anführungszeichen. Manche können das XML mit einem Schema validieren.

Microsoft Werkzeugen für die Arbeit mit XML

Visual Studio .NET – auch die kostenlosen Express-Ausgaben zum Herunterladen, stellen XML-Werkzeuge zur Verfügung, die mit dieser Arbeit helfen können. Wird das passende Schema geladen, bietet der XML-Editor für Menüband-XML IntelliSense an. Mehr Informationen erfahren Sie im VSTO-Abschnitt über Add-Ins des Kapitel 10.

Ab Visual Studio 2008 bietet die Ausgabe Professional (und höher) in den Visual Studio Tools for Office einen Entwurfsmodus für das Menüband. Diesen haben wir im VSTO-Abschnitt zum Thema Dokumentlösungen des Kapitels 10 vorgestellt. Da das Ergebnis in XML konvertiert werden kann, ist er auch für den VBA-Dokumententwickler vom Nutzen.

Sobald sichergestellt ist, dass das XML fehlerfrei ist, kann die Datei wie im Abschnitt »Die Menüband-Erweiterung in das Dokument einbinden« ab Seite 736 beschrieben, problemlos in ein Word-Dokument eingebunden werden.

Gute Dienste leistet auch das »Custom UI Editor Tool«, welches kostenlos unter <http://openxmldeveloper.org/articles/CustomUIEditor.aspx> heruntergeladen werden kann. Obwohl das Tool kein Intelli-

Sense anbietet, prüft es die Wohlgeformtheit sowie Gültigkeit des XML-Codes mit dem Menüband-Schema *customUI.xml* und weist auf mögliche Fehler hin. Zudem bindet es Grafikdateien für die Schaltflächen in das Zieldokument ein. Schließlich erstellt es alle benötigten Einträge für die »Relationships« im *_rels*-Ordner. Dieses Werkzeug setzt die Installation von .NET Framework in der Version 2.0 oder 3.0 voraus, das (ebenfalls kostenlos) von der Microsoft-Website heruntergeladen werden kann. Dieses Werkzeug wird im Abschnitt »Grafikdateien einbinden« ab Seite 744 näher vorgestellt.

HINWEIS .NET Framework ist nicht mit Visual Studio .NET zu verwechseln. Visual Studio .NET ist eine Programmierumgebung für .NET Framework, der Laufzeitumgebung für .NET-Anwendungen. Links zu den verschiedenen Versionen von .NET Framework befinden sich auf der Webseite <http://www.microsoft.com/downloads/Browse.aspx?displaylang=de&categoryid=1>. Achten Sie darauf, dass Sie die richtige Version für Ihren Rechner auswählen.

CD-ROM Die erwähnten Dateien und andere Hilfsmittel zum Thema finden Sie im Ordner *\Beilagen\Ribbon* auf der CD-ROM zum Buch.

Basiswissen

Wir fangen mit einem sehr einfachen Beispiel an, um die einzelnen Schritte der Menüband-Erweiterung zu veranschaulichen. Als Ziel streben wir das in Abbildung 16.2 ersichtliche Ergebnis an: Eine Menüband-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. 16.2 Das Ergebnis der in diesem Abschnitt beschriebenen Menüband-Anpassung



Das Dokument vorbereiten

Eine Menüband-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 Menüband-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-Proze-`

duren und den passenden Argumenten deklariert werden. Die Prozedur für eine Schaltfläche erfordert nur einen Parameter des Datentyps `Office.IRibbonControl`, wie dies in Listing 16.1 veranschaulicht wird.

Anschließend wird das Dokument geschlossen und gespeichert.

Listing 16.1 Die von der Menüband-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 Menüband-Erweiterung wird in einer getrennten Datei erstellt, die anschließend in das Word-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 wählen Sie im Kontextmenü den Eintrag *Neu/Ordner*.
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 Windows-Editor und geben Sie den XML-Code aus Listing 16.2 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 16.2 Eine sehr einfache benutzerdefinierte Erweiterung für das Word 2010 Menüband bzw. Word 2007 Multifunktionsleiste

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui" >
<!-- Hier die Syntax für die Office 2007-Multifunktionsleiste -->
<!-- < customUI xmlns=http://schemas.microsoft.com/office/2006/01/customui > -->
  <ribbon>
    <tabs>
      <tab id="MeinTab" label="Mein Tab" >
```

Listing 16.2 Eine sehr einfache benutzerdefinierte Erweiterung für das Word 2010 Menüband bzw. Word 2007 Multifunktionsleiste (Fortsetzung)

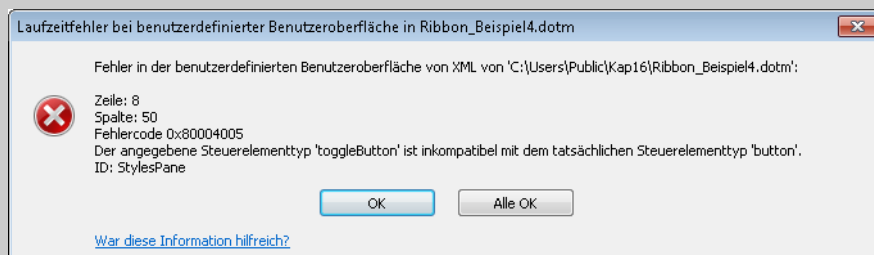
```
<group id="MeineGruppe" label="Meine Gruppe" >
  <button id="MeinButton1" label="Große Schaltfläche" size="large"
    onAction="MeinButton1_Click" />
  <button id="MeinButton2" label="Kleine Schaltfläche" size="normal"
    onAction="MeinButton2_Click" />
</group>
</tab>
</tabs>
</ribbon>
</customUI>
```

PROFITIPP

Es ist empfehlenswert, das Kontrollkästchen *Fehler des Benutzeroberflächen-Add-Ins anzeigen* in der Kategorie *Erweitert* der *Optionen* im Abschnitt *Allgemein* zu aktivieren. Ansonsten werden eventuell durch das Menüband-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.

Ein Beispiel ist in Abbildung 16.3 ersichtlich. Die Fehlermeldung berichtet, wo im XML das Problem liegt und was beanstandet wird. In diesem Fall wurde eine Word-eigene Schaltfläche angesprochen, aber der falsche Typ angegeben. Im XML muss ein `<toggleButton>`- statt eines `<toggleButton>`-Elements stehen.

Abbildg. 16.3 Eine Fehlermeldung erscheint, wenn das Menüband-XML nicht korrekt geschrieben wurde



Speichern Sie die Datei mit dem Namen *customUI14.xml* (für Office 2007 *customUI.xml*) in den 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.

HINWEIS

Achten Sie auf die unterschiedliche Syntax in der ersten Zeile des XML für Office 2007 und Office 2010. Da die Menüband-Funktionalität im Office 2010 um einige Neuigkeiten erweitert wurde, liegt dieser Version ein anderes Schema zugrunde. Office 2010 arbeitet problemlos mit für Office 2007 definierten Ribbons. Nur können diese die neue Funktionalität nicht anbieten. Wollen Sie ein Dokument mit einer Menüband-Anpassung bereitstellen, das sowohl in Word 2007 als auch in Word 2010 zur Verfügung steht, muss es mit dem Schema für Office 2007 definiert werden.

Ein Dokument darf auch zwei Menüband-XML-Dateien enthalten, eine für Office 2007 sowie eine für Office 2010. Word wird je nach Version die korrekte Anpassung laden.

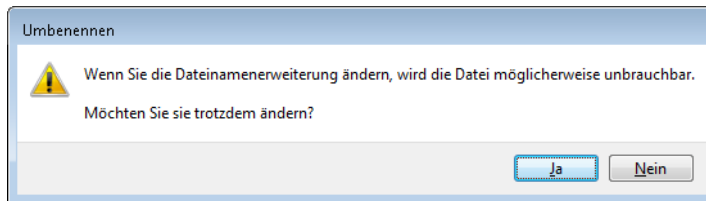
Die Menüband-Erweiterung in das Dokument einbinden

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 16.4, ob Sie dies wirklich wollen. (Der Dokumentname wird mit der Endung ».zip« ergänzt.)

Abbildg. 16.4

Sicherheitsabfrage, die bei Umbenennung einer Dateierweiterung erscheint



3. Doppelklicken Sie unter Windows Vista oder Windows 7 auf den Dokumentnamen (bei Windows XP wählen Sie im Kontextmenü den Untermenübefehl *Öffnen mit/ZIP-komprimierte Ordner*).
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 16.5).
6. Klicken Sie mit der rechten Maustaste auf die Datei *.rels* und wählen Sie im Kontextmenü den Untermenübefehl *Öffnen mit/Editor* (siehe Abbildung 16.5). (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 16.5).

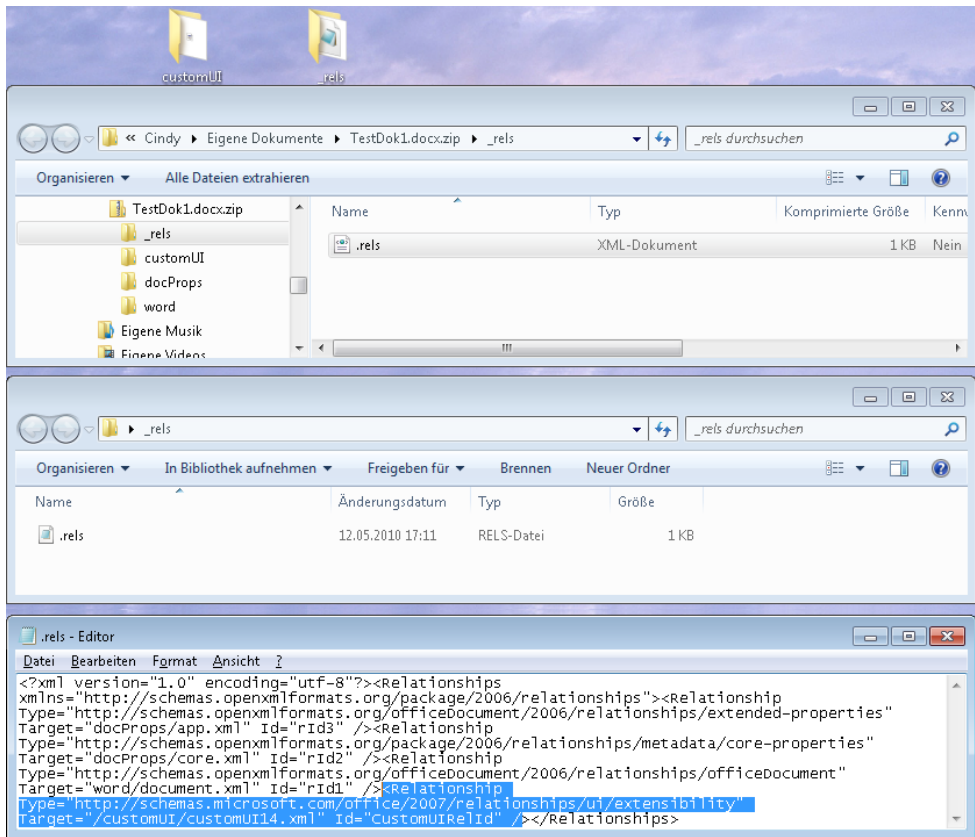
Office 2010:

```
<Relationship Type="http://schemas.microsoft.com/office/2007/relationships/ui/
extensibility" Target="/customUI/customUI14.xml" Id="customUIRelID" />
```

Office 2007:

```
<Relationship Type="http://schemas.microsoft.com/office/2006/relationships/ui/
extensibility" Target="/customUI/customUI.xml" Id="customUIRelID" />
```

Abbildg. 16.5

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 linker 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 Dokumentdatei 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* im Menüband (siehe Abbildung 16.2 auf Seite 733).

CD-ROM

Den Code zum aktuellen Abschnitt finden Sie in den Beispieldateien *Beispiel1_customUI.xml* sowie *Ribbon_Beispiel1.dotm*. Diese befindet sich auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap16*.

Erweiterte Funktionalität

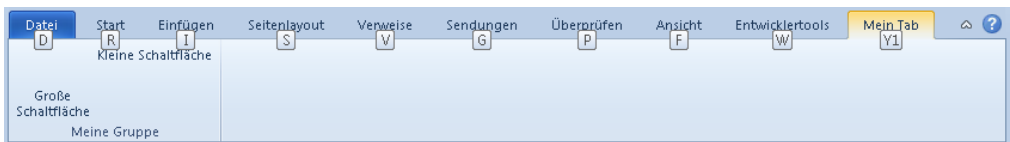
Im vorangegangenen Abschnitt wurden die Grundlagen zur Anpassung der Menüband-Schnittstelle vorgestellt. Ziel dieses Abschnitts ist es nun, anhand weiterer Beispiele einen vertieften Einblick in die Erweiterungsmöglichkeiten zu vermitteln.

Befehle mit Tastenkombinationen verbinden

Wenn der Benutzer die **[Alt]**-Taste drückt, erscheinen neben jeder Registerkarte ein oder mehrere Buchstaben. Durch Drücken dieser Buchstabenkombination wird die Registerkarte ausgewählt (Abbildung 16.6) und die Keytips seiner Steuerelemente eingeblendet. Um einen Keytip für die eigenen Registerkarten, Gruppen und Steuerelemente festzulegen, fügen Sie dem XML-Element das Attribut `keytip` zu:

```
<tab id="MeinTab" label="Mein Tab" keytip="M" >
```

Abbildg. 16.6 Tastenkombinationen im Menüband



Die Benutzerhandlung kann mittels der Methode `SendKeys` programmtechnisch ausgeführt werden. Diese Methode ist allerdings nicht besonders zuverlässig, denn es ist nicht sichergestellt, dass die gewünschte Applikation oder das richtige Fenster das Aktive ist. Zudem ändert Word die Tastenfolgen dynamisch, falls zwei Steuerelementen der gleiche `keytip` zugewiesen wurde.

Die Syntax dazu lautet:

```
SendKeys "%M"
```

CD-ROM

Das XML-Beispiel befindet sich in der Datei *Beispiel2_customUI.xml*. Sie finden es im Ordner *\Beispiele\Kap16* auf der CD-ROM zum Buch zusammen mit dem Beispieldokument *Ribbon_Beispiel2.dotm*.

Eine Registerkarte an eine beliebige Stelle positionieren

Standardmäßig erscheinen neu definierte Registerkarten am Ende des Menübands, in der Reihenfolge ihres Erscheinens in der Datei *customUI.xml*. Registerkarten können aber in beliebiger Reihenfolge aufgelistet werden. Dafür zuständig sind die Attribute `insertBeforeQ`, `insertBeforeMso`, `insertAfterQ` sowie `insertAfterMso`.

Die Eigenschaften `insertBeforeMso` und `insertAfterMso` legen die Position vor bzw. nach der angegebenen Word-eigenen Registerkarte fest.

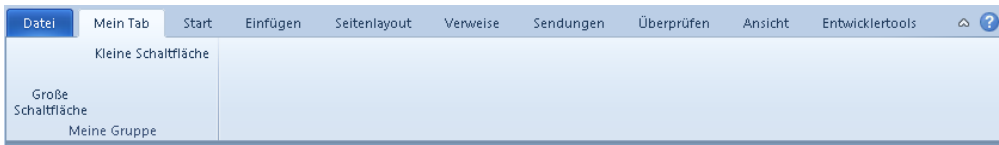
Mit `insertBeforeQ` und `insertAfterQ` kann die Reihenfolge der durch mehrere Add-Ins im gleichen Namensraum definierten Registerkarten festgelegt werden. Dieses Attribut erwartet den qualifizierten ID-Wert der Registerkarte in Form einer Zeichenkette.

HINWEIS Mehr über Namensräume in Menüband-Erweiterungen erfahren Sie im Abschnitt »Menüband-Registerkarten teilen« ab Seite 751.

Diese Attribute finden vor allem Gebrauch bei der Festlegung der standardmäßigen Registerkarte beim Öffnen eines Dokuments. Beim Öffnen eines Dokuments wird immer die erste Registerkarte standardmäßig angezeigt. Die Reihenfolge der Registerkarten kann so angepasst werden, dass die gewünschte ganz links an erster Stelle steht. Folglich wird sie beim Öffnen des Dokuments automatisch angewählt.

```
<tab id="MeinTab" label="Mein Tab" keytip="M" insertBeforeMso="TabHome" >
```

Abbildg. 16.7 Beim Öffnen eines Dokuments wird die erste Registerkarte angezeigt



CD-ROM Das XML-Beispiel befindet sich in der Datei *Beispiel3_customUI.xml*. Sie finden es im Ordner *\Beispiele\Kap16* auf der CD-ROM zum Buch zusammen mit dem Beispieldokument *Ribbon_Beispiel3.dotm*.

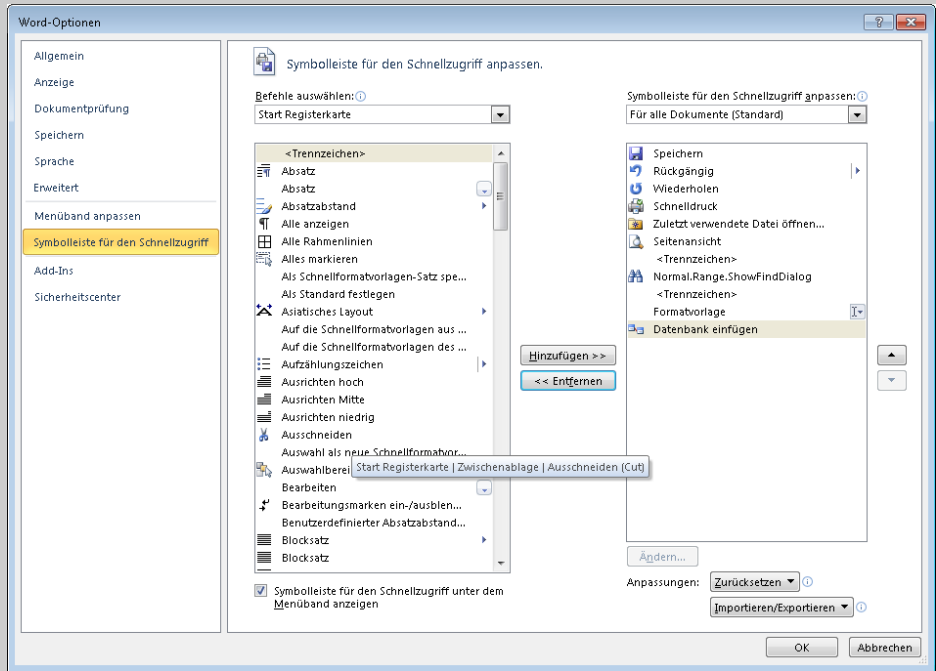
idMso-Werte

Der `idMso`-Wert eines Office-Steuerelements identifiziert dieses eindeutig. Es handelt sich um eine Zeichenkette, die ungefähr dem Namen des Befehls – in englischer Sprache – entspricht.

Eine echte Herausforderung ist das Herausfinden des passenden Werts: Wie lautet wohl die »gute« Zeichenkette für das Attribut `idMso`? Dazu 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 16.8).

Abbildg. 16.8 Den Wert für das Attribut *idMso* herausfinden



Zudem stellt Microsoft eine Liste der *idMso*-Werte in Form einer Excel-Arbeitsmappe zur Verfügung. Für Office 2007 unter <http://www.microsoft.com/downloads/details.aspx?familyid=4329d9e9-4d11-46a5-898d-23e4f331e9ae&displaylang=en#filelist>, bzw. für Office 2010 unter <http://www.microsoft.com/downloads/details.aspx?FamilyID=3f2fe784-610e-4bf1-8143-41e481993ac6&displaylang=en>. Die Werkzeuge *2007OfficeControlIDsExcel2003.exe* sowie *Office-2010ControlIDs.exe* befinden sich auf der CD-ROM im Ordner *Beilagen/Ribbon*.

Word-eigene Schaltflächen benutzen

Befehle, die dem Menüband einer Anwendung zur Verfügung stehen, können auch in benutzerdefinierten Menüband-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 16.8, wo die Word-eigenen Befehle *DateiNeu*, *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 die Ein- bzw. Ausblendung des Aufgabenbereichs *Formatvorlagen* in die Menüband-Erweiterung ein.


```
<button idMso="StylesPane" size="large" />
```

CD-ROM Das XML-Beispiel befindet sich in der Datei *Beispiel4_customUI.xml*. Sie finden es im Ordner *\Beispiele\Kap16* auf der CD-ROM zum Buch zusammen mit dem Beispieldokument *Ribbon_Beispiel4.dotm*.

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 19). Die Menüband-Erweiterung bietet diese Funktionalität ebenfalls an (was sich auch auf COM-Add-Ins ausdehnt) und ersetzt sukzessiv die alte Methode der Übersteuerung.

Die abzufangenden Befehle werden unter dem fakultativen Element `<commands>` unmittelbar vor dem Element `<ribbon>` aufgelistet. Die folgenden Codezeilen übersteuern den Befehl *Datei/Speichern*:

```
<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" />
</customUI>
```

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. Die 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 16.3).

Listing 16.3 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 *Datei*- bzw. *Office*-Schaltfläche außer Kraft gesetzt werden. Diese Schaltfläche selbst kann nicht aus dem Menüband entfernt werden. Zudem ist es auch nicht möglich, die beiden Befehle *Optionen* sowie *Beenden* außer Kraft zu setzen.

CD-ROM

Das XML-Beispiel befindet sich in der Datei *Beispiel5_customUI.xml*. Sie finden es im Ordner *\Beispiele\Kap16* auf der CD-ROM zum Buch zusammen mit dem Beispieldokument *Ribbon_Beispiel5.dotm*.

Word-eigene Registerkarten außer Kraft setzen

Es ist auch möglich, eine ganze Registerkarte zu sperren, indem sie unsichtbar gemacht wird. Um dies zu tun, wird die Registerkarte im Menüband-XML aufgeführt und deren Attribut *visible* auf Falsch gesetzt (hier beispielsweise die Registerkarte *Sendungen*).

```
<tab idMso="TabMailings" visible="false" />
```

CD-ROM

Das XML-Beispiel befindet sich in der Datei *Beispiel6_customUI.xml*. Sie finden es im Ordner *\Beispiele\Kap16* auf der CD-ROM zum Buch zusammen mit dem Beispieldokument *Ribbon_Beispiel6.dotm*.

QuickInfo für Menüband-Steuererelemente

Wird der Mauszeiger über eine Schaltfläche im Menüband positioniert, erscheint eine hellgelbe QuickInfo (Abbildung 16.9), die Informationen über den Befehl anzeigt. Diese QuickInfo besteht aus einem oberen und einem unteren Teil.

In der Menüband-Erweiterung werden diese mit den Attributen *screentip* bzw. *supertip* festgelegt:

```
<button id="MeinButton2" label="Kleine Schaltfläche" size="normal"
onAction="MeinButton2_Click" screentip="Meine kleine Schaltfläche"
supertip="Der Supertip für meine kleine Schaltfläche" />
```

Der *screentip* bildet den oberen Teil (die Überschrift) der QuickInfo. Der *supertip* erscheint im mittleren Teil der QuickInfo und dient als Hilfetext für den Befehl.

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 16.9). 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 Dritt-anbietern, 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 **F1** angebotene Hilfe enthält entsprechende Angaben.

Abbildg. 16.9 QuickInfo mit *screentip* und *supertip*

CD-ROM Das XML-Beispiel befindet sich in der Datei *Beispiel7_customUI.xml*. Sie finden es im Ordner *\Beispiele\Kap16* auf der CD-ROM zum Buch zusammen mit dem Beispieldokument *Ribbon_Beispiel7.dotm*.

Grafiken in die Menüband-Erweiterung einbinden

Grafiken helfen dem Benutzer, Befehle visuell schnell zu erkennen. Das kleine Beispiel, das uns bis hierher begleitet, sieht etwas fad, ja sogar nicht ganz fertig aus. Vor allem, wenn wir Abbildung 16.9 mit Abbildung 16.10 vergleichen.

Abbildg. 16.10 Schaltflächen mit Grafiken ergänzen



Integrierte Office-Grafiken verwenden

Die Office-Anwendungen enthalten viele Grafiken für Schaltflächen. Die meisten werden irgendwo in einer Anwendung benutzt, andere stehen explizit für den eigenen Gebrauch zur Verfügung. Diese sind im *Datei/Optionen/Symbolleiste für den Schnelzugriff/Ändern* ersichtlich (im Kapitel 1 abgebildet).

Alle Grafiken, die Office-eigenen sowie diejenigen für den eigenen Gebrauch, stehen für Menüband-Erweiterungen zur Verfügung. Um eine Grafik in das Menüband einzubinden, wird die Bezeichnung für das Attribut `imageMso` angegeben, wie in Listing 16.4 ersichtlich.

HINWEIS Sie können eine Liste vieler der zur Verfügung stehenden integrierten Grafiken unter <http://www.microsoft.com/downloads/details.aspx?FamilyID=12b99325-93e8-4ed4-8385-74d0f7661318&displaylang=en> herunterladen. Da diese Datei für eine Betaversion von Office 2007 erstellt wurde, können beim Öffnen der Excel-Mappe einige Meldungen erscheinen. Bestätigen Sie diese Meldungen und wechseln anschließend zur Registerkarte *Entwicklertools*. Sie finden die Auflistungen der Grafiken rechts in der Registerkarte.

Grafikdateien einbinden

Die Menüband-XML-Technologie arbeitet vorzugsweise mit Grafiken in PNG-Format.

Für Menüband-Erweiterungen, die im Dokument gespeichert sind, können die dazugehörigen Grafiken auch im Dokument gespeichert oder dynamisch aus eigenständigen Dateien geladen werden.

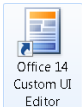
Wie der Abschnitt »Die Menüband-Erweiterung in das Dokument einbinden« ab Seite 736 veranschaulicht, erhalten Sie Zugang zum Inhalt eines Dokuments über den Windows-Explorer, wenn Windows dieses als ZIP-Datei betrachtet. Grafiken für die Menüband-Erweiterung werden im Ordner *customUI/images* gespeichert. Der Verweis auf die Grafik wird in die Datei *customUI/_rels/customUI.xml.rels* geschrieben:

```
<?xml version="1.0" encoding="utf-8"?>
<Relationships xmlns="http://schemas.openxmlformats.org/package/2006/relationships">
  <Relationship
    Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/image"
    Target="images/Eightball.png" Id="Eightball" />
</Relationships>
```

Die im Dokument gespeicherte Grafik wird über das Attribut *Image* mit einem Steuerelement verbunden (Listing 16.4). Dazu benötigen Sie den im Element *Relationship* angegebenen Id-Wert und *nicht* den Dateinamen.

Die manuelle Erstellung einer Menüband-Erweiterung über die ZIP-Datei ist mühsam und fehleranfällig. Ein sehr hilfreiches Werkzeug ist der kostenlose Custom UI Editor, den Sie bei <http://openxmldeveloper.org/articles/CustomUIeditor.aspx> herunterladen können.

Mit dem Custom UI Editor eine Menüband-Erweiterung erstellen



Bereiten Sie das Dokument oder die Dokumentvorlage wie im Abschnitt »Die Menüband-Erweiterung in das Dokument einbinden« ab Seite 736 beschrieben vor. Stehen alle Makros für *onAction*-Attribute bereit, gehen Sie wie folgt vor:

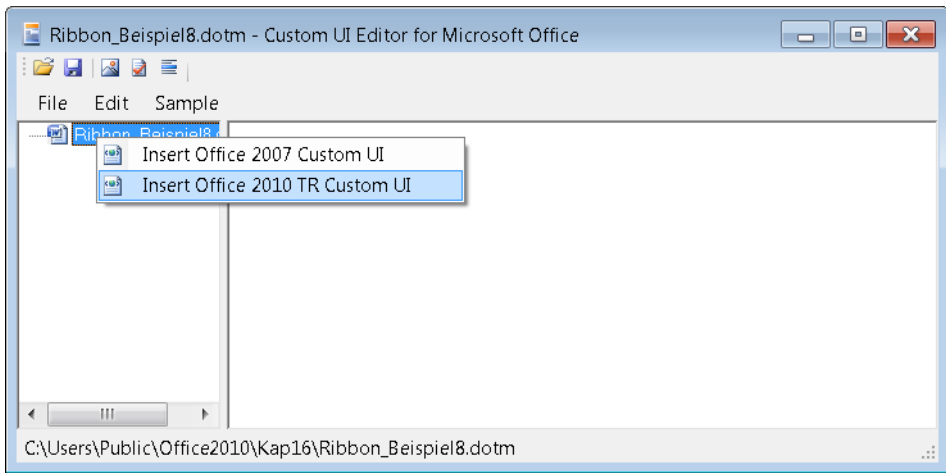
1. Starten Sie den Custom UI Editor.
2. Öffnen Sie über *File/Open* die Word-Datei. Der Dokumentname erscheint im Bereich links.

ACHTUNG

Ein Dokument kann in den Custom UI Editor nicht geöffnet werden, wenn dieses 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.

3. Klicken Sie mit der rechten Maustaste auf den Dateinamen und wählen Sie die Art der Menüband-Erweiterung (Office 2007 bzw. Office 2010) aus, wie in Abbildung 16.11 ersichtlich. (Eine XML-Datei wird unter dem Dokumentnamen erscheinen.)

Abbildg. 16.11 Einem Word-Dokument eine Menüband-Erweiterung zufügen



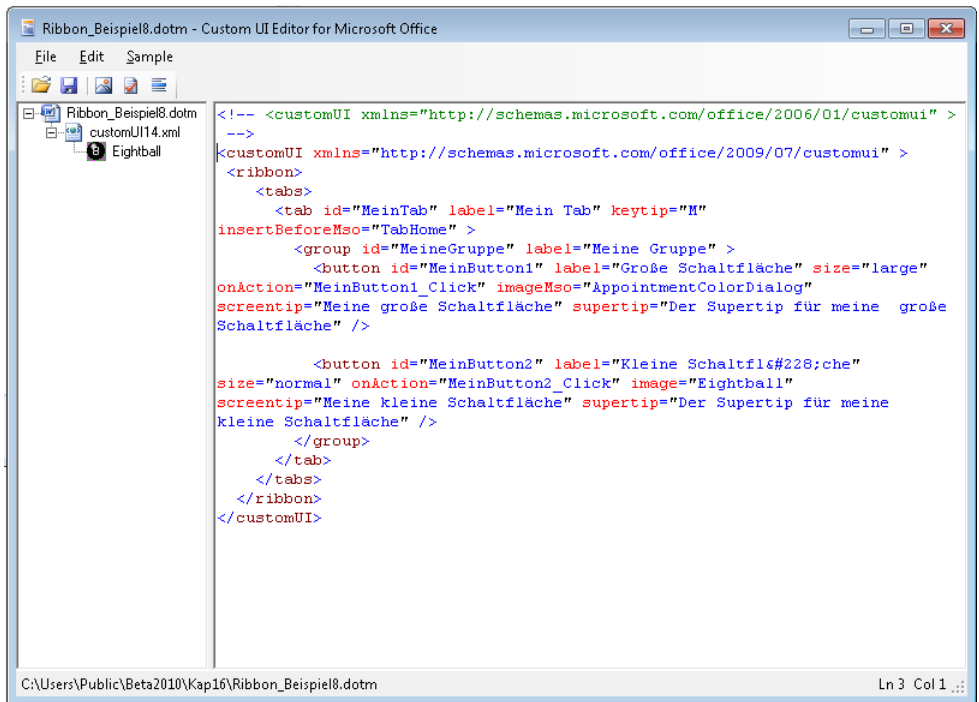
4. Geben Sie im rechten Bereich den gewünschten Menüband-XML-Code ein (Listing 16.4).

Listing 16.4 Die XML-Datei für die Menüband-Erweiterung in Abbildung 16.10

```
<!-- <customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui" > -->
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui" >
  <ribbon>
    <tabs>
      <tab id="MeinTab" label="Mein Tab" keytip="M" insertBeforeMso="TabHome" >
        <group id="MeineGruppe" label="Meine Gruppe" >
          <button id="MeinButton1" label="Große Schaltfläche" size="large"
            onAction="MeinButton1_Click" imageMso="AppointmentColorDialog"
            screentip="Meine große Schaltfläche"
            supertip="Der Supertip für meine große Schaltfläche" />

          <button id="MeinButton2" label="Kleine Schaltfläche" size="normal"
            onAction="MeinButton2_Click" image="Eightball"
            screentip="Meine kleine Schaltfläche"
            supertip="Der Supertip für meine kleine Schaltfläche" />
        </group>
      </tab>
    </tabs>
  </ribbon>
</customUI>
```

5. Um die Grafik *Eightball.png* für die Schaltfläche »MeinButton2« einzubinden, klicken Sie auf die Symbolschaltfläche *Insert Icons*. Navigieren Sie zum Ordner und wählen Sie die Grafikdatei aus (die Grafik erscheint unter der XML-Datei im Bereich links). Der Name der Grafikdatei und die Angabe des Attributs *Image* müssen genau übereinstimmen.

Abbildg. 16.12 Dokument mit Menüband-Erweiterung und eingebetteter Grafik


6. Falls es sich um eine Menüband-Erweiterung für Word 2007 handelt, können Sie auf die Schaltfläche *Validate* klicken, um die Wohlgeformtheit des XML zu kontrollieren.
7. Speichern sowie schließen Sie das Dokument und öffnen Sie es anschließend erneut in Word.

CD-ROM

Das XML-Beispiel befindet sich in der Datei *Beispiel8_customUI.xml*. Sie finden es im Ordner *\Beispiele\Kap16* auf der CD-ROM zum Buch zusammen mit dem Beispieldokument *Ribbon_Beiispiel8.dotm*.

Die dynamische Einbindung: *getImage* & *loadImage*

Für die dynamische Einbindung von Grafikdateien während der Ausführung stellt die Menüband-Erweiterung zwei Attribute zur Verfügung, die eine Codeprozedur abrufen: *loadImage* und *getImage*.

load-
Image

Bei *loadImage* handelt es sich um ein Attribut des Ribbon-Elements und wird nur beim Laden der Menüband-Erweiterung ausgeführt. Dieser Vorgang ist effizienter, das Resultat ist jedoch statisch. Ein Beispiel hierfür steht im Add-In-Beispiel des Abschnitts zum Thema VSTO-Add-Ins in Kapitel 10.

Der XML-Code:

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui" onLoad="OnLoad"
loadImage="ribbon_getImages">
```

Die mit dem Attribut verbundene Callback-Prozedur erhält über die Parameter `imageID` von jedem Steuerelement der Menüband-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 Menüband-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 16.5.

Listing 16.5 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
```

`getImage` Ein Beispiel für `getImage` finden Sie im Abschnitt »gallery« ab Seite 766. Im Gegensatz zu `loadImage` kann während der Word-Sitzung die Grafik, die durch die aufgerufene Prozedur an das Menüband zurückgegeben wird, geändert werden.

Code für den dynamischen Ablauf

In der Diskussion zum Thema Grafiken in der Menüband-Erweiterung wurden zwei Attribute wie `loadImage` vorgestellt, die mit Code verbunden sind, um eine dynamische Handlung zu ermöglichen. Bislang gingen wir nicht näher auf solche Attribute ein – die Wirkungsweise ist in diesem Moment wohl etwas schleierhaft und bedarf eine Erklärung.

Das Attribut `onAction` kennen Sie aus den bisherigen Beispielen. Die Idee ist nicht fremd; sie entspricht dem Click-Ereignis einer herkömmlichen Schaltfläche. Der Benutzer klickt darauf, der Code wird angestoßen, und eine Handlung wird ausgeführt. Nur heißt die Prozedur für die Menüband-Erweiterung nicht Ereignisse, sondern *Callback*.

HINWEIS Der deutsche Ausdruck für *Callback* ist Rückruf-Funktion. Mehr darüber erfahren Sie unter <http://de.wikipedia.org/wiki/Rückruffunktion>.

Die Bezeichnung der meisten Callback der Menüband-Erweiterung fangen mit den Buchstaben »get« (soviel wie »den Wert holen«) an, wie beispielsweise `getImage` (Grafik abrufen), `setVisible` (ist sichtbar), oder `getEnabled` (ist ungesperrt). Es gibt jedoch andere, wie `onLoad` oder `loadImage`. Jedes Steuerelement hat einen passenden Satz solcher Attribute.

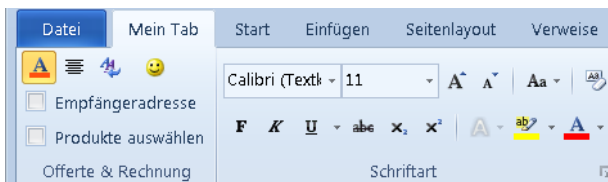
Der Auslöser der mit dem Attribut `onAction` verbundenen Prozedur ist klar: Der Benutzer klickt auf die Schaltfläche. Aber wie werden die anderen Callback-Prozeduren, die den Zustand eines Steuerelements beeinflussen, angestoßen?

Ein solcher Auslöser ist das Laden der Menüband-Erweiterung durch die Office-Anwendung. Dadurch werden alle Callback, die das Aussehen und Zustand des Menübands festlegen, ausgelöst.

Die andere Möglichkeit ist, diese Handlungen während einer Sitzung durchführen zu lassen. Dafür gibt es zwei Methoden des Ribbon-Objekts, die diese Callback anstoßen: `Invalidate`, die alle Callback dieser Art für die gesamte Menüband-Erweiterung auslöst, sowie `InvalidateControl`, welche die Attribute eines einzigen Steuerelements neu evaluieren lässt.

Wie das funktioniert, wird ersichtlicher mit der Hilfe eines kleinen Beispiels: Es liegt eine Dokumentvorlage für die Erstellung von Firmenbriefen vor. Sie enthält eine Gruppe *Offerte & Rechnung*, deren Steuerelemente weitere Gruppen mit Werkzeugen ein- und ausblenden. In Abbildung 16.13 ist die Gruppe *Schriftart* eingeblendet; dafür verantwortlich ist die Umschaltfläche, beschriftet mit dem großen Buchstaben »A«, links im Bild. Das Menüband-XML für diesen Ausschnitt der Menüband-Erweiterung sehen Sie in Listing 16.6 und die damit verbundenen Callback-Prozeduren in Listing 16.7.

Abbildg. 16.13 Die Gruppe links blendet weitere Gruppen ein und aus



Listing 16.6 Die Menüband-XML-Definition des Menübands in Abbildung 16.13

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui"
onLoad="ribbonGeladen" >

  <ribbon>
    <tabs>
      <tab id="MeinTab" label="Mein Tab" insertBeforeMso="TabHome" >
        <group id="GroupOfferte_Rechnung" label="Offerte && Rechnung" >
          <buttonGroup id="UntergruppeFormatieren">
            <toggleButton id="WordSchriftGruppe" imageMso="GroupFont"
screenTip="Schriftartgruppe einblenden"
supertip="Blendet die Word-Schriftartgruppe ein und aus."
onAction="WordSchriftGruppe_Click" keytip="H" />
            <!-- Weitere Schaltflächen werden hier definiert -->
          </buttonGroup>
        </group>
        <group idMso="GroupFont" getVisible="GroupFont_GetVisible" />
      </tab>
    </tabs>
  </ribbon>
</customUI>
```


Listing 16.7 Die von den Attributen in Listing 16.6 verbundenen Callback-Prozeduren

```

Public thisRibbon As Office.IRibbonUI
Private groupFontVisible As Boolean

'Callback for customUI.onLoad
'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
    'Für idMso muss das ganze Ribbon neu evaluiert werden,
    'weil es mehr als ein Steuerelement mit dem gleichen idMso
    'im Menüband haben könnte
End Sub

Sub GroupFont_GetVisible(control As IRibbonControl, ByRef returnedVal)
    returnedVal = groupFontVisible
End Sub

```

Beim Laden einer Menüband-Erweiterung werden die Callback-Prozeduren des Elements customUI ausgeführt. In diesem Fall ruft das Attribut onLoad (»beim Laden«) die Prozedur *ribbonGeladen* auf. Diese ihrerseits weist die Menüband-Erweiterung der Objektvariablen *thisRibbon* zu, sodass sie während der Sitzung angesprochen werden kann. Über das *thisRibbon*-Objekt haben wir Zugang auf die Methoden *Invalidate* sowie *InvalidateControl*. In diesem Beispiel brauchen wir die Erste, um die Sichtbarkeit der Gruppe *Schriftart* zu steuern.

WICHTIG

Die Menüband-Erweiterung kann nicht direkt durch das Word- oder Office-Objektmodell angesprochen werden. Wird sie beim Laden keiner Objektvariablen zugewiesen, steht sie dem Programmiercode nicht zur Verfügung.

**Das Steuerelement Umschaltfläche**

Eine gewöhnliche Schaltfläche (<button>-Element) führt immer den gleichen Befehl aus. Von einer Umschaltfläche (<toggleButton>-Element) 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 Gruppe *Schriftart*.

Bei Betätigung dieser Umschaltfläche wird die Prozedur *WordSchriftGruppe_Click* ausgeführt, die dem Attribut *onAction* zugewiesen ist. Das *onAction*-Callback einer Umschaltfläche verfügt im Gegensatz zu einer gewöhnlichen Schaltfläche über zwei Parameter. Der erste ist identisch: Er übergibt das rufende Steuerelement als ein *RibbonControl*-Objekt. Der andere (*pressed*) ist vom Datentyp *Boolean* und übermittelt, ob der gegenwärtige Zustand der Umschaltfläche *ein* oder *aus* ist. Anhand dieser Parameter wird entschieden, welche Handlungen auszuführen sind.

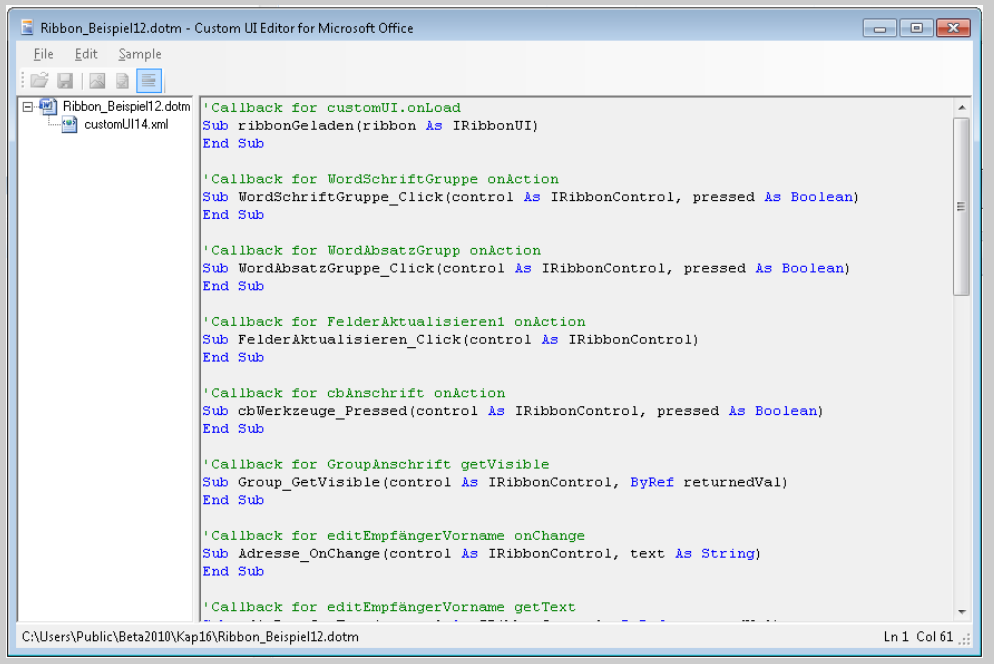
HINWEIS

Eine vollständige Auflistung der Steuerelemente und ihrer Attribute finden Sie im Artikel »Customizing the 2007 Office Fluent Ribbon For Developers« auf <http://msdn2.microsoft.com/en-us/library/aa338199.aspx>. Eine Liste der Callbacks (für die Programmiersprachen C#, VBA, C++ sowie VB.NET) mit ihren Parametern befindet sich im Teil 3 desselben Artikels (<http://msdn2.microsoft.com/en-us/library/ms406046.aspx>).

Der Custom UI Editor

Das Werkzeug Custom UI Editor hilft mit der Erstellung von Callback-Prozeduren. Bei Betätigung der Schaltfläche *Callbacks* schleift es durch alle Zeilen des XML-Codes, sucht Callback-Attribute und schreibt für jede die korrekte VBA-Prozedur in einem Editorfenster (Abbildung 16.14). Das Resultat kann in die Zwischenablage kopiert und in ein Word-VBA-Codefenster eingefügt werden.

Abbildg. 16.14 Die Makrostrukturen für Callback-Prozeduren vom Custom UI Editor erstellen lassen



Im vorliegenden Beispiel wird der von pressed gelieferte Wert einer globalen Variablen (groupFontVisible) zugewiesen, die ihn zwischenspeichert. Da die Arbeitsweise der Menüband-Erweiterung es verbietet, von außerhalb auf die Steuerelemente Einfluss zu nehmen, kann der Code die Gruppe nicht direkt ein- oder ausblenden.

Stattdessen muss die Menüband-Erweiterung erneut initialisiert werden, was die letzte Befehlszeile der Prozedur veranlasst: `thisRibbon.Invalidate`. Dadurch werden die Callbacks der gesamten Menüband-Erweiterung angestoßen, u.a. das `getVisible`-Callback der zweiten Gruppe in Listing 16.10:

```
<group idMso="GroupFont" getVisible="GroupFont_GetVisible" />.
```

Die Prozedur `GroupFont_GetVisible` übergibt den Wert der globalen Variablen an die Parameter `returnedVal`. Ist er wahr, wird die Gruppe eingeblendet, sonst wird sie ausgeblendet.

WICHTIG

Weil der Zustand eines einzigen Steuerelements geändert werden soll, würden Sie vielleicht fragen, warum die Methode `Invalidate` statt `InvalidateControl` eingesetzt wird. Der Grund ist, dass es sich in diesem Fall um eine Word-eigene Gruppe handelt (`idMso="GroupFont"`), was am Attribut `idMso` zu erkennen ist. Weil das Steuerelement mehrmals in der Benutzerschnittstelle vorkommen kann, kann kein spezifisches Steuerelement neu evaluiert werden. Deshalb muss in einem solchen Fall die ganze Menüband-Erweiterung aktualisiert werden.

CD-ROM

Das XML-Beispiel befindet sich in der Datei *Beispiel9_customUI.xml*. Sie finden es im Ordner `\Beispiele\Kap16` auf der CD-ROM zum Buch zusammen mit dem Beispieldokument *Ribbon_Beispiel9.dotm*.

Menüband-Registerkarten teilen

Eingangs wurde erwähnt, die Menüband-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 oder Add-Ins 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 sollen.

In früheren Word-Versionen wurde eine Symbolleiste mit Makros in einer Vorlage erstellt. Die Vorlage wird als globales Vorlagen-Add-In geladen, sodass diese Symbolleiste immer zur Verfügung steht. Die andere 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 Menüband-Erweiterungen, aber das »Wie« ist nicht sofort erkennbar. Der Schlüssel liegt im Gebrauch eines Namensraums (Namespace). Wie in Kapitel 21 diskutiert, ermöglicht ein Namensraum die Zuteilung der XML-Elemente zu bestimmten Kategorien. Standardmäßig gehören alle Elemente des `customUI`-XMLs dem Namensraum <http://schemas.microsoft.com/office/2009/07/customui> (Office 2010) bzw. <http://schemas.microsoft.com/office/2006/01/customui> (Office 2007) – das steht im eröffnenden XML-Element. Die Angabe weiterer Namensräume ist aber erlaubt.

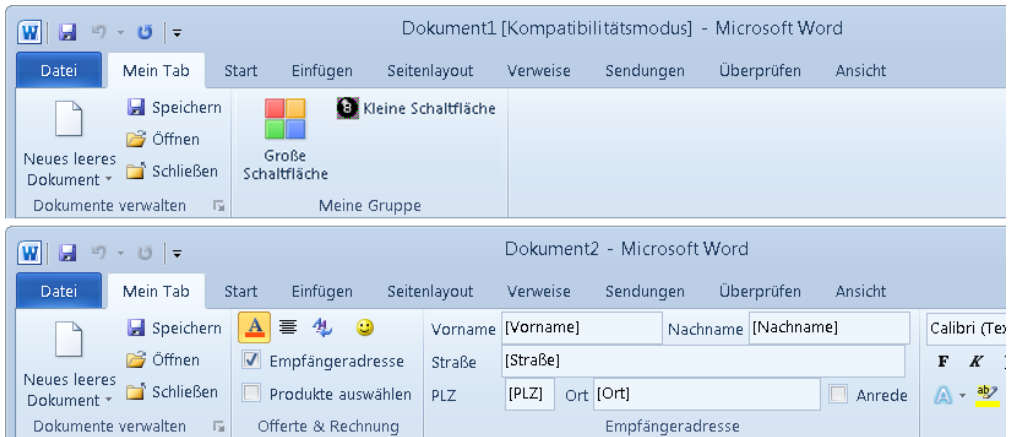
Vergleichen Sie die in Listing 16.6 wiedergegebenen `customUI`-XML mit demjenigen in Listing 16.8. Letzteres weist zusätzlich auf den Namensraum `xmlns:n="http://ISBN978-3866454583.com/RibbonXML"`. Das Präfix *n* für den Namensraum wird im Tab-Element als Teil des Attributwerts `idQ`

(steht für qualifizierte ID) eingesetzt: `idQ="n:MeinTab"`. Dies erlaubt mehreren Erweiterungen die gleiche Registerkarte zu benutzen, wie in Abbildung 16.15 ersichtlich. Die Gruppe links auf der Registerkarte *Mein Tab* ist in einer globalen Vorlage definiert. Die Gruppen rechts davon befinden sich in der Menüband-Erweiterung 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.

Abbildg. 16.15 Die Menüband-Erweiterungen mehrerer Dokumente teilen ine Registerkarte mit einer globalen Vorlage


CD-ROM

Das XML-Beispiel befindet sich in den Dateien *Beispiel10a_customUI.xml*, *Beispiel10b_customUI.xml* und *Beispiel10c_customUI.xml*. Sie finden diese im Ordner `\Beispiele\Kap16` auf der CD-ROM zum Buch zusammen mit den Beispielvorlagen *Ribbon_Beispiel10a.dotm*, *Ribbon_Beispiel10b.dotm* und *Ribbon_Beispiel10c.dotm*. Kopieren Sie *Ribbon_Beispiel10c.dotm* in den *Startup*-Ordner für Word, um die Vorlage als globales Add-In in Word zu laden. (Mehr zum Thema Add-Ins lesen Sie in Kapitel 10.)

Eine Registerkarte anwählen



In Office 2010 wurde die Menüband-Erweiterung um die Methoden `ActivateTab`, `ActivateTabMso` und `ActivateTabQ` erweitert; sie gehören alle dem `Ribbon`-Objekt an. Damit kann Code innerhalb einer Menüband-Erweiterung eine selbstdefinierte, Word-eigene bzw. qualifizierte Registerkarte anwählen. (Qualifizierte Steuerelemente wurden im Abschnitt »Menüband-Registerkarten teilen« ab Seite 751 kurz vorgestellt.) Die Syntax für alle Methoden:

```
ribbonObjekt.ActivateTab "[controlID]"
ribbonObjekt.ActivateTabMso "[msoId]"
ribbonObjekt.ActivateTabQ "[controlID]", "[Namensraum]"
```

Unser Beispiel hierfür basiert auf dem Steuerelement *Öffnen* der gemeinsamen Gruppe *Dokumente verwalten* in Abbildung 16.15. Bekanntlich zeigt Word beim Öffnen eines Dokuments standardmäßig die erste Registerkarte an. Früher hat es für Seriendruckdokumente automatisch die dazugehörige Symbolleiste mit den Werkzeugen für den Seriendruck eingeblendet. Dieses Beispiel erweitert die Funktionalität der Schaltfläche *Öffnen*, sodass beim Öffnen eines Seriendruckdokuments die Registerkarte *Sendungen* angezeigt wird. Das Menüband-XML sehen Sie in Listing 16.8, den VBA-Code in Listing 16.9.

Listing 16.8 Das Menüband-XML für die Gruppe *Dokumente verwalten* in Abbildung 16.15

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui" xmlns:n="http://
ISBN978-3866454583.com/RibbonXML" onLoad="RibbonInitialisieren" >
  <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" keytip="N" >
            <button idMso="FileNewDefault" />
            <menu keytip="V" >
              <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" keytip="S" />
          <button id="meinFileOpen" imageMso="FileOpen" label="Öffnen" screentip="Datei
öffnen"
            onAction="meinDateiOeffnen" keytip="O" />
          <button idMso="FileClose" keytip="C" />
          <dialogBoxLauncher>
            <button idMso="FileNew" />
          </dialogBoxLauncher>
        </group>
      </tab>
    </tabs>
  </ribbon>
</customUI>
```

Listing 16.9 Der VBA-Code, um eine Registerkarte anzuwählen

```
Option Explicit

Dim meinRibbon As IRibbonUI

Sub RibbonInitialisieren(ribbon As IRibbonUI)
  Set meinRibbon = ribbon
End Sub

'Weitere Prozeduren ...

Sub meinDateiOeffnen(control As Office.IRibbonControl)
  CommandBars.ExecuteMso "FileOpen"
  If ActiveDocument.MailMerge.MainDocumentType <> wdNotAMergeDocument Then
    meinRibbon.ActivateTabMso "TabMailings"
  End If
End Sub
```

Aus diesen Listings erkennen Sie die Initialisierung eines Ribbon-Objekts im VBA-Projekt. Beim Anklicken der Schaltfläche *Öffnen* wird die Prozedur *meinDateiOeffnen* angestoßen. Diese führt die Word-eigene *FileOpen* (Datei öffnen) mittels der Methode *CommandBars.ExecuteMso* aus und aktiviert anschließend die Registerkarte *Sendungen* (TabHome), falls das geöffnete Dokument ein Seriendruckdokument ist.

ACHTUNG

Es wäre sehr nützlich, wenn diese Methode es uns erlauben würde, eine beliebige Registerkarte beim Öffnen des Dokuments anzuwählen, statt sie an erster Stelle des Menübands positionieren zu müssen. Leider funktioniert dies nicht, es besteht ein Timingproblem. Um das Laden der Benutzerschnittstelle zu beschleunigen, werden Teile des Menübands erst bei Bedarf geladen. Das Ribbon-Objekt liegt zum Zeitpunkt des Öffnens oder Anlegens eines Dokuments nicht bereit. Nachdem das Menüband erfolgreich geladen und initialisiert wurde, funktioniert das Vorhaben. Aber am zuverlässigsten ist die Methode, wenn sie durch ein Steuerelement ausgelöst wird.

HINWEIS

In Word 2007 ist der einzige Weg, mit *SendKeys* die Tastenschläge für den Keytip durchzugeben (siehe den Abschnitt »Befehle mit Tastenkombinationen verbinden« ab Seite 738).

CD-ROM

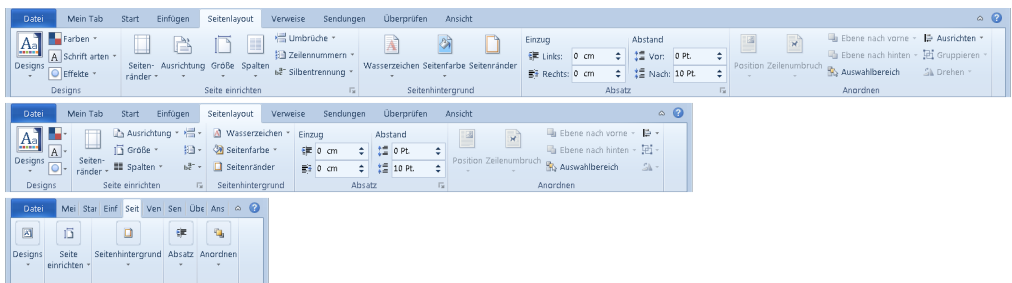
Das XML-Beispiel befindet sich in der Datei *Beispiel11_customUI.xml*. Sie finden es im Ordner *\Beispiele\Kap16* auf der CD-ROM zum Buch zusammen mit dem Beispieldokument *Ribbon_Beispiel11.dotm*.

Dynamische Größenanpassung von Gruppen



Wird das Word-Fenster schmaler, passt sich die Größe der Word-eigenen Gruppen im Menüband an. Große Schaltflächen werden kleiner, dann wird der Text entfernt und am Schluss repräsentiert lediglich ein Symbol mit Dropdownliste die Befehle der Gruppe (Abbildung 16.16).

Abbildg. 16.16 Die stufenweise Verkleinerung Word-eigener Gruppen im Menüband



Leider stand in Office 2007 den Entwicklern von Menüband-Erweiterungen keine entsprechende Funktionalität zur Verfügung. Die Gruppen werden entweder gänzlich angezeigt oder auf Symbolebene für die ganze Gruppe verkleinert. Diese Lücke wurde in Office 2010 durch Einführung des Attributs *AutoScale* geschlossen.

Das Attribut `AutoScale` erwartet einen booleschen Wert. Ist er Wahr (`True`), wird die Gruppe stufenweise verkleinert, ähnlich wie bei den Word-eigenen: Zuerst wird die Beschriftung der Spalten (von rechts nach links) ausgeblendet, dann die Gruppe in der Dropdownliste einer großen Schaltfläche zusammengefasst. Das Symbol der zusammengefassten Gruppe wird durch das Attribut `Image` festgelegt.

```
<group id="GroupProdukte" label="Produkte auswählen" getVisible="Group_GetVisible"
autoScale="true" imageMso="SmartArtChangeColorsGallery" >
```

CD-ROM

Das XML-Beispiel befindet sich in der Datei *Beispiel12_customUI.xml*. Sie finden es im Ordner *\Beispiele\Kap16* auf der CD-ROM zum Buch zusammen mit dem Beispieldokument *Ribbon_Beispiel12.dotm*.

Die Menüband-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 Artikel *Customizing the 2007 Office Fluent Ribbon For Developers* (Teil 2) aufgelistet.

Die Steuerelemente

Das Menüband unterstützt natürlich mehr Steuerelemente als die bisher vorgestellten, herkömmlichen Formen einer Schaltfläche oder einer Umschaltfläche. Eine kurze Übersicht bietet die Tabelle 16.2.

Tabelle 16.2 Liste der dem Menüband zur Verfügung stehenden Steuerelemente

XML-Elementbezeichnung	Beschreibung
<code><button></code>	Schaltfläche
<code><checkBox></code>	Kontrollkästchen
<code><comboBox></code>	Kombinationsfeld
<code><dialogBoxLauncher></code>	Startet ein Dialogfeld (blendet es ein)
<code><dropDown></code>	Dropdownfeld
<code><dynamicMenu></code>	Ein Menü, das bei Laufzeit erstellt wird
<code><editBox></code>	Bearbeitungsfeld

Tabelle 16.2 Liste der dem Menüband zur Verfügung stehenden Steuerelemente (Fortsetzung)

XML-Elementbezeichnung	Beschreibung
<gallery>	Katalog oder Sammlung. Bietet eine grafische Liste zur Auswahl an.
<labelControl>	Bezeichnungsfeld
<menu>	Menü
<menuSeparator>	Menütrennlinie
<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.

HINWEIS

Das Schema *customUI.xsd* für Office 2010 kann von der Microsoft-Webseite <http://www.microsoft.com/downloads/details.aspx?FamilyID=c2aa691a-8004-46ac-9852-102f1d5bcd18&displaylang=en> heruntergeladen werden. Das Schema für Office 2007 steht auf <http://www.microsoft.com/downloads/details.aspx?FamilyID=15805380-F2C0-4B80-9AD1-2CB0C300AEF9&displaylang=en> zur Verfügung.

Das folgende Beispiel veranschaulicht den Gebrauch vieler dieser Steuerelemente und ihrer Attribute. In Abbildung 16.17 ist die volle Funktionalität des Firmenbrief-Beispiels ersichtlich. Neben den schon erwähnten Word-eigenen Gruppen für die Schriftart- und Absatzformatierung enthält die Vorlage Gruppen für die Erfassung der Empfängeradresse sowie eine Liste der Firmenprodukte, die für das Schreiben einer Offerte oder einer Rechnung ins Dokument eingefügt werden können. Das vollständige Menüband-XML dieser Vorlage entnehmen Sie bitte dem Listing 16.10.

Die Steuerelemente der Gruppe *Empfängeradresse* sind vom Typ `editBox`. Ihr Inhalt wird aus den und in die vordefinierten Inhaltsteuerelemente im Dokument gelesen bzw. geschrieben. In dieser Gruppe befindet sich zusätzlich ein Kontrollkästchen, das ein Kombinationsfeld (`comboBox`) ein- und ausblendet. Damit kann der Benutzer eine vordefinierte Anrede auswählen oder eine eigene eingeben.

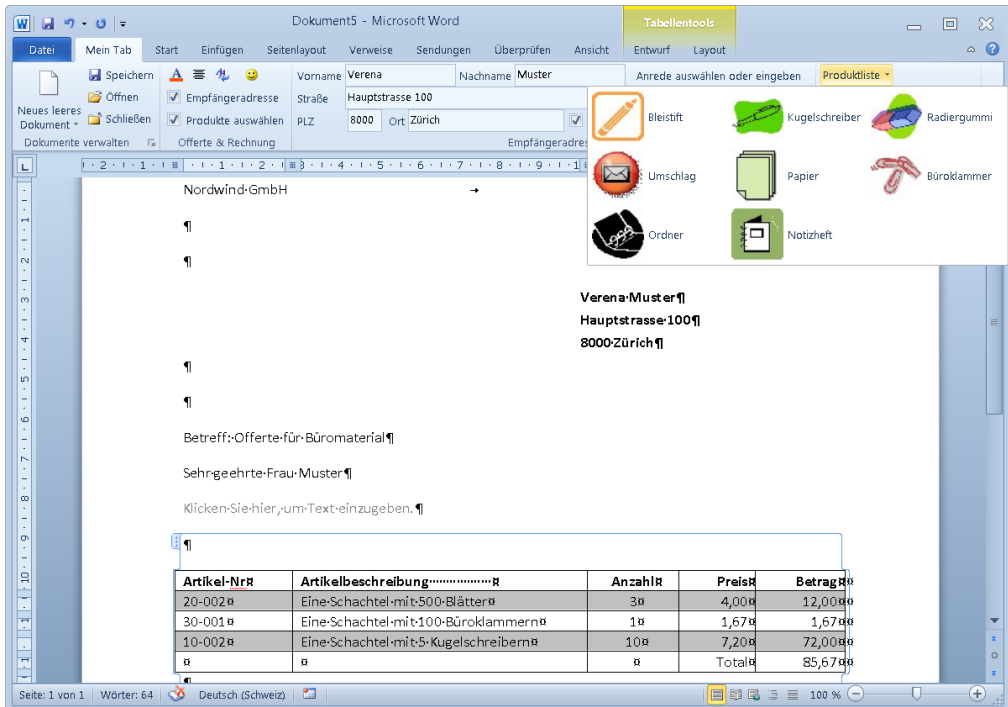
Die Gruppe *Produkte auswählen* enthält einen Katalog (`gallery`) mit Produkten. Im Dokument werden diese in einer Tabelle aufgelistet mit Feldfunktionen für die Berechnungen. Dem Benutzer bleibt lediglich die Aufgabe, die Quantität einzugeben und die Berechnungen zu aktualisieren, wofür es in der Registerkarte 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 Menüband-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. Ohne den Einsatz dieser Ele-

mente werden die Steuerelemente von oben nach unten und von links nach rechts angeordnet; die Ausrichtung ist eher chaotisch.

Nachfolgend werden die Steuerelemente im Detail zusammen mit dem mit diesen verbundenen VBA-Code vorgestellt.

Abbildg. 16.17 Die Menüband-Erweiterung der Vorlage für Offerten mit selbstdefinierten Gruppen eingeblendet



Listing 16.10 Der XML-Code der Beispieldokumentvorlage für Firmenbriefe

```
<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 && 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" />
            </buttonGroup>
          </box>
        </group>
      </tab>
    </tabs>
  </ribbon>
</customUI>
```

Listing 16.10 Der XML-Code der Beispieldokumentvorlage für Firmenbriefe (Fortsetzung)

```

        <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 für die Eingabe der Anschrift ein und aus."
        onAction="cbWerkzeuge_Pressed" keytip="E" />

    <checkBox id="cbProdukte" label="Produkte auswählen"
        screentip="Produkte auswählen"
        supertip="Blendet die Gruppe für das Einfügen der Produkte ein und aus."
        onAction="cbWerkzeuge_Pressed" keytip="P" />
</box>
</group>

<group id="GroupAnschrift" label="Empfängeradresse"
    getVisible="Group_GetVisible" autoScale="true" imageMso="ViewFormView" >
    <box id="boxEmpfängerName" boxStyle="horizontal" >
        <editBox id="editEmpfängerVorname" label="Vorname"
            sizeString="aaaaaaaaaaaaaa" onChange="Adresse_OnChange"
            getText="editBox_GetText" keytip="R" />

        <editBox id="editEmpfängerNachname" label="Nachname"
            sizeString="aaaaaaaaaaaaaa" onChange="Adresse_OnChange"
            getText="editBox_GetText" keytip="M" />
    </box>
    <box id="boxEmpfängerStraße" boxStyle="horizontal" >
        <editBox id="editEmpfängerStraße" label="Straße"
            sizeString="aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa"
            onChange="Adresse_OnChange" getText="editBox_GetText" keytip="B" />
    </box>
    <box id="boxEmpfängerPLZOrt" boxStyle="horizontal" >
        <editBox id="editEmpfängerPLZ" label="PLZ" maxLength="5"
            sizeString="00000" onChange="Adresse_OnChange" getText="editBox_GetText"
            keytip="L" />

        <editBox id="editEmpfängerOrt" label="Ort" sizeString="aaaaaaaaaaaaaaaaaaaaaa"
            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" />
    </box>
</group>

```

Listing 16.10 Der XML-Code der Beispieldokumentvorlage für Firmenbriefe (Fortsetzung)

```

        <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" autoScale="true"
    imageMso="SmartArtChangeColorsGallery" >
    <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" />

    <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>

```

Schaltfläche



Die Schaltfläche dient hauptsächlich dazu, eine Handlung über die mit dem Attribut `onAction` verbundene Prozedur auszuführen. Die meist gebrauchten Attribute der Schaltfläche werden von vielen Steuerelementen unterstützt und immer gleich eingesetzt. Das Steuerelement sowie die meisten Attribute wurden im Abschnitt »Erweiterte Funktionalität« ab Seite 738 vorgestellt.

- **tag** Ermöglicht die Hinterlegung von Informationen, die dem Programmablauf dienen (siehe auch das Listing 16.11)
- **label** Die Beschriftung des Steuerelements
- **size** Legt die Größe der Schaltfläche fest. Der Wert `large` veranlasst das Menüband, 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 Abbildung 16.15 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.
- **supertip** Der supertip erscheint im mittleren Teil der QuickInfo und dient als Hilfetext für den Befehl
- **onAction** Führt die angegebene Prozedur aus

dialogBoxLauncher



In der unteren rechten Ecke vieler Gruppen des Menübands 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>
```

Wie dem Codefragment zu entnehmen ist, muss dieses Steuerelement ein Unterelement des Typs `button` enthalten. Es befindet sich zwingend an letzter Stelle der Elemente einer Gruppe, wie Listing 16.8 zu entnehmen ist.

Hier wird das Word-eigene Dialogfeld *Neues Dokument* eingeblendet. Das Unterelement `button` darf aber auch über `onAction` eine Prozedur aufrufen, die ein benutzerdefiniertes Formular einblendet.

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 globalen Dokumentvorlage des Abschnitts »Menüband-Registerkarten teilen« ab Seite 751 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 16.11 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-Attributs des rufenden Steuerelements, was zu tun ist. Je nach Steuerelement wird die passende Dokumentvorlage für das zu erstellende Dokument gewählt.

Listing 16.11 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_Beiispiel10a.dotm"
        Case "Offerte oder Rechnung"
            Documents.Add Template:=pfad & "Ribbon_Beiispiel12.dotm"
        Case Else
            End Select
    End Sub
```

toggleButton



Dieses Steuerelement wurde im Abschnitt »Code für den dynamischen Ablauf« ab Seite 747 vorgestellt. 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 Menüband-Erweiterung festgelegt werden. Standardmäßig wird sie im Zustand »aus« (entspricht dem Wert »falsch«) geladen. Mit einer Callback-Prozedur kann dynamisch entschieden werden, welchen Zustand die Umschaltfläche haben soll. Dieses Attribut besitzt das Steuerelement `checkBox` ebenfalls und wird in jenem Abschnitt näher erläutert.

checkBox



Ein Kontrollkästchen ist einer Umschaltfläche sehr ähnlich. Beide werden für die Verwaltung eines »Ein-oder-aus«-Zustands eingesetzt und haben viele Attribute gemeinsam. Ein `checkBox`-Steuerelement hat jedoch weniger Gestaltungsmöglichkeiten und verfügt daher über weniger Attribute. Es unterstützt beispielsweise keine Grafik und seine Größe ist festgeschrieben.

Im vorliegenden Beispiel rufen beide Kontrollkästchen der Gruppe *Offerte & Rechnung* die gleiche `onAction`-Prozedur auf, die in Listing 16.12 ersichtlich sind.

```
<checkBox id="cbAnschrift" label="Empfängeradresse"
    screentip="Empfängeradresse eingeben"
    supertip="Blendet die Gruppe für die Eingabe der Anschrift ein und aus."
    onAction="cbWerkzeuge_Pressed" keytip="E" />
```

Der Ablauf ist fast gleich wie bei der im Abschnitt »Code für den dynamischen Ablauf« ab Seite 747 vorgestellten Umschaltfläche: 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 Call-back für `setVisible` aufzurufen, der die Prozedur `Group_GetVisible` ausführt.

```
<group id="GroupAnschrift" label="Empfängeradresse" setVisible="Group_GetVisible" >
```

Listing 16.12 Mit den *checkBox*-Steuerelementen verbundene 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
            MsgBox "Unbekannte Gruppe: " & control.Tag
    End Select
End Sub
```

box

Das Steuerelement `box` gruppiert die Steuerelemente innerhalb einer Gruppe. Es darf alle befehlsausführenden Steuerelemente sowie weitere `box`- und `buttonGroup`-Elemente enthalten.

Neben 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` (waagerecht) sowie `vertical` (senkrecht).

Das Beispiel veranschaulicht, wie das Steuerelement das Layout beeinflusst. Es legt fest, welche Steuerelemente 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>
```

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 waagrecht 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>
```


editBox

Ein Bearbeitungsfeld ist für die Texteingabe vorgesehen. Zu Zeiten der Symbolleiste machte es meist Sinn, ein Dialogfeld einzublenden. Die Anordnungsmöglichkeiten der Menüband-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 interessante Attribute:

- **getText** Holt den im Bearbeitungsfeld anzuzeigenden Text
- **maxLength** Legt die maximale Anzahl der einzugebenden Zeichen fest

PLZ [PLZ]

- **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 Bearbeitungsfelds 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 16.13 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 Bearbeitungsfelds 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.

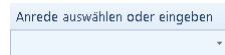
Listing 16.13 Die vom Steuerelement *editBox* aufgerufenen Prozeduren

```
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
        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
```


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 dynamischen Attribute für eine Auflistung eingesetzt; diese werden unter gallery näher erläutert. Das Beispiel enthält nur Attribute, die unter editBox bereits vorgestellt wurden. Dafür veranschaulicht es, wie im XML-Code mittels des Elements item eine statische Liste definiert wird.

```
<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>
```

item hat die üblichen Attribute id, image, imageMso, label, screentip sowie supertip.

Der mit diesem Steuerelement verbundene VBA-Code ist in Listing 16.14 ersichtlich. Das Kombinationsfeld *Anrede* 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 16.14 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
```

Listing 16.14 Die vom Steuerelement *comboBox* aufgerufenen Prozeduren (Fortsetzung)

```
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
```

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.

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 Indexwert 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 16.15 ersichtlich. Die Daten für die Produktliste befinden sich in einer XML-Datei. Vom besonderen Interesse ist die Prozedur *Gallery_GetItemImage*, da sie statt einer Zeichenkette eine Grafik an die Menüband-Erweiterung zurückgibt. Die Grafik muss als *stdOLE-Picture* vorliegen – nicht etwas, das die meisten Office-Entwickler tagtäglich einsetzen.

Listing 16.15 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
    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
```

Listing 16.15 Vom Steuerelement *gallery* aufgerufene Prozeduren (Fortsetzung)

```

    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

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

```

Listing 16.15 Vom Steuerelement *gallery* aufgerufene Prozeduren (Fortsetzung)

```

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")
    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

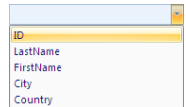
```

CD-ROM Die Beispieldateien *Ribbon_Beispiel10c.dotm*, *Ribbon_Beispiel12.dotm* und *Produkte.xml* sowie die zugehörigen acht Grafikdateien finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap16*. Kopieren Sie den Ordner auf Ihren Rechner und vertrauen Sie ihm unter *Word Optionen/Vertrauensstellencenter/Einstellungen für das Vertrauensstellencenter/Vertrauenswürdige Speicherorte/Neuen Speicherort hinzufügen*, wie in Kapitel 14 beschrieben.

dropDown

Die Steuerelemente `comboBox`, `editBox` und `dropDown` haben viele Attribute gemeinsam. Zusätzlich stellt `dropDown` die folgenden Attribute zur Verfügung:

- **getSelectedItemID** Die aufgerufene Prozedur legt bei der Initialisierung anhand des ID-Werts (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="dropdownMergeFieldListe" onAction="MergeFieldName_Select"
getItemCount="dropDownFeldListe_getItemCount"
getItemID="dropDownFeldListe_getItemLabel"
getItemLabel="dropDownFeldListe_getItemLabel"
getEnabled="sdAktiviert_getEnabled"
sizeString="Das Seriendruckfeld auswählen:" />
```

dynamicMenu

Wie bereits erwähnt, muss ein Menüband vollständig definiert sein, bevor es von einer Anwendung geladen wird. Mittels der mit dem XML verbundenen Call-back-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 16.16 aufzeigt, ist es möglich, nach Bedarf ein Menü dynamisch zu definieren. Das XML wird genau gleich aufgebaut wie in der XML-Datei für das Menüband. Achten Sie darauf, dass der Namensraum des Menüband-XML im Element `menu` enthalten sein muss.



Listing 16.16 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
    Dim q As String
    q = Chr(34)
    menuXML = "<menu xmlns=" & q & "http://schemas.microsoft.com/office/2009/07/customui" _
        & q & "><button idMso=" & q & "EnvelopesAndLabelsDialog" & q & " /><button idMso=" _
        & q & "LabelsDialog" & q & " /></menu>"
    menuString = menuXML
End Sub
```

HINWEIS Sie finden noch ein Beispiel eines dynamischen Menüs im Abschnitt »Kontextmenüs definieren« ab Seite 771.

CD-ROM Der Beispielcode für *loadImage* und *dynamicMenu* befindet sich in der Datei *Ribbon_Beiispiel13.docm* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap16*.

Kontextmenüs definieren



Kontextmenüs in der Office-Umgebung sind seit der Einführung der Menüband-Erweiterung wichtiger denn je. Da es nicht mehr möglich ist, eine Symbolleiste beliebig auf der Seite zu positionieren und VBA-Entwickler keine Aufgabenbereiche erstellen können, muss der Benutzer teilweise weite Strecken mit der Maus zurücklegen, um an Befehle zu kommen. Es ist wichtig, oft benutzte Befehle näher der Arbeit zur Verfügung zu stellen – beim Rechts anklicken also.

Wie im Buch gelegentlich erwähnt, wird der Teil des Office-Objektmodells, der mit Symbolleisten zu tun hat, kontinuierlich ausgemustert. In Office 2007 diente es noch für die Erstellung und Verwaltung von Kontextmenüs. Ab Office 2010 wurde diese Funktionalität in das Menüband-XML aufgenommen. Alter Code funktioniert weiterhin, es ist aber vorhersehbar, dass er in irgendeiner zukünftigen Office-Version nicht mehr unterstützt wird. Die *CommandBars*-Methode bleibt jedoch die einzige, die eine dynamische Verwaltung der Kontextmenüs ermöglicht.

HINWEIS Das Kapitel zu *CommandBars* aus der 2. Auflage dieses Buchs, mit einer Diskussion über Kontextmenüs, finden Sie auf der CD-ROM im Ordner *\Beilagen\Kap16_Menüs_und_Symbolleisten*.

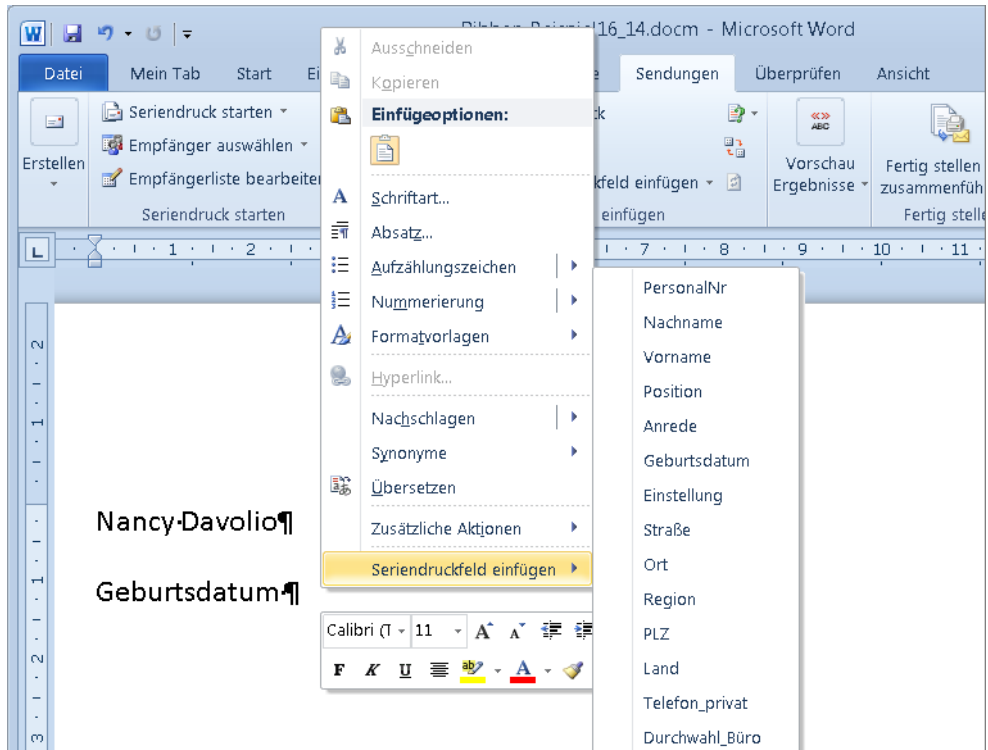
Die Arbeit mit diesem Teil der Office Fluent UI unterscheidet sich unwesentlich von der für das Menüband. Mit Ausnahme der Steuerelemente für die Benutzereingabe, wie *editBox* und *comboBox*, stehen die gleichen Elemente zur Verfügung.

Die Restriktionen sind in etwa die gleichen. Es ist beispielsweise nicht möglich, den Inhalt oder die Reihenfolge eines Word-eigenen Menüs zu ändern. Eigene Menüpunkte können hinzugefügt und frei positioniert werden.

Es ist auch nicht möglich, Kontextmenüs dynamisch zu erstellen. Wie beim Menüband müssen alle Komponenten in dem Menüband-XML vorgängig definiert sein (mit Ausnahme eines *dynamicMenu*-Elements).

Nehmen wir als Beispiel das Einfügen von Seriendruckfeldern in ein Seriendruckdokument. Die standardmäßige Vorgehensweise ist relativ zeitraubend: Der Benutzer gibt Text ein, geht dann mit der Maus zum Menüband, blendet die Registerkarte *Sendungen* ein, klickt auf eine Schaltfläche und wählt das gewünschte Feld. Viel schneller wäre, er könnte im Text dort, wo das Feld stehen soll, rechts anklicken und aus einer Liste wählen, wie in Abbildung 16.18 ersichtlich.

Abbildg. 16.18 Feld aus einem dynamischen Menü wählen



Das XML hierfür ist in Listing 16.17 ersichtlich. Statt eines `<ribbon>`-Elements ist das Element `<contextMenus>`, gefolgt von `<contextMenu>` deklariert. Dazwischen befinden sich einige der im Abschnitt »Die Steuerelemente« ab Seite 755 vorgestellten Steuerelemente. Von Interesse ist vor allem das dynamische Menü.

TIPP

Die XML-Datei darf sowohl einen `<ribbon>` wie auch einen `<contextMenus>`-Abschnitt enthalten. Das Vorhandensein des einen schließt den anderen *nicht* aus.

Listing 16.17 Der XML-Code für ein Kontextmenü

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
  <contextMenus>
    <contextMenu idMso="ContextMenuText">
      <button idMso="HyperlinkInsert" enabled="false" />
      <menuSeparator id="sd_MenuSeparator" />
      <dynamicMenu id="sd_Felder" label="Seriendruckfeld einfügen">
```


Listing 16.17 Der XML-Code für ein Kontextmenü (Fortsetzung)

```

        getContent="FeldListeErstellen" />
    </contextMenu>
</contextMenus>
</customUI>

```

Das Attribut `getContent` ruft die Prozedur `FeldListeErstellen` in Listing 16.18 auf. Innerhalb einer Schleife durch die Seriendruckfelder wird die Zeichenkette mit dem XML für das Menü zusammengestellt. Neben den üblichen Attributen wie `id` und `label` sind auch `onAction` und `tag` definiert. `onAction` führt die Prozedur `SeriendruckfeldEinfügen` aus, die das Feld ins Dokument einfügt. So weiß die Prozedur, um welches Feld es sich handelt, wurde der Feldname der Tag-Eigenschaft des jeweiligen `<button>`-Steuerelements zugewiesen, die die Prozedur abfragt.

Listing 16.18 Die Zeichenkette für das dynamische Menü zusammenstellen

```

Sub FeldListeErstellen(control As Office.IRibbonControl, menuString)
    Dim menuXML As String
    Dim q As String
    Dim doc As Word.Document
    Dim mmData As Word.MailMergeDataSource
    Dim fieldName As Word.MailMergeFieldName
    Dim fName As String

    q = Chr(34)
    'Anfang des Menü-XML
    menuXML = "<menu xmlns=" & q & "http://schemas.microsoft.com/office/2009/07/customui" _
    & q & ">"

    Set doc = ActiveDocument
    If doc.MailMerge.MainDocumentType <> wdNotAMergeDocument Then
        Set mmData = doc.MailMerge.DataSource
        For Each fieldName In mmData.FieldNames
            fName = fieldName.Name
            'Einzelne Menüeinträge
            menuXML = menuXML & "<button id=" & q & "sd_" & fName & q _
            & " label=" & q & fName & q & _
            " onAction=" & q & "SeriendruckfeldEinfügen" & q & _
            " tag=" & q & fName & q & " />"
        Next
    End If
    'Ende
    menuXML = menuXML & "</menu>"
    menuString = menuXML
End Sub

Sub SeriendruckfeldEinfügen(control As IRibbonControl)
    Dim rng As Word.Range
    Dim fieldName As String

    Set rng = Selection.Range
    fieldName = control.Tag
    ActiveDocument.Fields.Add rng, wdFieldMergeField, fieldName, False
End Sub

```

CD-ROM Der Beispielcode befindet sich in der Datei *Ribbon_Beispiel16_14.docm* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap16*. Dieses Dokument ist mit der Tabelle *Personal* der Datenbank *Nordwind.mdb* als Datenquelle verknüpft.

Wie das Beispiel veranschaulicht, werden neue Steuerelemente einem bestehenden, Word-eigenen Kontextmenü zugefügt. Folglich müssen wir den Id-Wert dieser Menüs wissen. Diese finden Sie in der Excel-Mappe *WordControls.xlsx* der Datei *Office2010ControlIDs.exe* nach den Id-Werten der Schaltflächen für das Menüband-XML (Abbildung 16.19).

Abbildg. 16.19 Id-Werte für Kontextmenüs befinden sich in der Datei *WordControls.xlsx*

Control Name	Control Type	Tab Set	Tab	Group/Context	Parent Control
3347 ContextMenuTextEdit	contextMenu	None (Context Menu)	None (Context Menu)		
3348 Cut	button	None (Context Menu)	None (Context Menu)	ContextMenuTextEdit	
3349 Copy	button	None (Context Menu)	None (Context Menu)	ContextMenuTextEdit	
3350 PasteGalleryMini	gallery	None (Context Menu)	None (Context Menu)	ContextMenuTextEdit	
3351 TextEditModeExit	button	None (Context Menu)	None (Context Menu)	ContextMenuTextEdit	
3352 FontDialog	button	None (Context Menu)	None (Context Menu)	ContextMenuTextEdit	
3353 ParagraphDialog	button	None (Context Menu)	None (Context Menu)	ContextMenuTextEdit	
3354 BulletsGalleryWord	gallery	None (Context Menu)	None (Context Menu)	ContextMenuTextEdit	
3355 ListLevelGallery	gallery	None (Context Menu)	None (Context Menu)	ContextMenuText BulletsGallery	
3356 BulletDefineNew	button	None (Context Menu)	None (Context Menu)	ContextMenuText BulletsGallery	
3357 NumberingGalleryWord	gallery	None (Context Menu)	None (Context Menu)	ContextMenuText	
3358 ListLevelGallery	gallery	None (Context Menu)	None (Context Menu)	ContextMenuText NumberingGal	
3359 DefineNewNumberForm	button	None (Context Menu)	None (Context Menu)	ContextMenuText NumberingGal	
3360 ListSetNumberingValue	button	None (Context Menu)	None (Context Menu)	ContextMenuText NumberingGal	
3361 ImeReconvert	button	None (Context Menu)	None (Context Menu)	ContextMenuTextEdit	
3362 HyperlinkInsert	button	None (Context Menu)	None (Context Menu)	ContextMenuTextEdit	
3363 HyperlinkEdit	button	None (Context Menu)	None (Context Menu)	ContextMenuTextEdit	
3364 HyperlinkOpen	button	None (Context Menu)	None (Context Menu)	ContextMenuTextEdit	
3365 HyperlinkRemove	button	None (Context Menu)	None (Context Menu)	ContextMenuTextEdit	

HINWEIS Die Datei *Office2010ControlIDs.exe* steht auf <http://www.microsoft.com/downloads/details.aspx?FamilyID=3f2fe784-610e-4bf1-8143-41e481993ac6&displaylang=en> bereit. Es befindet sich auf der CD-ROM im Ordner *Beilagen/Ribbon*.

Backstage



In Office 2007 wurde das Menü *Datei* durch die »Pizza«-Schaltfläche ersetzt (*Office*-Schaltfläche). Diese ist in wieder Office 2010 verschwunden. An dessen Stelle ist jetzt eine Registerkarte mit der alten Beschriftung *Datei* vorhanden. Was dahinter steckt, ist nur bedingt mit dem alten Menü zu vergleichen. Statt eine Registerkarte oder ein Menü wird ein Fenster eingeblendet (»Backstage«), welches das Dokument verdeckt. Diese Ansicht stellt hauptsächlich Befehle für die Dokumentverwaltung zur Verfügung, während diejenigen für die Dokumentbearbeitung sich im Menüband befinden.

Da die Backstage-Ansicht auch Teil der Office Fluent UI ist, ist im Hintergrund wie beim Menüband und Kontextmenüs Menüband-XML im Spiel. Auch hier befindet sich die Definition der Ansicht in der gleichen XML-Datei wie die des Menübands und der Kontextmenüs. In diesem Fall heißt das Hauptelement des Abschnitts <backstage>.

Für die Backstage-Ansicht wurde die Funktionalität des Menübands massiv ausgebaut, um aussagekräftige, benutzerfreundliche Umgebungen zu ermöglichen. Es ist uns leider nicht möglich, aufgrund des begrenzten Buchumfangs, mehr als eine kurze Einführung zu bieten.

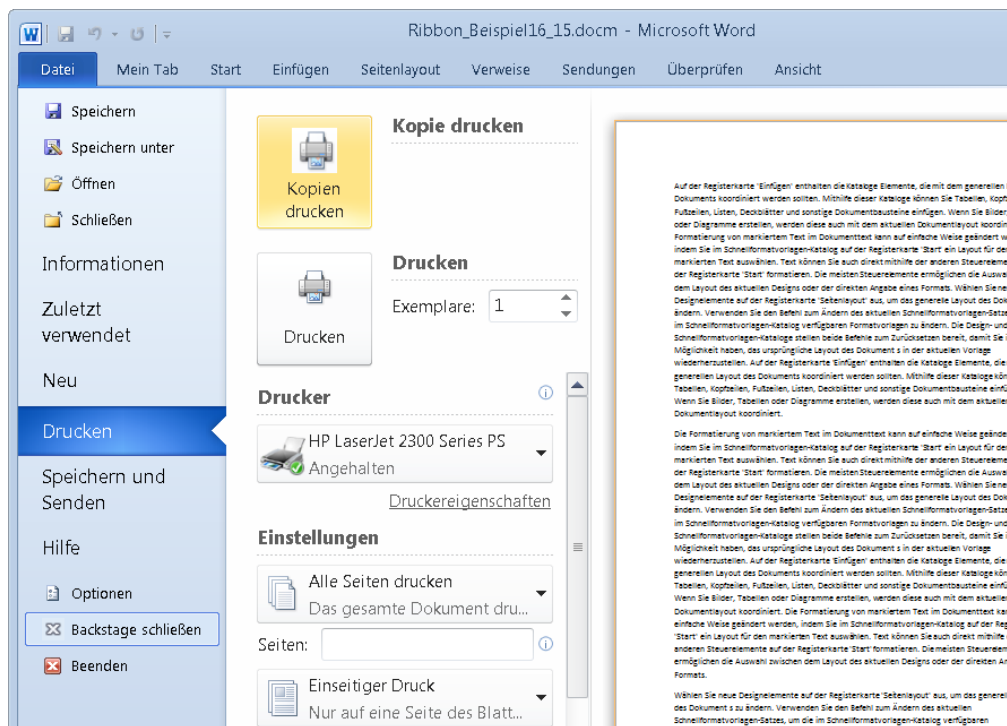
HINWEIS Ausführliche Information zur Backstage-Ansicht und dem dahinter stehenden XML finden Sie in den zwei Artikeln *Einführung in die Office 2010-Backstage-Ansicht für Entwickler* (<http://msdn.microsoft.com/de-de/library/ee691833.aspx>) und *Anpassen der Office 2010-Backstage-Ansicht für Entwickler* (<http://msdn.microsoft.com/de-de/library/ee815851.aspx>).

Auch hier untersteht der Entwickler ähnlichen Restriktionen bezüglich Word-eigenen Registerkarten und Gruppen. Selbstdefinierte können hinzugefügt und deren Reihenfolge festgelegt werden, aber die Reihenfolge und Inhalt von Words eigenen Steuerelementen können nicht geändert werden. Dazu kommt eine Begrenzung bei der Zahl der selbstdefinierten Registerkarten: maximal 255 sind erlaubt.

Die Backstage-Ansicht besteht aus mehreren räumlichen Teilen. Links aufgelistet sind die selbstständigen Befehle, wie *Speichern*, sowie die Überbegriffe, wie *Informationen*. In der mittleren Spalte befinden sich die dem Oberbegriff gehörenden Befehle, während die rechte Spalte ergänzende Informationen enthält.

Unser kleines Beispiel veranschaulicht, wie die Backstage-Ansicht durch selbstständige Befehle in der linken Spalte sowie eine Gruppe in einer Word-eigenen Registerkarte ergänzt wird.

Abbildg. 16.20 Die Registerkarte *Drucken* in der Backstage-Ansicht mit selbstdefinierter Gruppe



Den Befehl *Backstage-Ansicht schließen* haben wir der linken Spalte in Abbildung 16.20 zugefügt. Es ist nicht allen Benutzern klar, wie die Ansicht zu verlassen ist (durch Anklicken einer anderen Registerkarte oder Drücken der Taste `[ESC]`); oft probieren sie es über das rote »X« des Anwendungsfensters, was Word beendet. Eine Schaltfläche, die das Drücken der Taste `[ESC]` wiedergibt, hilft ihnen.

Zuoberst in der Registerkarte *Drucken* wurde eine Gruppe für den Druck einer »Kopie« zugefügt. Der Code fügt ein Wasserzeichen mit dem Text »Kopie« in die Kopfzeile des Dokuments ein, druckt es und entfernt anschließend das Wasserzeichen.

Das XML für diese Anpassungen finden Sie in Listing 16.19. Wie daraus ersichtlich, stehen die Elemente für die linke Spalte in der Ebene direkt unter dem Abschnittselement `<backstage>`.

Um eine Gruppe einer Registerkarte zuzufügen, müssen zuerst die Elemente für die Registerkarte, gefolgt von jenem der gewünschten Spalte, stehen. Steuerelemente innerhalb einer Gruppe müssen einer Auflistung zugewiesen werden, die die Position innerhalb der Gruppe festlegt. Das `<primaryItem>` mit einem `<button>`-Element entspricht der großen Schaltfläche links in der Gruppe. Dazu können eine Liste `<topItems>` definiert werden, die rechts daneben erscheinen, wie der Eintrag *Exemplare* der Word-eigenen Gruppe *Drucken*.

Listing 16.19 XML für die Anpassung der Backstage-Ansicht

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
  <backstage>
    <button id="bsSchließen" label="Backstage schließen" onAction="BackstageSchliessen"
      image="SchliessenWeiss" insertBeforeMso="FileExit" />
    <tab idMso="TabPrint" >
      <firstColumn>
        <group id="grpKopieDrucken" label="Kopie drucken"
          insertBeforeMso="GroupPrintSettings" >
          <primaryItem>
            <button id="KopieDrucken" label="Kopien drucken" image="Drucker"
              onAction="KopieDrucken" />
          </primaryItem>
        </group>
      </firstColumn>
    </tab>
  </backstage>
</customUI>
```

CD-ROM Der Beispielcode befindet sich in der Datei *Ribbon_Beiispiel16_15.docm* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap16*.

ACHTUNG Reihenfolge der Office Fluent UI Abschnitte

Ein einziges RibbonXML (`<customUI>`) kann eine beliebige Kombination von Anpassungen für die Word-Umgebung enthalten. Es darf also Einträge für Befehle (`<commands>`), das Menüband (`<ribbon>`), die Backstage-Ansicht (`<backstage>`) und/oder die Kontextmenüs (`<contextMenus>`) umfassen. Wichtig ist lediglich, dass diese in der richtigen Reihenfolge (wie hier angegeben) erscheinen, sonst ist das XML ungültig.

Zusammenfassung

In diesem Kapitel wurde die Office Fluent UI für die Menüband-Erweiterung, Kontextmenüs sowie die Backstage-Ansicht 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 Menüband-Lösungen erfolgreich zu realisieren.

In diesem Kapitel wurden die folgenden Themen behandelt:

- In einer kurzen Übersicht (Seite 728 ff.) wurden die Unterschiede zu Symbolleisten sowie Beweggründe für die Änderung diskutiert. Auch wurde die Rolle von Symbolleisten ab Word 2007 erwähnt.
- Die einzelnen Schritte, um eine Menüband-Erweiterung zu erstellen und sie in einem Word-Dokument einzubinden, wurden anhand eines einfachen Beispiels ab Seite 732 erläutert. Einige Werkzeuge für die Arbeit mit XML wurden vorgestellt.
- Der nächste Teil (Seite 738 ff.) veranschaulichte die erweiterte Funktionalität der Menüband-Erweiterung
- Ein kurzer Überblick der verschiedenen Steuerelemente und ihrer Attribute wurde anhand eines Beispiels erläutert (Seite 755 ff.).
- Schließlich wurden die neuen XML-Abschnitte für Kontextmenüs und die Backstage-Ansicht ab Seite 771 vorgestellt

HINWEIS

Nachfolgend sind noch einige Ressourcen im Internet zum Thema »Ribbon« aufgeführt:

- Jens Häupels Blog: <http://blogs.msdn.com/search/SearchResults.aspx?q=ribbon§ions=4510>
- 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 17

Tastaturbelegungen

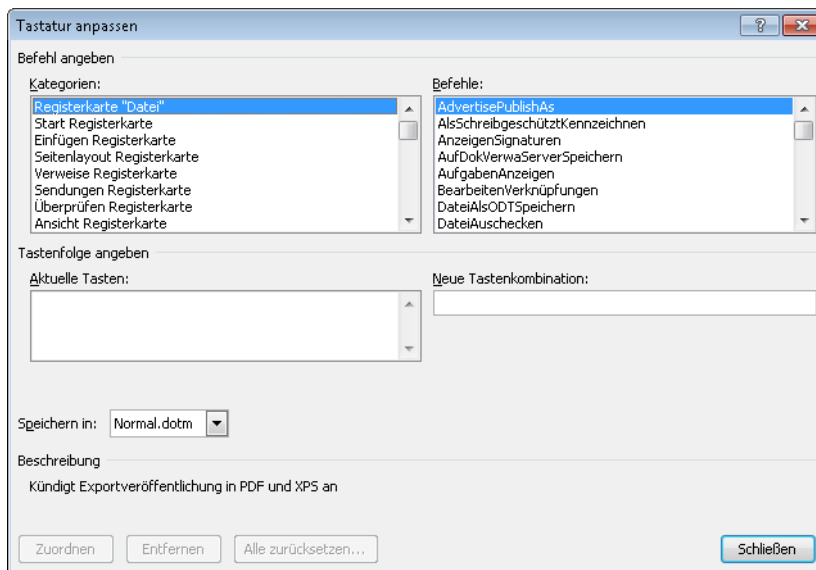
In diesem Kapitel:

Tastenbelegungen im Objektmodell	781
Tastenkombinationen verwalten	789
Tastenbelegung mit COM-Add-In verbinden	791
Zusammenfassung	792

Da Word eine Textverarbeitung ist, arbeiten viele ihrer 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 im Stein gemeißelt. So ziemlich alle Tastenbelegungen können angepasst werden. Damit kann der Entwickler seine Werkzeuge oder der Ersteller einer Dokumentvorlage wichtige Formatierungsbefehle und Bausteine zugänglich machen.

In der Benutzeroberfläche geschieht dies, indem Sie die Registerkarte *Menüband anpassen* unter *Datei/Optionen* wählen. Die Schaltfläche *Tastenkombinationen: Anpassen* blendet das Dialogfeld in Abbildung 17.1 ein. Hier 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.

Abbildg. 17.1 Fast jedem Befehl, Makro, Formatvorlage, Schriftart oder AutoText-Eintrag kann eine Tastenkombination zugewiesen werden



Es ist wichtig, den *Kontext* festzulegen. Wurde 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 *Alle Seiten drucken* im Bereich *Datei/Drucken* den Eintrag *Tastenbelegung*, dann klicken Sie auf OK. (In Word 2007 befindet sich der Name des Dropdownfelds im Dialogfeld *Drucken*.)


HINWEIS





Bekanntlich können Makros und Formatvorlagen (und früher AutoText-Einträge und Symbolleisten) über den Menübefehl und Dialogfeld *Entwickler Tools/Vorlagen/Dokumentvorlage/Organisieren* zwischen Word-Dokumenten kopiert werden. Es gibt jedoch kein Word-eigenes Werkzeug, mit dem sich auch Tastaturkürzel übertragen ließen. Unter <http://www.chris-woodman.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 17.1.

`Command` spezifiziert, was zu machen ist, beispielsweise der Befehl-, Makro- oder Formatvorlagenname (siehe auch Tabelle 17.1). 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.pdf* 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 17.1.) Benutzerdefinierte Prozeduren, die Parameter verlangen, stehen Tastenbelegungen nicht zur Verfügung.

Zudem befindet sich eine Spalte `WordBasic.[KeyMacro$]()` in Tabelle 17.1. Diese Funktion liefert für einige `KeyCategory` 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.dotm*; 1 für die angehängte Dokumentvorlage des aktuellen Dokuments

WordBasic.KeyMacro\$(1, 0) 'Beispiel-Ergebnis: Fett statt Bold

Tabelle 17.1 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.docx"	"Open test.docx"
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	"¼"	[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«. Die folgende Kommandozeile gibt den Wert »2« zurück, da das ganze KeyBinding ein Makro ausführt.</p> <p>MsgBox KeyBindings(1).KeyCategory</p> <p>Die folgende Kommandozeile gibt jedoch 7 zurück, da die Kombination [Alt] + [A] die erste Hälfte einer zerteiligen Belegung ist.</p> <p>MsgBox Application.FindKey(wdKeyAlt+wdKeyA).KeyCategory</p>		

Tabelle 17.1 Beispiele für die Eigenschaften *KeyCategory*, *Command* und *CommandParameter* (Fortsetzung)

KeyCategory	Command	Command-Parameter	WordBasic. [KeyMacro\$]()
wdKeyCategoryNil -1	<p>Diese KeyCategory-Eigenschaft gibt wdKeyCategoryNil (-1) zurück, wenn die Tastenkombination im fraglichen Kontext nicht zugewiesen wurde. Beispiel:</p> <p>Die Kombination Strg + ↵ + X 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 (o_Key.Execute), soll, analog wie in der Benutzerschnittstelle, nichts passieren, keine Fehlermeldung und keine Aktion erfolgt.</p>		
<p>Zum »?« in der Spalte »CommandParameter«: Wenn das erste Zeichen der CommandParameter-Eigenschaft eines »Symbol«-Command 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 AscW ermittelt wurde, gibt VBA ein Fragezeichen zurück. Das Gleiche gilt für das Ergebnis der Command-Eigenschaft des KeyCategory wdKeyCategorySymbol.</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
```

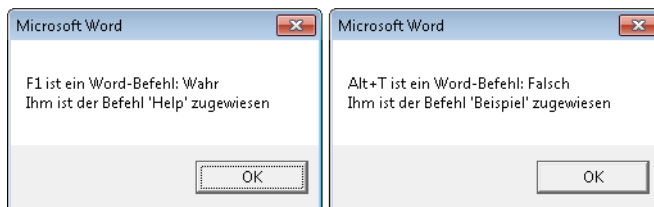
Tastaturkombinationen können auch mit zwei aufeinander folgenden Tastenfolgen definiert werden. In Word sind für internationale Zeichen einige Tastenkombinationen vordefiniert, wie `[Strg]+[']`, `[U]`, mit der das Zeichen »ú« eingefügt wird. Dann haben die Eigenschaften wie `FindKey` folgende Syntax:

```
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 17.1 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 17.1 ist in Abbildung 17.2 ersichtlich. Die Zeichenkette mit der Tastenkombination wurde durch die Eigenschaft `KeyString` des `KeyBinding`-Objekts erzeugt.

Abbildg. 17.2 Zwischen einer benutzerdefinierten Tastenbelegung und einem Word-Befehl unterscheiden



Listing 17.1 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
```

Listing 17.1 Feststellen, ob eine Tastenbelegung mit einem benutzerdefinierten oder Word-internen Befehl verbunden ist (Fortsetzung)

```
Set kb = Application.KeyBindings.Key(1KeyCode)
On Error GoTo 0
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.




CD-ROM Die Beispieldatei *Bsp17_01.dotm* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap17*.

Das Listing 17.2 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 Results ist in Abbildung 17.3 ersichtlich. Vergleichen Sie die Spalte »Kat.« mit den Angaben in Spalte »KeyCategory« der Tabelle 17.1.

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. 17.3 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	RemoveBulletsNumbers		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 ,  und . 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 17.2 enthalten).

Am Schluss wird die Zeichenkette sBefehlsliste in das aktuelle Dokument eingefügt und in eine Tabelle umwandelt. Diese wird nach den Spalten »Word-Befehl«, dann »Tastaturk.« sortiert (die benutzerdefinierte Tastenbelegungen stehen damit am Tabellenanfang).

Listing 17.2 Alle Tastenbelegungen eines Kontexts 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 17.2 Alle Tastenbelegungen eines Kontexts 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 17.2 Alle Tastenbelegungen eines Kontexts 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
```

CD-ROM Die Beispieldatei *Bsp17_02.dotm* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap17*.

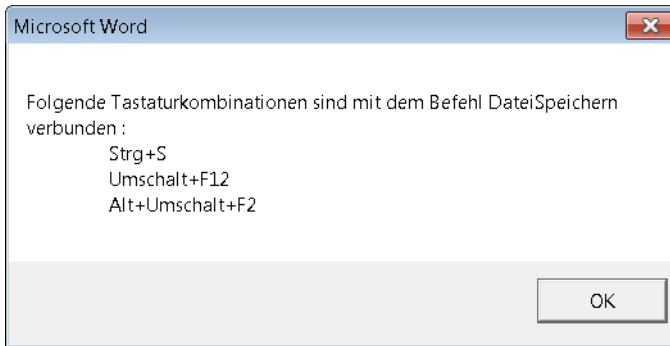
Tastenbelegungen eines Befehls ermitteln

Gelegentlich will man wissen, welche Tastenkombinationen, wenn überhaupt, mit einem bestimmten Befehl verbunden sind. Es wäre möglich, wie oben beschrieben, durch alle KeyBindings durchzuschleifen, und die Command-Eigenschaft zu prüfen. Dies wäre jedoch kompliziert und zudem relativ langsam.

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 17.3 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. 17.4 Die Tastaturbelegungen für den Befehl *DateiSpeichern*Listing 17.3 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

    Application.CustomizationContext = NormalTemplate
    sCommand = "FileSave"
    lKeyCategory = wdKeyCategoryCommand

    sList = "Folgende Tastaturkombinationen sind mit dem Befehl " & _
        "DateiSpeichern verbunden : "
    For Each kb In Application.KeysBoundTo(lKeyCategory, sCommand)
        sList = sList & vbCrLf & vbTab & kb.KeyString
    Next
    MsgBox sList
End Sub

```

CD-ROM

Die Beispieldatei *Bsp17_03.dotm* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap17*.

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 anstelle einer gesperrten im Menü erscheinen. 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 gültiger `CommandParameter` befindet sich in der Hilfe zu diesem Thema.

Diese Methode ermöglicht es, Tastenbelegungen zu definieren, die im Dialogfeld *Tastatur anpassen* nicht erlaubt sind, wie etwa Kombinationen mit den - und -Tasten.

HINWEIS

Um zu ermitteln, ob eine Taste oder Tastenkombination im Dialogfeld *Tastatur anpassen* benutzt werden darf, kann ihre `Protected`-Eigenschaft abgefragt werden.

Das folgende Codefragment zeigt, wie ein Satz von Tastenkombinationen erstellt wird, um eine Gruppe ähnlicher Bausteine einzufügen. Jede Kombination beginnt mit **[Alt]+[T]**, gefolgt von einer weiteren Taste. Bitte beachten Sie, dass damit allein **[Alt]+[T]** 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 unter <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 **[Alt]+[F]**, die gegenwärtig mit einem Makro verbunden ist, die Formatvorlage »FarbigerText« zuzuweisen:

```
Dim kb as Word.Keybinding
Set kb = Application.FindKey(BuildKeyCode(wdKeyAlt, wdKeyF))
kb.Rebind KeyCategory:=wdKeyCategoryStyle, Command:="FarbigerText"
```

Bitte beachten Sie, dass Sie durchaus die Methode Add benutzen dürfen, um eine bestehende Kombination mit einem anderen Befehl zu belegen. Rebind bietet sich an, wenn im Code bereits ein KeyBinding-Objekt vorliegt.

Tastenbelegung mit COM-Add-In verbinden

Bekanntlich können Tastenkombinationen nur öffentliche Prozeduren eines Word-Moduls zugewiesen werden. Es ist also nicht möglich, eine Tastenkombination unmittelbar mit einer Prozedur zu verbinden, die Teil eines COM-Add-Ins bildet. Dazu braucht es Code in der Word-Umgebung, die die Verbindung vermittelt: eine so genannte Callback-Prozedur.

Ein Beispiel dafür finden Sie im VSTO-Abschnitt über Add-Ins des Kapitels 10.

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 780)?
- Wie werden die Tastenkombinationen eines Kontextes ermittelt (Seite 783)?
- Wie lässt sich herausfinden, welche Tastenkombinationen einem bestimmten Befehl zugewiesen sind (Seite 788)?
- Wie werden Tastenbelegungen entfernt (Seite 789)?
- Wie werden neue Tastenkombinationen hinzugefügt (Seite 790)?

Kapitel 18

Word-Aufgabenbereiche

In diesem Kapitel:

Allgemeines zum Aufgabenbereich	794
Der programmtechnische Umgang mit Aufgabenbereichen (Task Panes)	795
Zusammenfassung	810

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. Der Aufgabenbereich bietet eine Alternative zu (nicht modalen) Dialogfeldern.

Dieses Konzept wurde inzwischen gut aufgenommen und 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 gab es dokumentspezifische Lösungen über die Smart Document-Funktionalität und die darauf basierenden VSTO-Werkzeuge (Kapitel 10). Dazu wurde Entwicklern in Office 2007 die `CustomTaskPane`-Schnittstelle zur Verfügung gestellt, die leider in der VBA-Umgebung nicht unterstützt wird. Das VSTO-Add-In-Werkzeug bindet jedoch diese Schnittstelle ein; die Erstellung eines eigenen Aufgabenbereichs damit wird in Kapitel 10 kurz vorgestellt. In Word 2010 gibt es in dieser Hinsicht nichts Neues zu berichten.

HINWEIS

Entwickler, die VSTO nicht einsetzen möchten, finden mehr Informationen über die `ICustomTaskPaneConsumer`-Schnittstelle unter <http://msdn.microsoft.com/en-us/library/aa338197.aspx>.

Was den ersten Teil dieser Frage angeht – Word-eigene Aufgabenbereiche den eigenen Bedürfnissen anzupassen – stehen uns nur beschränkte Möglichkeiten zur Verfügung. Seit ihrer Einführung wurde die Funktionalität in jeder neuen Version von Word mehr oder minder geändert. Am Anfang konnte beispielsweise nur ein einzelner Aufgabenbereich eingeblendet werden, dessen Inhalt (Aufgabe) sich dem vom Benutzer ausgeführten Befehl angepasst hat. Seit Word 2007 haben Befehle ihren eigenen Aufgabenbereich und es können mehrere gleichzeitig ins Word-Fenster eingeblendet werden. Es wurden über die Jahre auch Aufgabenbereiche entfernt und neue eingeführt.

Dieses Kapitel fasst die zur Verfügung stehende Funktionalität für Word 2007 und 2010 zusammen.

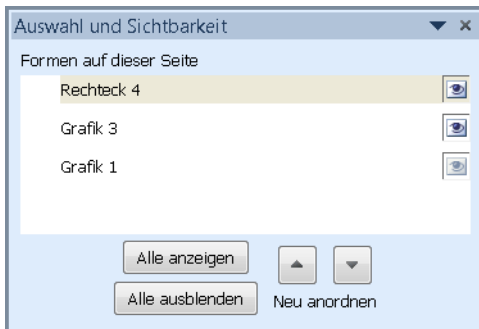
Allgemeines zum Aufgabenbereich

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 oder vom Entwickler 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 »sieht« der Makrorekorder 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 19).

Die Ähnlichkeit zu einer üblichen Symbolleiste ist in Abbildung 18.1 unverkennbar. Das einzige Steuerelement erstreckt sich über die ganze Symbolleiste. Dessen Inhalt ist grundsätzlich HTML-Code, der seinerseits weitere ActiveX-Steuerelemente definiert, um die Skripts auszuführen. Aufgabenbereiche dürfen rechts oder links (und manchmal oben oder unten) neben anderen Aufgabenbereichen andockt werden oder freistehend über dem Dokumentfenster liegen. Bei freistehenden Fenstern kann die Größe frei angepasst werden.

Abbildg. 18.1 Ein neuer Aufgabenbereich in Word 2010



Der programmtechnische Umgang mit Aufgabenbereichen (Task Panes)

Wenn wir in diesem Kapitel von *Task Panes* reden, meinen wir damit ein `CommandBar.Control` des Typs `msoControlWorkPane`, das den Inhalt der einzelnen Aufgabenbereiche definiert. Es gibt zwei Arten von Aufgabenbereichen; diejenige, wofür Microsoft einen `WdTaskPanes`-Wert definiert hat, und alle anderen, die nicht so direkt angesprochen werden können.

Sie können einen Aufgabenbereich über dessen Bezeichnung oder seinen `WdTaskPanes`-Konstantwert ansprechen, wie Listing 18.1 veranschaulicht. Das `TaskPane`-Objekt hat nur wenige Eigenschaften, wovon `Visible` die einzige allgemein nützliche ist. Um Aufgabenbereiche zu positionieren, brauchen Sie die Eigenschaften des `CommandBars`-Objektmodells.

Einen bestimmten Aufgabenbereich einblenden

Falls Sie einen Aufgabenbereich einblenden möchten, empfiehlt es sich, dies über das `TaskPane`-Objekt zu tun, da die `Visible`-Eigenschaft des `CommandBar`-Objekts nur dann zur Verfügung steht, wenn der Aufgabenbereich während der laufenden Word-Sitzung schon einmal eingeblendet wurde.

Listing 18.1 Einen Aufgabenbereich einblenden und positionieren

```
Sub AufgabenbereicheManipulieren()
    Dim tp As Word.TaskPane
    Dim cb As Office.CommandBar

    Set tp = Application.TaskPanes(wdTaskPaneNav)
    tp.Visible = True
    Application.TaskPanes(wdTaskPaneFormatting).Visible = True
    Set cb = Application.CommandBars("Styles")
    'Funktioniert nur, wenn der Aufgabenbereich während der Sitzung
    'schon eingeblendet wurde, entweder vom Benutzer oder
    'über Application.TaskPanes(WdTaskPane).Visible = True
    cb.Visible = True
    cb.Position = msoBarFloating
    cb.Left = 12
End Sub
```

Listing 18.1 Einen Aufgabenbereich einblenden und positionieren (Fortsetzung)

```

    cb.Height = 200
    cb.Width = 200
End Sub

```

Die Tabelle 18.1 listet die Aufgabenbereiche auf mit ihren Bezeichnungen, ihren WdTaskPanes-Typen, deren Werten und zu welchem Kontext er gehört.

PROFITIPP

Es fällt Ihnen bestimmt auf, dass für die Aufgabenbereiche *ClipArt* sowie *Zwischenablage* kein WdTaskPanes-Konstantwert vorhanden ist. Die Zwischenablage kann mit der Methode `Application.ShowClipboard` oder `Application.CommandBars.ExecuteMso "ShowClipboard"` eingeblendet werden. Für den Aufgabenbereich für ClipArt steht einzig die allgemeine Methode `ExecuteMso` zur Verfügung:
`Application.CommandBars.ExecuteMso "ClipArtInsert"`

Tabelle 18.1 Die von Word zur Verfügung gestellten Aufgabenbereiche mit Informationen für den programmtechnischen Umgang

Beschriftung	Bezeichnung (Name-Eigenschaft)	WdTaskPanes-Enumerator	Wert	Kontext
<i>ClipArt</i>	<i>Clip Art</i>	[kein]	[kein]	ClipArt einfügen
<i>Zwischenablage</i>	<i>Office Clipboard</i>	[kein]	[kein]	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>	<i>Document Actions</i>	wdTaskPane-DocumentActions	7	Eine Smart Document-Lösung ist aktiv
<i>Formatierung und Bearbeitung einschränken</i>	<i>Restrict Formatting and Editing</i>	wdTaskPane-Documen-t-Protection	6	Schnittstelle für die Festlegung der Schutzart für ein Dokument und aktiviert diesen Schutz
<i>Dokumentaktualisierungen</i>	<i>Document Updates</i>	wdTaskPane-Documen-tUpdates	13	Das aktive Dokument ist eine Kopie eines im Dokument-arbeitsbereich vorhandenen Dokuments
<i>Dokumentverwaltung</i>	<i>Document Management</i>	wdTaskPane-Documen-t-Management	16	Stellt Features einer Dokument-arbeitsbereich-Website eines SharePoint-Services zur Verfügung
<i>Faxdienste</i>	<i>Fax Service</i>	wdTaskPane-FaxService	11	Eine Faxdienstleistung ist verfügbar
<i>Formatvorlagen</i>	<i>Styles</i>	wdTaskPane-Formatting	0	Stellt eine Liste von Formatvorlagen und im Dokument vorhandenen Formatierungen zur Verfügung

Tabelle 18.1 Die von Word zur Verfügung gestellten Aufgabenbereiche mit Informationen für den programmtechnischen Umgang (Fortsetzung)

Beschriftung	Bezeichnung (Name-Eigenschaft)	WdTaskPane-Enumerator	Wert	Kontext
Formatinspektor		wdTaskPane-StyleInspector	15	Bietet Einsicht in die Formatierungseigenschaften der gegenwärtigen Markierung und ermöglicht deren Anpassung
Formatvorlage übernehmen	Apply Styles	wdTaskPane-ApplyStyle	17	Dient als Ersatz für die Combobox in der früheren Symbolleiste <i>Formatierung</i>
Hilfe	Help	wdTaskPaneHelp	9	Seit Word 2007, blendet das <i>Hilfe</i> -Fenster ein
Seriendruck	Mail Merge Panes	wdTaskPane-MailMerge	2	Enthält sechs Schritte, die den Benutzer durch die Erstellung eines Seriendrucks führt. Kann teilweise mit VBA beeinflusst werden (siehe Kapitel 7).
Recherchieren	Research	wdTaskPane-Research	10	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 http://msdn2.microsoft.com/en-us/library/aa159647(office.11).aspx .
Formatierungen anzeigen	Reveal Formatting	wdTaskPane-Reveal-Formatting	1	Zeigt an, mit welchen Formatierungen die Markierung formatiert wurde und ob diese direkt oder von einer bestimmten Formatvorlage stammen
Einfache Suchoptionen		wdTaskPane-Search	4	In Word 2002 und 2003 war dies eine Schnittstelle, um nach Dateien zu suchen. Der Konstantwert befindet sich noch im Objektmodell, verursacht jedoch die Fehlermeldung »Dieser Befehl ist nicht verfügbar«.
Freigegebene Arbeitsbereiche		wdTaskPane-SharedWorkspace	8	Verwaltung von Dokumenten, die auf einer SharePoint-Seite gespeichert sind

Tabelle 18.1 Die von Word zur Verfügung gestellten Aufgabenbereiche mit Informationen für den programmtechnischen Umgang (Fortsetzung)

Beschriftung	Bezeichnung (Name-Eigenschaft)	WdTaskPane-Enumerator	Wert	Kontext
<i>Recherchieren</i>	<i>Research</i>	wdTaskPane-Translate	3	Der Arbeitsbereich <i>Recherchieren</i> im Übersetzungsmodus
<i>Signaturen</i>	<i>Signatures</i>	wdTaskPane-Signature	14	Listet die digitalen Signaturen, mit denen ein Dokument signiert wurde, auf
<i>XML-Dokument</i>	<i>XML Document</i>	wdTaskPane-XMLDocument	12	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 22).
<i>XML-Struktur</i>	<i>XML Structure</i>	wdTaskPaneXML-Structure	5	Zeigt die Benutzerdefinierten XML-Tags in einem Dokument als Baumstruktur an sowie eine Liste der XML-Tags im angehängten Schema, die ins Dokument eingefügt werden können (siehe auch Kapitel 22)
<i>Zugriffsprüfung</i>	<i>Accessibility Checker</i>	[kein]	[kein]	Entspricht dem Schalter <i>Barrierefreiheit überprüfen</i> in <i>Datei/Information/Auf Probleme überprüfen</i> . Zeigt Ergebnisse der Prüfung für Dokumentinhalt, der Personen mit Behinderungen nicht zugänglich sein könnte.
<i>Anlageoptionen</i>	<i>Attachment Options</i>	[kein]	[kein]	Nur relevant für Word in der Outlook-Umgebung
<i>Besprechungsarbeitsbereich</i>	<i>Meeting Workspace</i>	[kein]	[kein]	Nur relevant für Word in der Outlook-Umgebung

Einen Aufgabenbereich positionieren

Aufgabenbereiche werden über das CommandBars-Objektmodell positioniert (siehe Kasten). Mittlerweile dürfen sie frei auf dem Bildschirm verschoben werden – auch außerhalb des Word-Fensters. Sie können am linken sowie am rechten Rand des Word-Fensters angedockt werden. Da der Inhalt der meisten Aufgabenbereiche senkrecht angeordnet ist, gibt Word eine Fehlermeldung zurück, wenn über Code versucht wird, sie oben oder unten anzudocken.

Das *CommandBars*-Objektmodell

Seit in Word 2007 das Menüband (auch als Multifunktionsleiste bekannt) die Symbolleisten als Benutzerschnittstelle ersetzte, spielt das *CommandBars*-Objektmodell keine führende Rolle mehr bei der programmtechnischen Anpassung von Words Benutzerschnittstelle. Deshalb wird ihm in dieser dritten Auflage kein eigenes Kapitel gewidmet. Dennoch übernimmt es einige wichtige Aufgaben bezüglich Aufgabenbereiche und des Menübands. Hier also eine kurze Einführung zu den Eigenschaften und Methoden des *CommandBars*-Objektmodells in Zusammenhang mit Aufgabenbereichen.

Positionierung

Die Positionierung eines Aufgabenbereichs wird durch die Eigenschaften *Left*, *Position*, *RowIndex* und *Top* des entsprechenden *CommandBar*-Objekts festgelegt bzw. ermittelt. Alle, mit Ausnahme von *Position*, verwenden Ganzzahlwerte.

Position erwartet einen Konstantwert des Typs *MsoBarPosition*. Diese Enumeration ist in Tabelle 18.2 aufgelistet. Die Eigenschaft legt fest oder gibt zurück, am welchen Rand des Anwendungsfensters der Aufgabenbereich sich befindet oder ob er frei positionierbar ist. Der Einfluss der Eigenschaften *Height*, *Left* und *Top* ist nur dann sichtbar, wenn die *Position* eines Aufgabenbereichs einen Wert von *msoBarFloating* hat.

Tabelle 18.2 *MsoBarPosition* legt die Positionierung des Aufgabenbereichs fest

MsoBarPosition-Enum	Wert	Beschreibung
<i>msoBarBottom</i>	3	Die Befehlsleiste wird am unteren Rand des Anwendungsfensters verankert
<i>msoBarFloating</i>	4	Die Befehlsleiste befindet sich über dem Anwendungsfenster
<i>msoBarLeft</i>	0	Die Befehlsleiste wird an der linken Seite des Anwendungsfensters verankert
<i>msoBarMenuBar</i>	6	Die Befehlsleiste ist eine Menüleiste (nur Macintosh)
<i>msoBarPopup</i>	5	Die Befehlsleiste ist ein Kontextmenü
<i>msoBarRight</i>	2	Die Befehlsleiste wird an der rechten Seite des Anwendungsfensters verankert
<i>msoBarTop</i>	1	Die Befehlsleiste wird am oberen Rand des Anwendungsfensters verankert

RowIndex

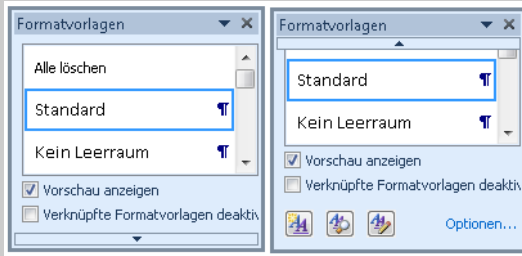
Die Reihenfolge nebeneinander stehender Aufgabenbereiche wird durch diese Eigenschaft beeinflusst bzw. kann dadurch teilweise ermittelt werden.

Größe

Auch die Eigenschaften *Width* und *Height* erwarten eine Ganzzahl. In Beziehung auf Aufgabenbereiche hat *Width* als Mindestwert die Ganzzahl 106. Obwohl kleinere Werte ohne Fehlermeldung akzeptiert werden, wird der Aufgabenbereich nicht schmaler. Es ist also nicht möglich, einen Aufgabenbereich unsichtbar zu machen, indem die Breite auf Null (0) festgelegt wird.

Ähnliches gilt für Height. Hier ist der untere Grenzwert 134. Falls nicht der ganze Inhalt sichtbar ist, erscheint am unteren bzw. oberen Rand ein Pfeil, womit gescrollt werden kann (Abbildung 18.2).

Abbildg. 18.2 Eine Pfeil erscheint, um den Inhalt eines zu kleinen Aufgabenbereichs durchzublätern



Sichtbarkeit

Über die Eigenschaft `Visible` kann ein Aufgabenbereich ein- bzw. ausgeblendet werden. Allerdings wird eine Laufzeit-Fehlermeldung eingeblendet, wenn der Aufgabenbereich in der laufenden Word-Sitzung noch nicht angesprochen wurde. Falls ein `WdTaskPane`-Konstantenwert für den Aufgabenbereich zur Verfügung steht, ist es besser, den Aufgabenbereich über die `Visible`-Eigenschaft des `TaskPane`-Objekts einzublenden.

Inhalt eines Aufgabenbereichs aktualisieren

Falls Sie programmtechnisch den Inhalt eines Aufgabenbereichs aktualisieren, werden diese Änderungen nicht immer automatisch wiedergeben. Um eine Aktualisierung zu erzwingen, blenden Sie den Aufgabenbereich aus und anschließend wieder ein:

```
With CommandBars("[Bezeichnung]")
    .Visible = False
    .Visible = True
End With
```

CD-ROM Die Beispieldatei *Bsp18_01.docm* enthält Beispielcode für die Einblendung von Aufgabenbereichen. Sie befindet sich auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap18*.

Fehlermeldungen

In Tabelle 18.3 finden Sie einige Fehlermeldungen, die auftauchen könnten, wenn Sie mit dem `WorkPane`-Objekt arbeiten. Diese gelten für alle Aufgabenbereiche und sind in der Arbeitsmappe *errormsg.xls* des Office Resource Kit nicht dokumentiert, da sie aus externen Bibliotheken stammen, hauptsächlich der von InfoPath.

ACHTUNG Achten Sie insbesondere auf die Fehlermeldungen mit der gleichen Nummer!

Tabelle 18.3 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 wdTaskPaneDocumentActions 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 wdTaskPaneXMLDocument 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 TaskPanes -Indexwert existiert nicht
6162	Dieser Befehl ist außerhalb des Fax-/E-Mail-Umschlags nicht verfügbar	Es wurde versucht, in der falschen Umgebung den Aufgabenbereich wdTaskPaneFaxService einzublenden
-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, voraussehbaren Fehler auszuweichen, anstatt sie abzufangen. Die Tests in Tabelle 18.4 sind nützlich, um festzustellen, ob die Umgebung gewisse Aufgabenbereiche unterstützt.

Tabelle 18.4 Prüfen, ob die Umgebung das Anzeigen eines Aufgabenbereichs unterstützt

Funktionalität	Prüfen mit
Geöffnetes Dokument vorhanden	<code>Application.Documents.Count > 0</code>
SharePoint vorhanden	<code>ActiveDocument.SharedWorkspace.Connected</code>
XML im Dokument	<code>ActiveDocument.XMLNodes.Count > 0</code> Dieser Test ist, nach dem Gerichtsentscheid in Sache XML-Knotenpunkte auf der Dokumentfläche anzuzeigen, nur von bedingtem Nutzen. Da es jedoch weiterhin möglich bleibt, XML-Knotenpunkte während einer Sitzung ins Dokument einzufügen, wird der Eintrag belassen.
XSL- und XSD-Unterstützung vorhanden	<code>Application.ArbitraryXMLSupportAvailable</code> <code>ActiveDocument.XMLSchemaReferences.Count > 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>

Aufgabenbereiche mit VBA abfangen oder sie außer Kraft setzen

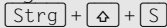
Die meisten Befehle, die Aufgabenbereiche ein- oder ausblenden, sind in einem Word VBA-Projekt abfangbar: Unter Umständen können Sie auch eine Alternative anbieten. Diese Möglichkeiten sind in Tabelle 18.5 vorgestellt.

Tabelle 18.5 Die Menübefehlsfolgen (alphabetisch nach Menüpunkt), die einen Aufgabenbereich einblenden, und der VBA-Code, um die Handlung abzufangen

Befehlspfad	Codeschnipsel
Dialoglauncher der Gruppe <i>Clipboard</i> in der Registerkarte <i>Start</i>	'Fängt den Menübefehl ab Sub EditOfficeClipboard() End Sub
<i>Ansicht/Dokumentaktionen</i> (wenn eine Smart Document-Lösung aktiv ist)*	'Fängt den Befehl ab Sub DocumentActionsPane() End Sub
<i>Dokumentaktualisierungen</i> (aus der Liste im Aufgabenbereich)	'Fängt den Befehl ab Sub ShowSmPane() End Sub
<i>Sendungen/Seriendruck starten/Seriendruck-Assistent mit Schritt-für-Schritt-Anweisungen</i>	'Den alten Seriendruckassistent statt 'des Aufgabenbereichs einblenden Sub MailMergeWizard() Dialogs(wdDialogMailMergeHelper).Show End Sub
<i>Überprüfen/Dokumentüberprüfung/Recherchieren</i>	'Fängt den Befehl ab Sub Research() End Sub Sub ResearchLookup End Sub
<i>Überprüfen/Dokumentüberprüfung/Übersetzen</i>	'Fängt den Befehl ab Sub Translate() End Sub Sub TranslatePane() End Sub
<i>Überprüfen/Dokumentüberprüfung/Thesaurus</i>	'Das alte Dialogfeld einblenden Sub ToolsThesaurusRR() Dialogs(wdDialogToolsThesaurus).Show End Sub
<i>Internet Faxdienst unter Datei/Freigeben</i>	'Fängt den Befehl ab Sub FaxService() End Sub

* Smart Dokument-Lösungen basieren auf die XML-Technologie, die wegen eines Gerichtsentscheids missbilligt wurde. Diese werden in Word 2003 und europäischen Versionen von Word 2007 noch lauffähig sein; in Word 2010 jedoch nicht mehr.

Tabelle 18.5 Die Menübefehlsfolgen (alphabetisch nach Menüpunkt), die einen Aufgabenbereich einblenden, und der VBA-Code, um die Handlung abzufangen (*Fortsetzung*)

Befehlspfad	Codeschnipsel
Dialoglauncher der Gruppe <i>Start/Formatvorlagen</i>	'Das Dialogfeld Formatvorlage anstelle des Aufgabenbereichs einblenden Sub FormattingPane() Dialogs(wdDialogFormatStyle).Show End Sub
Der Befehl <i>Formatvorlage übernehmen</i> , der der QAT zugewiesen werden kann, bzw. die Tastenkombination 	'Fängt den Befehl ab Sub StyleApplyPane() End Sub
Die Befehlsfolge <i>Vorbereiten/Signaturen anzeigen</i> im Menü der <i>Office</i> -Schaltfläche, sofern das Dokument digital signiert ist	'Fängt den Befehl ab Sub ViewSignatures() End Sub

CD-ROM Die Beispieldatei *Bsp18_02.docm* enthält einige der Prozedurskelette der Tabelle 18.5. Sie befindet sich auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap18*.

Der Aufgabenbereich *Formatvorlagen*

Der einzige Word-eigene Aufgabenbereich in den Versionen 2007 und 2010, dessen Inhalt über das Objekt-Modell angepasst werden kann, ist der, der Formatvorlagen auflistet. 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, sodass sie übersichtlich bleibt?

Den Aufgabenbereich in Word 2007 sowie Word 2010 entnehmen Sie bitte der Abbildung 18.3. 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. 18.3 Der Aufgabenbereich für Formatvorlagen kann programmtechnisch angepasst werden

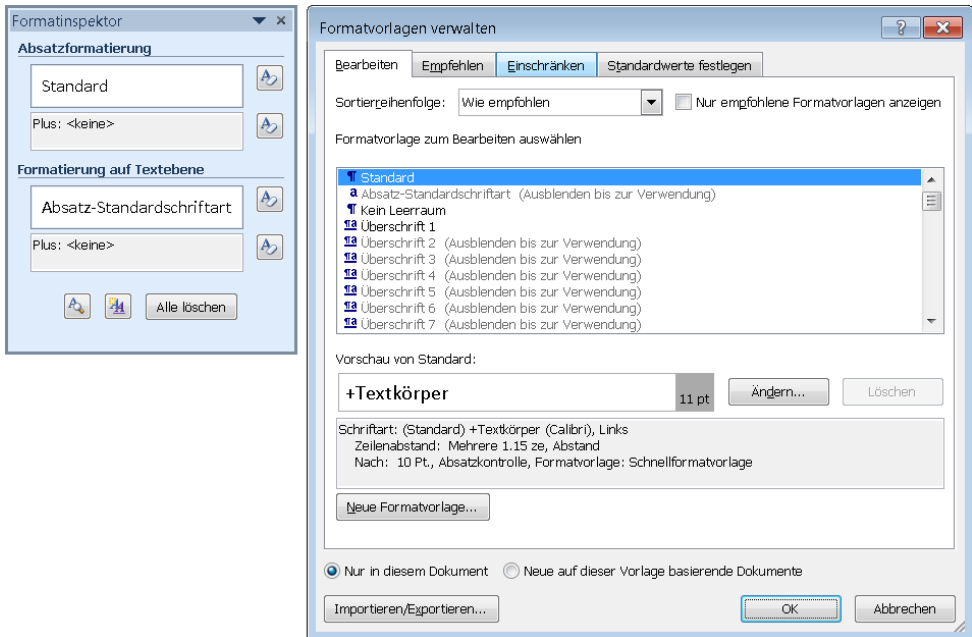


Die drei Schaltflächen sowie der Link blenden weitere Aufgabenbereiche und Dialogfelder ein (von links nach rechts)

- *Neue Formatvorlage*
- *Formatinspektor* (Abbildung 18.4, links), womit die direkte Formatierung eines Absatzes oder einer Textstelle gezielt entfernt werden kann
- *Formatvorlagen verwalten* (Abbildung 18.4, 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 18.5)

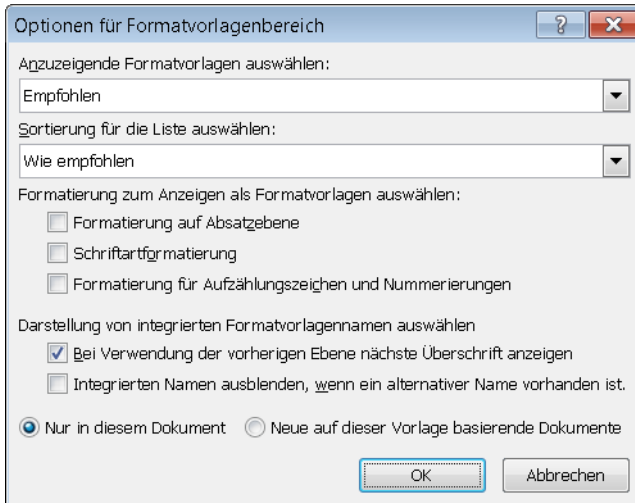
Abbildg. 18.4

Werkzeuge für die Arbeit mit Formatvorlagen in Word 2007 und Word 2010



Abbildg. 18.5

Festlegen, welche Einträge im Aufgabenbereich *Formatvorlagen* sichtbar sind



HINWEIS

Mehr Informationen zu den Möglichkeiten dieser Schnittstellen für die Verwaltung von Formatierungen finden Sie im Microsoft Press-Buch »Office 2007 – Das Profibuch« (ISBN-13: 978-3-86645-634-1).

Im Gegensatz zu Word 2003 und früher hat der Anwender nun 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

Wenn Sie diese Dialogfelder betrachten und sich der Möglichkeiten bewusst sind, ist es verständlich, dass die Handhabung sich etwas komplex gestaltet. Bis man damit vertraut ist, stimmt diese Aussage vor allem, wenn die Anpassung programmtechnisch erfolgen soll.

Optionen

Die Tabelle 18.6 listet die benötigten Eigenschaften auf, um die Optionen in Abbildung 18.5 über das Objektmodell zu ändern. Bitte beachten Sie, dass alle diese Optionen sich auf ein Dokument-Objekt beziehen und nicht auf die Word-Anwendung.

Wenn Sie beispielsweise den Eintrag *Alle löschen* aus der Liste verbannen möchten, sollte es mit dieser Codezeile klappen:

```
ActiveDocument.FormattingShowClear = False
```

Tabelle 18.6 Eigenschaften, die das Aussehen des Aufgabenbereichs *Formatvorlagen* beeinflussen

Dialogfeldeinstellung	Eigenschaft	Datentyp
"Alle löschen" anzeigen	Document.FormattingShowClear	Boolean
Anzuzeigende Formatvorlagen auswählen	Document.FormattingShowFilter	WdShowFilter (siehe Tabelle 18.7)
Schriftartformatierung	Document.FormattingShowFont	Boolean
Formatierung auf Absatzebene	Document.FormattingShowParagraph	Boolean
Formatierung für Aufzählungszeichen und Nummerierung	Document.FormattingShowNumbering	Boolean
Bei Verwendung der vorherigen Überschriftenebene die nächste Überschriftenebene anzeigen	Document.FormattingShowNextLevel	Boolean
Integrierte Namen ausblenden, wenn ein alternativer Name vorhanden ist	Document.FormattingShowUserStyleName	Boolean
Datei/Optionen/Erweitert/Formatierung mitverfolgen	Options.FormatScanning	Boolean

Tabelle 18.7 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

Soviel zu den allgemeinen Befehlen für den Aufgabenbereich. Wie steht's mit der Auflistung der Formatvorlagen?

Listen-
inhalt

Das Style-Objekt hat eine *Visibility*-Eigenschaft, die die Sichtbarkeit des Eintrags einer Formatvorlage im Aufgabenbereich festlegt. Kurioserweise wird der Eintrag aufgeführt, wenn *Visibility* = *False*, und wird unterdrückt, wenn *Visibility* = *True*. In der Benutzerschnittstelle entspricht sie den Schaltflächen *Anzeigen* (*Visibility* = *False*) sowie *Ausblenden* (*Visibility* = *True*) im Dialogfeld *Formatvorlagen verwalten*. Das Listing 18.2 veranschaulicht die programmtechnische Voreinstellung des Aufgabenbereichs *Formatvorlagen und Formatierungen*.

WICHTIG

In Word 2007 bleibt die Eigenschaft *Visibility* verborgen (mehr über verborgene Mitglieder des Objektmodells lesen Sie im Kapitel 1).

Listing 18.2 Den Inhalt des Aufgabenbereichs *Formatvorlagen und Formatierungen* festlegen

```
Public Sub FormatvorlagenListeAnpassen()
    Dim doc As Word.Document

    Set doc = ActiveDocument

    'Nur die erwünschten Formatvorlagen anzeigen
    With doc.Styles
        .Item(wdStyleBodyText).Visibility = False
        .Item(wdStyleListBullet).Visibility = False
        .Item(wdStyleListBullet2).Visibility = 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
```

CD-ROM Die Beispieldatei *Bsp18_03.dotm* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap18*.

Neben der »Ein-Aus«-Möglichkeit, die *Visibility* bietet, können Einträge auch bedingt in der Liste angezeigt werden. Diese Eigenschaft trägt die Bezeichnung *UnhideWhenUsed*. 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 »True« festgelegt, bleibt der Eintrag im Aufgabenbereich sichtbar, außer *Visibility* wird erneut auf »Wahr« festgelegt.

WICHTIG Auch *UnhideWhenUsed* ist im Objektmodell von Word 2007 verborgen.

Eine verborgene Eigenschaft kontrolliert die Sichtbarkeit einer Formatvorlage: *Hidden*. Sie kommt zwar auch im Objektmodell von Word 2003 vor, zeigt dort jedoch keine Wirkung. Ab Word 2007 hingegen wird bei der Festlegung *Hidden* = *True* die Formatvorlage aus jeglicher Liste, auch denjenigen im Dialogfeld *Formatvorlagen verwalten*, verbannt.

Die Eigenschaft *Priority* (legt die Priorität der Eintrag in einer Liste fest) 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 die Sortierreihenfolge kann über das Objektmodell festgelegt werden, und zwar durch die Verwendung der Eigenschaft *Document.StyleSort*. Die verfügbaren Werte entnehmen Sie bitte der Tabelle 18.8.

Tabelle 18.8 *WdStyleSort*-Werte für die 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 18.3 veranschaulicht einige der hier vorgestellten 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 18.3 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
```

Listing 18.3 Die Formatvorlagen für das aktuelle Dokument einrichten (Fortsetzung)

```

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

Function FV_U2_Erstellen(doc As Word.Document) As Word.Style
    'Ähnlicher Code wie für die Funktion FV_U1_Erstellen folgt.
End Function

```

Listing 18.3 Die Formatvorlagen für das aktuelle Dokument einrichten (*Fortsetzung*)

```
Function FV_U3_Erstellen(doc As Word.Document) As Word.Style  
    'Ähnlicher Code wie für die Funktion FV_U1_Erstellen folgt.  
End Function
```

CD-ROM Die Dokumentvorlage *Bsp18_04.dotm* befindet sich auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap18*. Kopieren Sie die Datei in den *StartUp*-Ordner, um die Formatvorlageneinstellungen in einem beliebigen Dokument vorzunehmen.

Zusammenfassung

Aufgabenbereiche stellen eine interessante Alternative zu traditionellen Menüs und Dialogfeldern dar. Leider werden dem VBA-Entwickler nur minimale Schnittstellen für deren Automatisierung zur Verfügung gestellt. Dieses Kapitel veranschaulichte die vorhandene Funktionalität und ihre Möglichkeiten. Unter anderem wurde Folgendes vorgestellt:

- Die hinter dem Aufgabenbereich stehende Technologie (Seite 794)
- Die programmtechnische Verwaltung von Aufgabenbereichen (Seite 795)
- Welche Schnittstellen das Word-Objektmodell für »Work Panes« zur Verfügung stellt (Seite 798)
- Von Aufgabenbereichen verursachte Fehlermeldungen (Seite 800)
- Wie der Inhalt des Aufgabenbereichs *Formatvorlagen* angepasst wird (Seite 803)

Kapitel 19

Interne Word-Befehle übersteuern

In diesem Kapitel:

Word-Befehl außer Kraft setzen

812

Zusammenfassung

817

Bereits in Kapitel 1 wurde dargestellt, wie sich Word verhält, wenn mehrere Makros mit gleichem Namen aufeinandertreffen. Welches der Makros dann aktiviert wird, ist durch eine klar definierte Hierarchie geregelt.

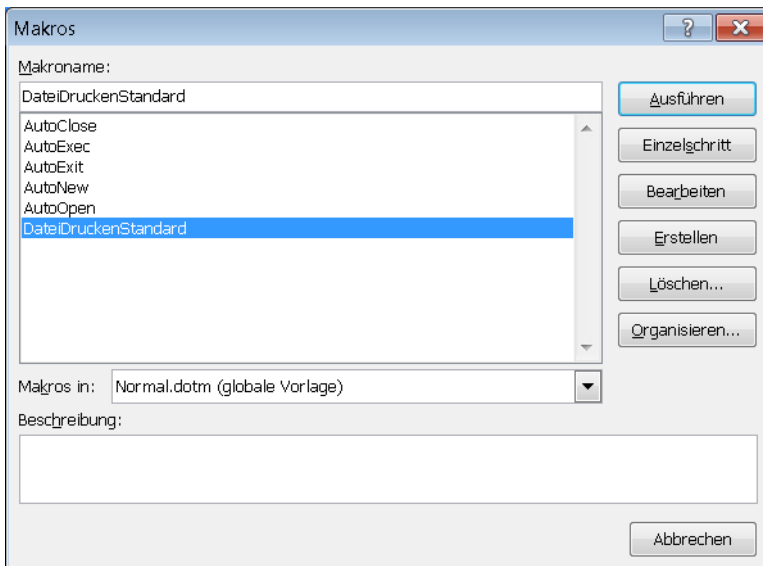
Dieser Hierarchie muss eigentlich noch eine weitere Ebene hinzugerechnet werden, nämlich die der »internen Word-Befehle«, deren Verwendungsmöglichkeiten das vorliegende Kapitel gewidmet ist.

Word-Befehl außer Kraft setzen

Bei der Entwicklung von Word wurde weitestgehend 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.dotm* angelegt wird.

Abbildg. 19.1 Übersteuerter Befehl *DateiDruckenStandard* in der Liste der verfügbaren Makros



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 (siehe Abbildung 19.1). Wird der einzelne Befehl mehrmals übersteuert, so gilt die bereits bekannte Reihenfolge.

- Ein Makro in der *Normal.dotm* übersteuert den internen Word-Befehl

- Ein Makro in einem Add-In übersteuert jenes aus der *Normal.dotm*
- 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 *Bsp19_01.dotm* 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.

WICHTIG

Wird ein interner Word-Befehl übersteuert, steht die ursprüngliche Funktionalität nicht mehr zur Verfügung. Wird diese, wie in Listing 19.1, weiterhin benötigt, muss innerhalb des entsprechenden Makros wieder eine Möglichkeit eingebaut werden.

Listing 19.1

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
    Else
        Dialogs(wdDialogFileSaveAs).Show
    End If
End Sub
```

Neben dem Beispiel aus Listing 19.1 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 Aufruf *Datei/Drucken* im Menüband kann der Befehl *Drucken* in der Symbolleiste für den Schnellzugriff gestartet werden. Abhängig vom Grund für die eigentliche Übersteuerung müssten wohl zwei Makros (*DateiDruckenStandard* und *DateiDrucken*) erzeugt werden, damit die gewünschte Funktionalität erreicht wird (siehe Listing 19.2 und Listing 19.3).

Listing 19.2 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 19.3 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 der Arbeitsbereich für das Drucken eingeblendet. Ansonsten wird dieser Aufruf ebenfalls verweigert und der Anwender entsprechend informiert.

In Word 2010 wurde im Menüband der Eintrag *Datei* aufgenommen. Leider kann jetzt der Befehl *Datei/Drucken* nicht mehr übersteuert werden. Um dies zu erreichen, muss mittels XML-Programmierung die Backstage-Ansicht (also der Inhalt der Registerkarte *Datei*) angepasst werden. Mehr dazu ist in Kapitel 16 im Abschnitt »Backstage« beschrieben.

HINWEIS

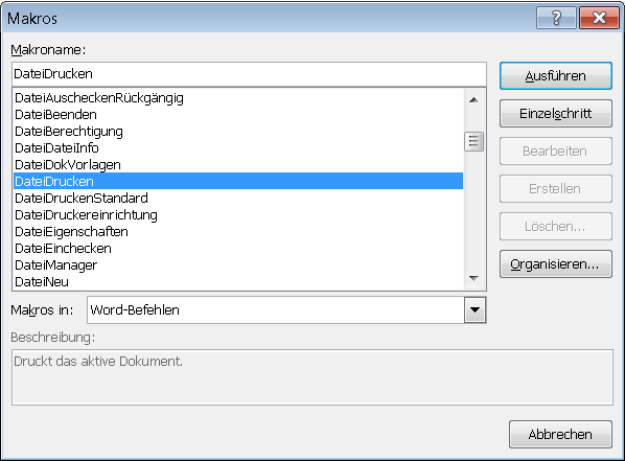
Der programmtechnische Umgang mit internen Dialogfeldern von Word wurde bereits in Kapitel 15 erläutert.

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.

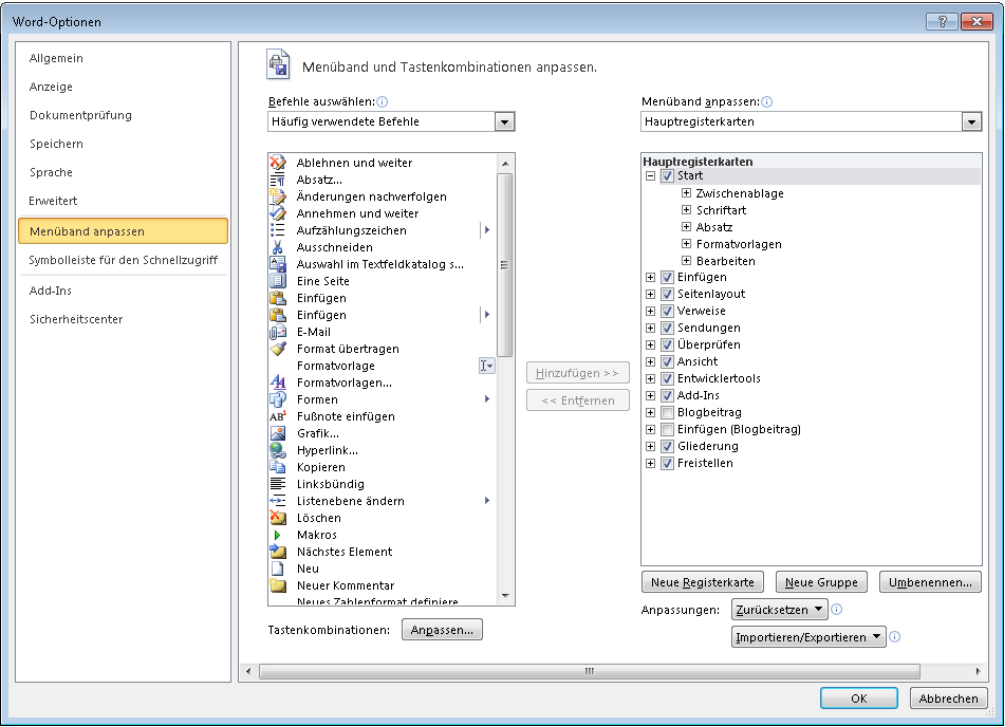
Um diese Liste einsehen zu können, wählen Sie im Menüband auf der Registerkarte *Entwicklertools* die Schaltfläche *Makros*, um das Dialogfeld in Abbildung 19.2 einzublenden.

Abbildg. 19.2 Eine Liste der integrierten Word-Befehle anzeigen lassen



Eine weitere Möglichkeit, den internen Namen eines Befehls zu ermitteln, besteht über den Menüpunkt *Datei/Optionen/Menüband anpassen*.

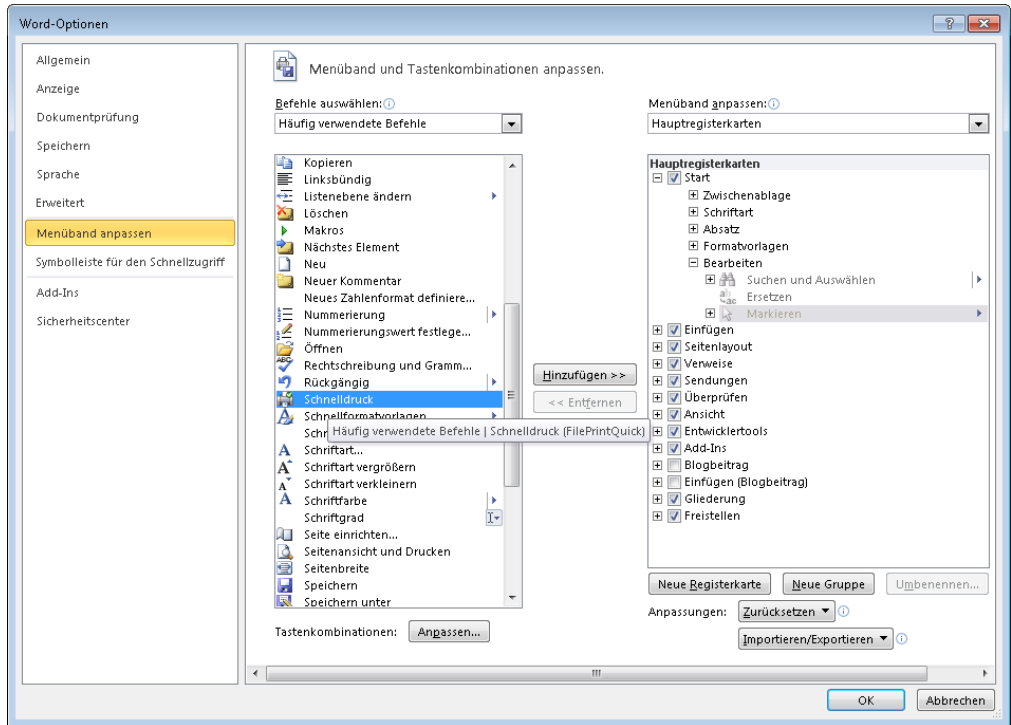
Abbildg. 19.3 Menüband anpassen



Optimierung der
Benutzerschnittstelle

Wenn Sie in der linken Befehlsübersicht mit der Maus einen Eintrag markieren und den Mauszeiger über den Befehl platzieren, wird nach kurzer Zeit der Befehlsname mit der englischen internen Bezeichnung angezeigt.

Abbildg. 19.4 Interne Befehlsnamen ermitteln



ACHTUNG Leider können auch in Word 2010 nicht alle angezeigten Befehle auf diesem Weg abgefangen werden.

So sind die meisten Einträge im Menüband *Datei* nicht abfangbar (mit Ausnahme von *Speichern*, *Speichern unter* und wenige weitere).

Wenn Sie jedoch über den Menüpunkt *Menüband anpassen* einzelne Befehle Ihrer Schnellzugriffsleiste hinzufügen, können deutlich mehr Befehle abgefangen und durch eigenen VBA-Code ersetzt oder erweitert werden.

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 19.2 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 19.1 zusammengefasst.

Tabelle 19.1 Word-Befehle und die zugehörige englische Bezeichnung

Word-Befehl, deutsch	Word-Befehl, englisch
DateiNeu	FileNew
DateiNeuStandard	FileNewDefault
DateiSpeichern	FileSave
DateiSpeichernUnter	FileSaveAs
DateiDrucken	FilePrint
DateiDruckenStandard	FilePrintQuick

CD-ROM Eine komplette Liste der deutschen Word-Befehle und ihrer englischen Benennung finden Sie in der Datei *Interne Word-Befehle.pdf* bzw. *Interne Word-Befehle.xlsx* 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, wird das Makro mit der englischen Bezeichnung abgearbeitet.

CD-ROM Die Prozeduren aus diesem Kapitel finden Sie in der Beispieldatei *Bsp19_01.dotm* auf der CD-ROM zum Buch im Ordner `\Beispiele\Kap19`.

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 812)
- Es wurde darauf hingewiesen, dass die ursprüngliche Funktionalität nicht mehr zur Verfügung steht, sofern diese nicht im eigentlichen Makro zusätzlich eingebaut wird (Seite 812)
- Schließlich wurde vermittelt, wie die Bezeichnung eines Befehls ermittelt und unabhängig von der eingesetzten Programmsprache verwendet werden kann (Seite 814)

Kapitel 20

Zugriff auf den Visual Basic-Editor (VBE)

In diesem Kapitel:

Notwendige Verweise und Sicherheitseinstellungen	820
Der Visual Basic-Editor	821
Auslesen des VBA-Codes von Komponenten	829
Ersetzen und Entfernen von VBA-Codezeilen	837
Hinzufügen von Komponenten zu einem Projekt	840
Entfernen von Komponenten aus einem Projekt	850
Anzeigen von dynamisch erzeugten UserForms	851
Eigenschaften von Controls über das Designer-Objekt dynamisch ändern	854
Verweise auf Bibliotheken und Dateien ermitteln und zur Laufzeit setzen	856
Zusammenfassung	864

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 for Applications-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« ab Seite 821 über den Aufbau des Visual Basic-Editors und die Codedarstellung darin, erhalten Sie im Abschnitt »Auslesen des VBA-Codes von Komponenten« ab Seite 829 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-Codezeilen« ab Seite 837 zeigt Ihnen anschließend, wie Sie Codezeilen ersetzen und auch entfernen können.

Im Abschnitt »Hinzufügen von Komponenten zu einem Projekt« ab Seite 840 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« ab Seite 851 zeigt Ihnen, wie sich die neu erstellen Benutzerformulare und Module aus einem Makro heraus anzeigen und starten lassen.

Im letzten Abschnitt »Ungültige Verweise korrigieren bzw. entfernen« ab Seite 860 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.



In Word 2007 wird die Sicherheit im Vertrauensstellungszentrum der Makrosicherheit eingestellt. Das Vertrauensstellungszentrum rufen Sie entweder über den Eintrag *Makrosicherheit* auf der Registerkarte *Entwicklertools* oder über *Office/Word-Optionen/Vertrauensstellungszentrum/Einstellungen für das Vertrauensstellungszentrum/Einstellungen für Makros*.

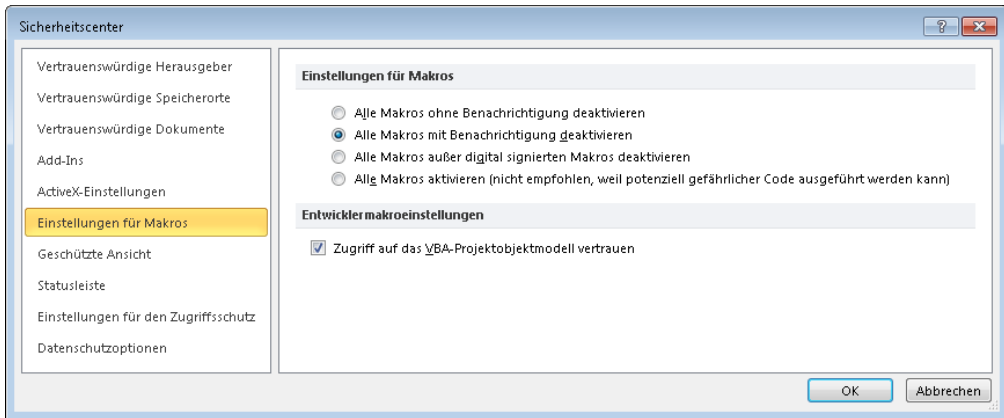
Stellen Sie im Vertrauensstellungszentrum sicher, dass sich die Dokumentvorlage in einem vertrauten Speicherort befindet (Eintrag »Vertrauenswürdige Speicherorte«), und aktivieren Sie das Kontrollkästchen *Zugriff auf das VBA-Projektobjektmodell vertrauen* in der Kategorie *Entwicklertoolmakroeinstellungen*.



In Word 2010 wird die Sicherheit im *Sicherheitscenter* der *Makro-Sicherheit* verwaltet. Das Sicherheitscenter erreichen Sie entweder über den Eintrag *Makrosicherheit* auf der Registerkarte *Entwicklertools* im Menüband (Ribbon) oder über *Datei/Optionen/Sicherheitscenter/Einstellungen für das Sicherheitscenter*.

Stellen Sie im Sicherheitscenter sicher, dass sich die Dokumentvorlage in einem vertrauten Speicherort befindet (Eintrag »Vertrauenswürdige Speicherorte«), und aktivieren Sie das Kontrollkästchen *Zugriff auf das VBA-Projektobjektmodell vertrauen* im Abschnitt *Entwicklertoolmakroeinstellungen*.

Abbildg. 20.1 Sicherheitsstufe in Word 2010 anpassen, um dem Zugriff auf Visual Basic-Projekte vertrauen



Über das Kontrollkästchen *Zugriff auf das VBA-Projektobjektmodell vertrauen* legen Sie fest, ob eigene und fremde Anwendungen oder Makros Zugriff auf die Visual Basic-Projekte erhalten dürfen oder nicht. Markieren Sie dieses Kontrollkästchen, damit die Zugriffe auf den Visual Basic-Editor und die Visual Basic-Projekte nicht mit der Fehlermeldung »Dem programmatischen Zugriff auf das Visual Basic-Projekt wird nicht vertraut« verweigert werden.

Wenn der generelle Zugriff auf das VBA-Projektobjektmodell – also dem 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 (Abbildung 20.2) ist die Umgebung, in der Sie neuen VBA-Code und neue VBA-Prozeduren schreiben bzw. vorhandenen VBA-Code und vorhandene Verfahren bearbeiten.

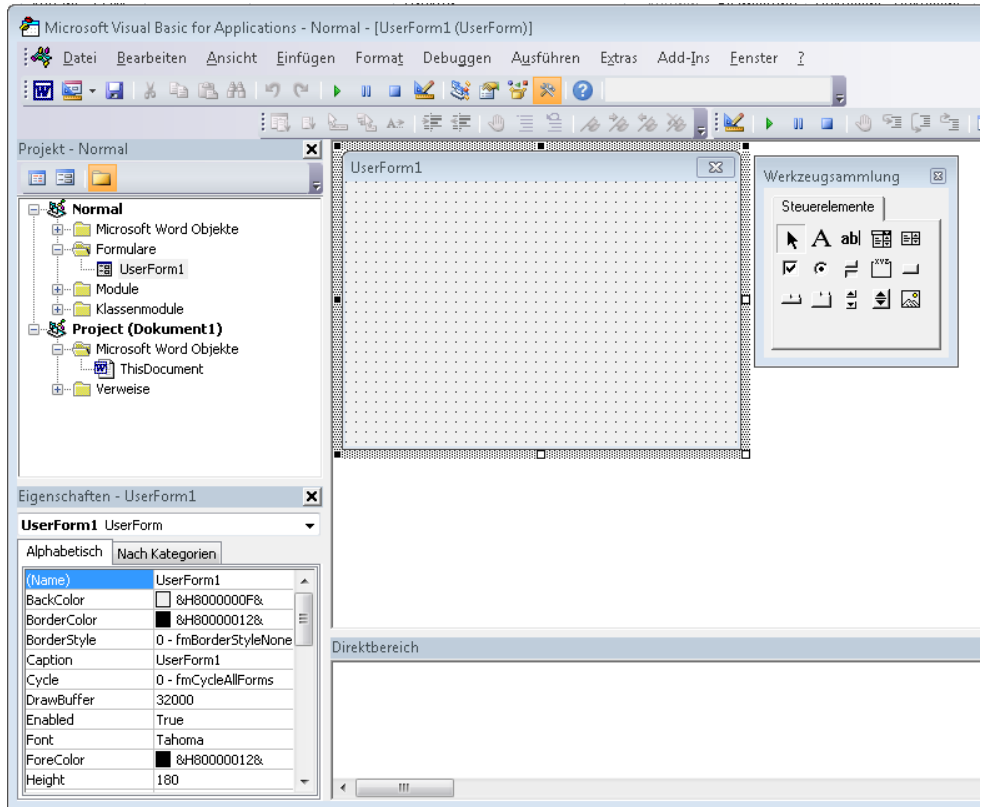
Der Visual Basic-Editor unterteilt sich grob in folgende Bereiche:

- **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. 20.2

Übersicht über den Visual Basic-Editor mit UserForm-Fenster, Projekt-Explorer und Eigenschaftenfenster



HINWEIS Über das Menü *Ansicht* können Sie die verschiedenen Bereiche ein- oder ausblenden: *Code* ([F7]) bzw. *Objekt* (UserForm/Benutzerformular, $\text{Alt} + \text{F7}$), *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-Objekts 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
```

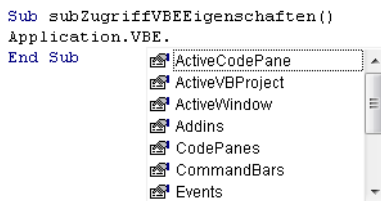
TIPP

Alle Eigenschaften der Hauptebene unter dem Application-Objekt können auch ohne die Angabe des übergeordneten Application-Objekts dargestellt werden.

Wenn Sie aber häufiger VBA-Code in andere Sprachen oder Umgebungen portieren, sollten Sie die vollständige Objekthierarchie verwenden, damit das Application-Objekt korrekt mitportiert wird.

Über die Eigenschaften des VBE-Objekts 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 20.3).

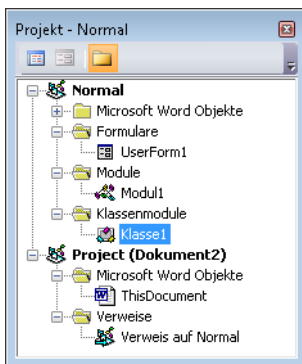
Abbildg. 20.3 Eigenschaft des VBE-Objekts



Die VBA-Projekte

Im Projekt-Explorer des Visual Basic-Editors (Abbildung 20.4) 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. 20.4 Übersicht über den Projekt-Explorer mit allen Projektkomponenten



Sofern diese Projekte nicht als Add-Ins mit Word geladen sind (das können Sie in Word 2010 nach Aufruf des Befehls *Add-Ins* im Menüband *Entwicklertools* bzw. in Word 2007 im Menüband *Entwicklertools* über den Befehl *Dokumentvorlage* im angezeigten 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 kennt zwei Typen von *StartUp*-Ordner:

- Einen im Programmordner der verwendeten Microsoft Office-Version
 2003 `C:\Programme\Microsoft Office\OFFICE11\STARTUP`
 2007 `C:\Programme\Microsoft Office\OFFICE12\STARTUP`
 2010 `C:\Program Files\Microsoft Office\Office14\STARTUP`
- Einen benutzerspezifischen für jeden Anwender unter Windows
 Windows XP `C:\Dokumente und Einstellungen\<Benutzername>\Anwendungsdaten\Microsoft\Word\StartUp`
 Windows Vista/7 `C:\Users\<Benutzername>\AppData\Roaming\Microsoft\Word\STARTUP`
 Dieser Pfad wird im Explorer von Windows 7 auf Deutsch wie in Abbildung 20.5 angezeigt, sofern Microsoft Office im vorgeschlagenen Standardverzeichnis auf dem Laufwerk C: installiert wurde.

Abbildg. 20.5

Pfadanzeige unter Windows 7

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.

CD-ROM

Die in diesem Abschnitt verwendeten Beispielprozeduren finden Sie auf der CD-ROM in der Datei `\Beispiele\Kap20\Bsp20_01.docm` im Modul `modProjektNamen`.

Das Beispiel in Listing 20.1 zeigt für jedes Projekt neben dem Namen auch die Zugriffsmöglichkeit an:

Listing 20.1

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 Projekts 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 Projektnamen 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 inklusive Pfad des aktuellen Projekts und somit auch den Typ (Dokument oder Dokumentvorlage, erkennbar an der Dateiendung), wie in Listing 20.2 veranschaulicht.

Listing 20.2 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
```

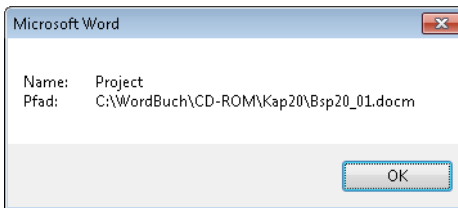
Das aktive VBA-Projekt

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 und den Pfad des aktuellen Projekts erhalten Sie, wie in Abbildung 20.6 ersichtlich, wieder über die Name-Eigenschaft und FileName-Eigenschaft des ActiveVBProject-Objekts:

```
Sub subAktiveVBProjekt()
    MsgBox "Name: " & vbTab & VBE.ActiveVBProject.Name & vbCrLf & _
        "Pfad: " & vbTab & VBE.ActiveVBProject.FileName
End Sub
```

Abbildg. 20.6 Ermittlung des aktuellen Projekt-Namens und Pfads

Abbildg. 20.7 Anzeige der Projekt-Informationen

TIPP

Eine weitere Möglichkeit, den Pfad und Dateinamen des *aktuellen* Projekts zu erhalten, besteht über die MacroContainer-Eigenschaft des Application-Objekts.

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.

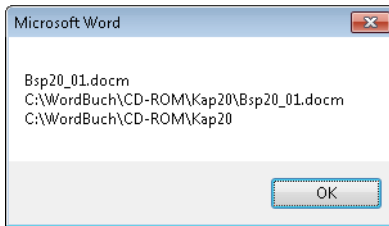
Leider werden für diese Eigenschaft die zur Verfügung stehenden Methoden nicht angezeigt, wenn man nach dem Namen einen Punkt eingibt. Die wichtigsten Methoden der MacroContainer-Eigenschaft sind:

- **Name** Gibt den Namen des Dokuments oder der Dokumentvorlage, in der die aufrufende Prozedur enthalten ist, zurück
- **FullName** Gibt den vollständigen Namen einschließlich dem Pfad des Dokuments oder der Dokumentvorlage, in der die aufrufende Prozedur enthalten ist, zurück
- **Path** Gibt den Pfad des Dokuments 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 20.8):

```
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. 20.8 Ermittlung der Projektinformationen über die *MacroContainer*-Eigenschaft



Zugriff auf die in einem Projekt enthaltenen Komponenten

Jedes VBProject-Objekt umfasst eine Reihe von Komponenten: die VBComponents-Objekte.

Zu diesen Komponenten gehören alle im Projekt-Explorer unterhalb des Projekts angezeigten Elemente, wie in Abbildung 20.4 ersichtlich:

- **Microsoft Word Objekte** Das mit dem Projekt verbundene Dokument. In Microsoft Word wird dieses Objekt immer durch *ThisDocument* dargestellt und repräsentiert das aktuelle Dokument oder Dokumentvorlage.
- **Formulare (UserForms)** Alle im Projekt enthaltenen Benutzerformulare (UserForms)
- **Module (Sammlung von Prozeduren)** Alle im Projekt enthaltenen Module mit den Prozeduren (Makros). Ein Modul kann dabei mehrere Prozeduren (Sub, Function oder Property) enthalten.
- **Klassenmodule** Alle im Projekt enthaltenen Klassenmodule
- **Verweise** 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 Projekts 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 20.1.

Tabelle 20.1 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

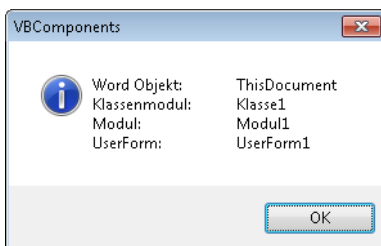
Listing 20.3 Auflistung aller in einem Projekt enthaltenen *VBComponents* mit Typangabe

```

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: "
            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 20.3 werden alle im aktiven *VBProject* enthaltenen Komponenten mit Typ-Aufschlüsselung und ihrem Namen angezeigt, wie in Abbildung 20.9 ersichtlich.

Abbildg. 20.9 Ausgabe aller *VBComponents*-Einträge eines Projekts


Der VBA-Code einer jeden Komponente wird im Codefenster angezeigt, dort eingegeben oder geändert. Dieses Codefenster wird durch das `CodeModule`-Objekt dargestellt, das den gesamten VBA-Code der Komponente umfasst.

Über die Eigenschaften des `CodeModule`-Objekts können Sie VBA-Code hinzufügen, ändern, bearbeiten und löschen.

Die Tabelle 20.2 listet die Methoden auf, die zum Erstellen, Bearbeiten und Löschen von VBA-Code im Codemodul 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.

CD-ROM Die in diesem Abschnitt verwendeten Beispielprozeduren finden Sie auf der CD-ROM zum Buch in der Datei `\Beispiele\Kap20\Bsp20_01.docm` im Modul `modActiveVBProject`.

Auslesen des VBA-Codes von Komponenten

Über die Methoden in Tabelle 20.2 erhalten Sie Zugriff auf die Zeilen eines Codemoduls.

Tabelle 20.2 Übersicht über die Methoden zum Zugriff auf ein Codemodul

Zugriffsmethode	Beschreibung
<code>CountOfDeclarationLines</code>	Gibt die Anzahl der Codezeilen im Deklarationsabschnitt eines Codemoduls zurück (siehe den Abschnitt »Zugriff auf den Deklarationsbereich« ab Seite 831)
<code>CountOfLines</code>	Gibt die Gesamtzeilenzahl des Codemoduls an (siehe den Abschnitt »Zugriff auf die Codezeilen einzelner Prozeduren« ab Seite 832)
<code>Find</code>	Gibt an, ob in einem Codemodul ein angegebener Text enthalten ist (siehe den Abschnitt »Auflisten und Durchsuchen aller Projekte« ab Seite 835)
<code>Lines</code>	Gibt eine angegebene Anzahl von Zeilen aus dem Codemodul zurück (siehe den Abschnitt »Zugriff auf die Codezeilen einzelner Prozeduren« ab Seite 832)
<code>ProcBodyLine</code>	Gibt die erste Zeile einer Prozedur zurück, in der die Sub -, Function - oder Property -Anweisung enthalten ist (siehe den Abschnitt »Zugriff auf die Codezeilen einzelner Prozeduren« ab Seite 832)
<code>ProcCountLines</code>	Gibt die Anzahl der Zeilen in der festgelegten Prozedur zurück (siehe den Abschnitt »Zugriff auf die Codezeilen einzelner Prozeduren« ab Seite 832)
<code>ProcOfLine</code>	Gibt den Namen der Prozedur zurück, in der sich die festgelegte Zeile befindet (siehe den Abschnitt »Zugriff auf die Codezeilen einzelner Prozeduren« ab Seite 832)

Tabelle 20.2 Übersicht über die Methoden zum Zugriff auf ein Codemodul (*Fortsetzung*)

Zugriffsmethode	Beschreibung
ProcStartLine	Gibt die Zeile zurück, an der die festgelegte Prozedur beginnt (siehe den Abschnitt »Zugriff auf die Codezeilen einzelner Prozeduren« ab Seite 832)

Diese Methoden lassen sich wieder grob in zwei Gruppen unterteilen:

- Methoden zum allgemeinen Zugriff auf die Codezeilen
- Methoden zum Zugriff auf die Prozeduren

CD-ROM Die in diesem Abschnitt verwendeten Beispielprozeduren finden Sie auf der CD-ROM zum Buch in der Datei `\Beispiele\Kap20\Bsp20_01.docm` im Modul *modReadVBComponents*.

Allgemeine Zugriffe auf die Codezeilen

Über die beiden Eigenschaften `CountOfLines` und `Lines` erhalten Sie Zugriff auf alle Codezeilen im Codemodul. 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 Codemodul zurückliefert.

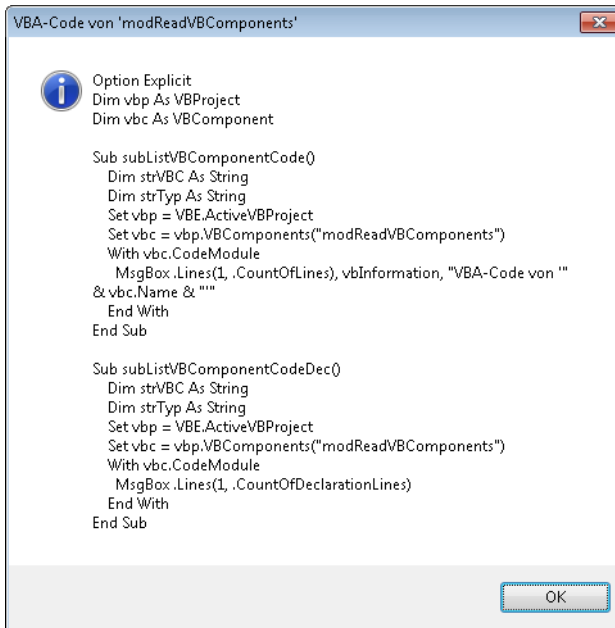
Listing 20.4 Ausgabe des gesamten VBA-Code eines Codemoduls

```
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), vbInformation, "VBA-Code von '" & vbc.Name & "'"
    End With
End Sub
```

Als Ergebnis erhalten Sie die Übersicht über den VBA-Code in Abbildung 20.10.

WICHTIG Da die `MsgBox`-Funktion nur Zeichenfolgenlängen von 1.024 Zeichen ausgeben kann, wird in den Fällen, wo diese Grenze überschritten wird, die Ausgabe abgebrochen. Bei umfangreicheren Codemodulen kann es daher passieren, dass nicht alle Codezeilen angezeigt werden.

Abbildg. 20.10 Anzeige des VBA-Codes eines Codemoduls



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 20.5 veranschaulicht.

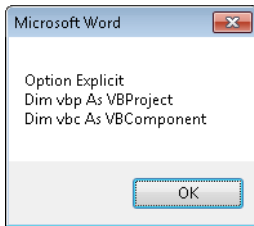
Listing 20.5 Anzeigen des Deklarationsbereiches

```

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 20.10 in Abbildung 20.11 angezeigt.

Abbildg. 20.11 Anzeige der Zeilen des Deklarationsbereichs eines Codemoduls


Zugriff auf die Codezeilen einzelner Prozeduren

Innerhalb einer Komponente befinden sich neben allgemeinen Deklarationen häufig auch mehrere Prozeduren: z.B. in Abbildung 20.10 die beiden Prozeduren *subListVBComponentCode* und *subListVBComponentCodeDec*.

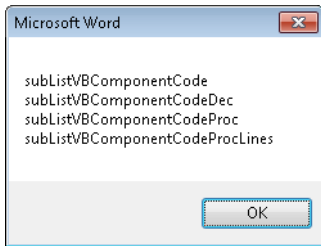
Die Prozeduren selbst lassen sich nur dann ansprechen, wenn die Prozedurnamen bekannt sind. Leider steht keine Methode oder Eigenschaft zur Verfügung, die diese Informationen direkt liefert. Sie können jedoch die Eigenschaft *ProcOfLine* verwenden, die den Prozedurnamen der aktuellen Zeile zurückliefert. Somit kann man mit einer Schleife über alle Zeilen eines Codemoduls 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 20.5 durchläuft zeilenweise den Code der Komponente *modReadVBComponents* und liefert nur die Prozedurnamen zurück. Das Ergebnis ist in Abbildung 20.12 ersichtlich.

Listing 20.6 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. 20.12 Ermittlung und Ausgabe der Prozedurnamen in einer Komponente



Mit diesen Informationen können Sie nun zusammen mit den Eigenschaften ProcBodyLine und ProcCountLines den Codebereich einer Prozedur ermitteln.

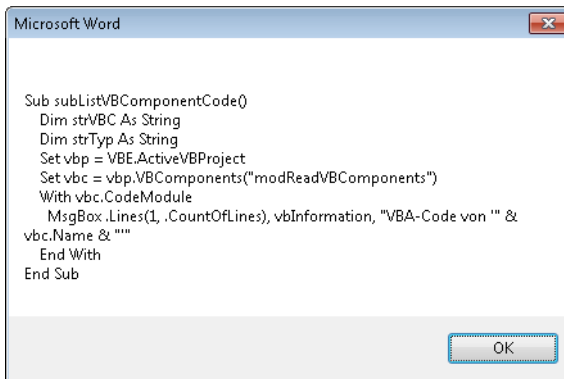
ACHTUNG Die ProcBodyLine-Eigenschaft liefert die erste Zeile einer Prozedur zurück, in der die Sub-, Function- oder Property-Anweisung enthalten ist. Allerdings können vor der Anweisung auch noch Leerzeilen und Kommentarzeilen stehen, die ebenfalls zu dieser Prozedur gehören.

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 20.7; das Ergebnis befindet sich in Abbildung 20.13.

Listing 20.7 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. 20.13 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« ab Seite 827 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 20.8 überprüft zeilenweise auf die Prozedurtypen und liest dann über die passende Konstante die Prozedur aus.

Listing 20.8 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 20.9 zeigen, wie Sie alle ungeschützten Projekte nach einer Prozedur durchsuchen können. Wird diese gefunden, wird die gesamte Prozedur angezeigt.

CD-ROM Die Beispielprozedur zum Durchsuchen aller Projekte finden Sie auf der CD-ROM zum Buch in der Datei *\Beispiele\Kap20\Bsp20_01.docm* im Modul *modSearchAndReplaceLines*.

Listing 20.9 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)
          End If
        Next intLine
      End With
    Next vbc
  Next vbp
End Sub
```

Listing 20.9 Durchsuchen aller Projekte nach einer bestimmten Prozedur (Fortsetzung)

```

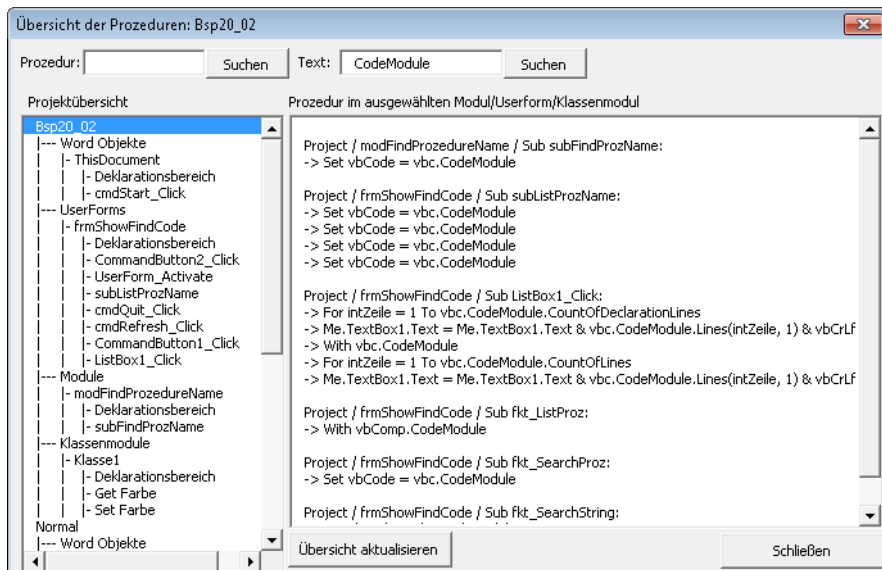
        MsgBox strMsg, vbInformation, vbc.Name
    Exit For
End If
Next intLine
End With
Next vbc
p_next:
Next vbp
Set vbCode = Nothing
End Sub

```

CD-ROM

In der Beispieldatei *Bsp20_02.docm*, auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap20*, finden Sie ein zusätzliches Beispiel, wie Sie den gesamten VBA-Code in allen ungeschützten Projekten in einer Art Baumstruktur anzeigen lassen können, und wie Sie den gesamten VBA-Code nach einer Prozedur und nach einem beliebigen String durchsuchen können.

Nach dem Öffnen dieser Datei rufen Sie die UserForm zur Prozedur- und Textsuche über die angezeigte Schaltfläche *VBA-Code/-Prozedur suchen* auf. Es öffnet sich daraufhin das in Abbildung 20.14 gezeigte Benutzerformular.

Abbildg. 20.14 Benutzerformular zum Durchsuchen aller Projekte nach einem Prozedurnamen oder einem beliebigen Text


Ersetzen und Entfernen von VBA-Codezeilen

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 20.2 aufgelisteten Zugriffsmethoden sind aber auch Methoden zum Ersetzen und Entfernen von VBA-Codezeilen enthalten.

Tabelle 20.3 Methoden zum Suchen und Entfernen von Codezeilen

Zugriffsmethode	Beschreibung
DeleteLines	Löscht eine oder mehrere Zeilen
ReplaceLine	Ersetzt eine vorhandene Codezeile durch eine angegebene Codezeile

Die ReplaceLine-Methode erwartet als Parameter neben der genauen Zeile im CodeModule-Objekt der Komponente den einzufügenden Text. Beachten Sie dabei, dass einzufügende Anführungszeichen mithilfe der Chr(34)-Funktion maskiert werden müssen.

Das Makro in Listing 20.10 sucht in allen Projekten nach der Prozedur *subFindProzName* (siehe Listing 20.9) 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 20.15 ersichtlich; beachten Sie die beiden markierten Bereiche, wo die Änderungen vorgenommen wurden.

CD-ROM Die Beispielprozeduren finden Sie auf der CD-ROM zum Buch in der Datei `\Beispiele\Kap20\Bsp20_01.docm` im Modul *modSearchAndReplaceLines*.

Listing 20.10 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
        Next intLine
        If bFound = True Then
            For intLine = 1 To .CountOfLines
```

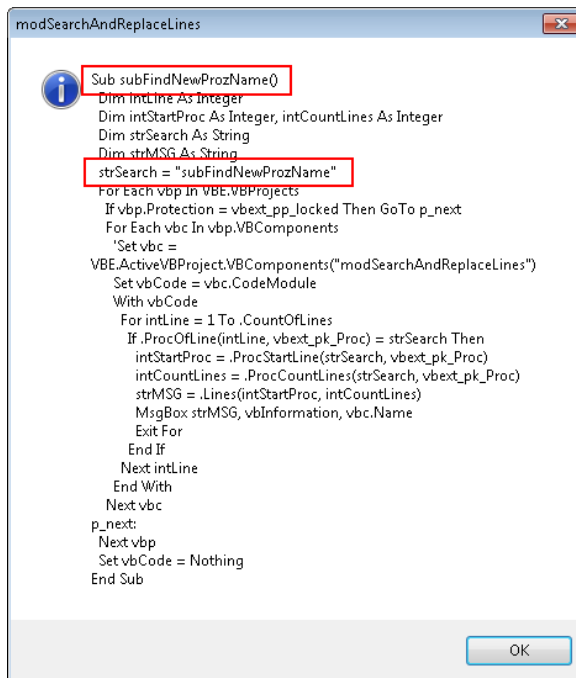
Listing 20.10 Umbenennen und Ersetzen einer Zeile in einer Prozedur und Ausführen der geänderten Prozedur (Fortsetzung)

```

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. 20.15 Ergebnis nach der Ausführung von Listing 20.10



WICHTIG Wenn sich die aufzurufende geänderte Prozedur im *selben* Codemodul 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 Codemodul 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 Codemodul 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`-Objekts. Diese Methode startet zu einem bestimmten Zeitpunkt das angegebene Makro:

```
Application.OnTime Now, "Prozedurname", 1
```

Als Zeitpunkt können Sie auf die `Now`-, `TimeValue`- oder `TimeSerial`-Funktion verwenden und kombinieren. Als dritten Parameter können Sie eine Toleranzzeit angeben, zu dem das angegebene Makro abgebrochen wird, wenn es zum angegebenen Zeitpunkt nicht ausgeführt werden konnte (z.B. wenn noch ein Dialogfeld geöffnet ist).

Um beispielsweise eine Prozedur in fünf Sekunden zu starten, können Sie den folgenden Aufruf verwenden:

```
Application.OnTime Now + TimeSerial(0, 0, 5), "Prozedurname", 1
```

Mit der `DeleteLines`-Methode können Sie eine oder mehrere Zeilen löschen.

Als Parameter erwartet diese Methode die absolute Zeilenzahl im Codemodul und die Anzahl der Zeilen. Das Makro in Listing 20.11 löscht die ersten beiden Zeilen im Codemodul, 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.

Listing 20.11 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
```

Hinzufügen von Komponenten zu einem Projekt

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« ab Seite 827):

- Sie importieren eine als Datei (siehe Kapitel 1) vorliegende Komponente (Benutzerformular, Modul oder Klassenmodul) mit dem VBA-Code, oder
- Sie erstellen eine neue Komponente per VBA-Code und füllen diese per VBA mit Codezeilen

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 20.4 stehen Ihnen zum Erstellen und Bearbeiten von Komponenten, Ereignissen und Codezeilen zur Verfügung.

Tabelle 20.4 Übersicht über die Methoden zum Erstellen und Bearbeiten von Komponenten per VBA-Code

Zugriffsmethode	Beschreibung
AddFromFile	Fügt den Inhalt einer angegebenen Datei in ein Codemodul ein (siehe den Abschnitt »Makro-Anweisungen per VBA-Code importieren« ab Seite 843)
AddFromString	Fügt einen angegebenen Text in das Codemodul in die erste Zeile ein (siehe den Abschnitt »Makro-Anweisungen per VBA-Code einfügen« ab Seite 844)
CreateEventProc	Erstellt für ein angegebenes Objekt eine Ereignisprozedur (siehe den Abschnitt »Makro-Anweisungen per VBA-Code einfügen« ab Seite 844)
DeleteLines	Löscht eine oder mehrere Zeilen (siehe den Abschnitt »Ersetzen und Entfernen von VBA-Codezeilen« ab Seite 837)
InsertLines	Fügt eine oder mehrere Zeilen an einer bestimmten Stelle in das Codemodul ein (siehe den Abschnitt »Makro-Anweisungen per VBA-Code einfügen« ab Seite 844)
ReplaceLine	Ersetzt eine vorhandene Codezeile durch eine angegebene Codezeile (siehe den Abschnitt »Ersetzen und Entfernen von VBA-Codezeilen« ab Seite 837).

Eine vorhandene Komponente in ein Projekt importieren

Für den Import einer Komponente zu einem Projekt steht Ihnen die Import-Methode des VBProject-Objekts zur Verfügung.

Über die Import-Methode können Sie dem aktuellen Projekt einen der drei genannten Komponententypen (UserForm, Standardmodul, Klassenmodul) hinzufügen. Als Parameter müssen Sie dazu den vollständigen Pfad zu der zu importierenden Datei angegeben werden.

TIPP

Diese Methode erwartet standardmäßig den vollständigen Pfad zur Datei. Wenn die Komponente jedoch in einem Unterverzeichnis des aktuellen Projekts 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« ab Seite 825):

```
VBE.ActiveVbProject.VbComponents.Import MacroContainer.Path & _
"\Beispiele\modImportModul.bas"
```

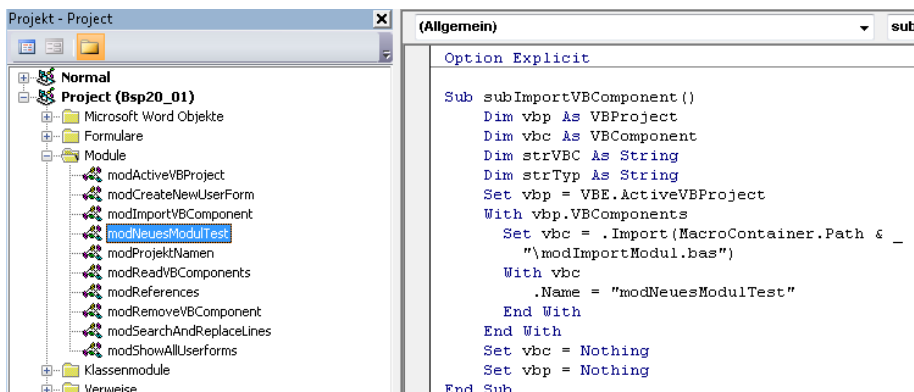
Diese MacroContainer-Eigenschaft bietet sich immer dann an, wenn der absolute Pfad zur Projektdatei (Dokument oder Dokumentvorlage), in der die aktuelle Prozedur enthalten ist, benötigt wird. Auch zur Ermittlung des aktuellen Pfades zur Projektdatei bietet sich diese Eigenschaft an.

Als Komponenten-Name wird der normalerweise in der importierten Datei als Attribut vermerkte Name verwendet. Sie können diesen aber nach dem Import über die Name-Eigenschaft ändern, wie Listing 20.12 veranschaulicht. Das Ergebnis ist in Abbildung 20.16 ersichtlich.

Listing 20.12 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
```

Abbildg. 20.16 Anzeige des importierten Moduls im Projekt-Explorer



CD-ROM Sie finden die beiden Dateien *Bsp20_01.docm* und *modImportModul.bas* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap20*.

WICHTIG Existiert bereits eine Komponente mit dem intern vermerkten Komponentennamen, wird beim Importieren der Komponente der Name mit einer fortlaufenden Nummer erweitert. Wenn Sie nach dem Import als Namen für die Komponente einen bereits vergebenen Namen zuweisen möchten, erhalten Sie eine Fehlermeldung. Dies ist beispielsweise der Fall, wenn Sie die Prozedur *subImportVBComponent* aus Listing 20.12 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« ab Seite 827 wird beschrieben, wie Sie die Namen aller Komponenten ermitteln können.

Eine neue Komponente einem Projekt hinzufügen

Um dem aktiven Projekt eine neue Komponente hinzuzufügen, verwenden Sie die *Add*-Methode. Diese Methode erwartet als Parameter einen der in Tabelle 20.1 *vbext*-Konstantenwerte.

Weitere Informationen zu den verschiedenen *VBComponents*-Typen finden Sie im Abschnitt »Zugriff auf die in einem Projekt enthaltenen Komponenten« ab Seite 827.

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 20.13 erstellt ein neues leeres Standardmodul mit dem Namen *modNeuesModul*.

Listing 20.13 Hinzufügen eines neuen Standardmoduls zum aktiven Projekt

```
Sub subAddVBComponent()  
    Dim vbp As VBProject  
    Dim vbc As VBComponent  
    Dim strVBC As String  
    Dim strTyp As String  
    Set vbp = VBE.ActiveVBProject  
    With vbp.VBComponents  
        Set vbc = .Add(vbext_ct_StdModule)  
        With vbc  
            .Name = "modNeuesModul"  
        End With  
    End With  
    Set vbc = Nothing  
    Set vbp = Nothing  
End Sub
```

Nach Ausführen der Prozedur *subAddVBComponent* wird die neue Komponente im Projekt-Explorer unter den Komponenten *Module* einsortiert angezeigt.

Dieses Modul ist jedoch noch leer und muss mit Codezeilen 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. Wenn Sie das Makro (Listing 20.13) ein zweites Mal ausführen, erhalten Sie eine Fehlermeldung, da das Modul bereits vorhanden ist.

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 Codebereich der jeweiligen Komponente angezeigt wird.

Zum Importieren von Makro-Anweisungen in eine Komponente steht Ihnen die `AddFromFile`-Methode des `CodeModule`-Objekts 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 20.14 zeigt ein Beispiel hierfür, wenn sich die einzufügende Datei im selben Ordner wie die Beispieldatei *Bsp20_01.docm* befindet.

Listing 20.14 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 & _
            "\modSearchProc.bas"
        .Name = "modNeuesModulTest"
    End With
    Set vbc = Nothing
    Set vbp = Nothing
End Sub
```

CD-ROM

Sie finden die beiden Dateien *Bsp20_01.docm* und *modSearchProz.bas* auf der CD-ROM zum Buch im Ordner `\Beispiele\Kap20`.

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 20.15 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 20.15 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 20.15 dahingehend um, indem Sie die `InsertLines`-Methode verwenden:

Listing 20.16 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

```

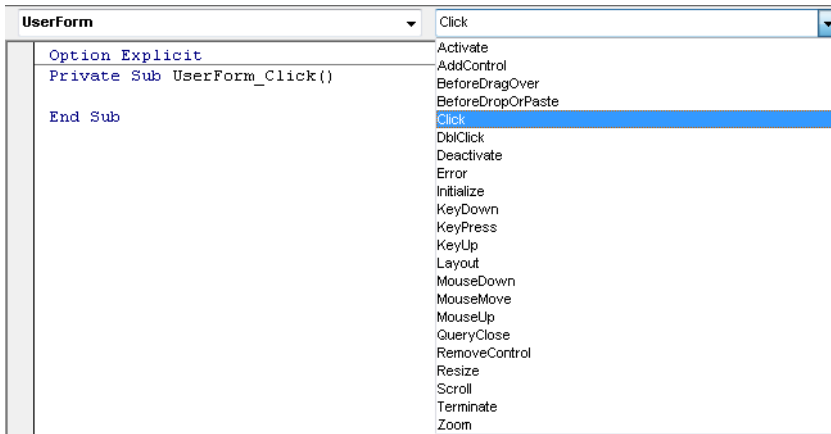
Da Sie alle Codezeilen als in Anführungszeichen eingeschlossenen Text angeben müssen, müssen alle syntaktisch notwendigen Anführungszeichen durch ihren Zeichencode *Chr(34)* maskiert werden.

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 denen dieses Ereignis beginnt.

Um aber ein Objektereignis 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« ab Seite 851 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 20.17), wenn Sie sich in der Codeanzeige des UserForms befinden.

Abbildg. 20.17 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 20.17 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 20.17 Erstellen des *QueryClose*-Ereignisses für ein Benutzerformular

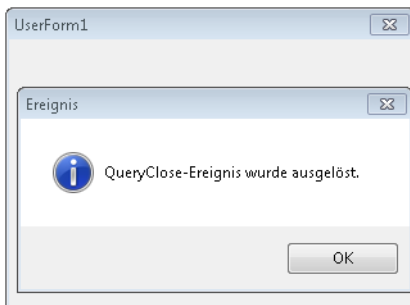
```
Sub subAddEventToUserForm()
    Dim strQ As String
    strQ = Chr(34)
    Dim vbp As VBProject
    Dim vbc As VBComponent
    Dim intZeile As Integer
    Set vbp = VBE.ActiveVBProject
    Set vbc = vbp.VBComponents.Add(vbext_ct_MSForm)
    vbc.Name = "frmTestModule"
    With vbc.CodeModule
        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 20.18), die dann ausgegeben wird.

Listing 20.18 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 20.18 ausgegeben.

Abbildg. 20.18 Ausgabe des ausgelösten *QueryClose*-Ereignisses

Steuerelemente auf einem UserForm erzeugen

Um ein Objektereignis anlegen zu können, benötigen Sie, mit Ausnahme der UserForm selbst, ein entsprechendes Objekt (Control) in Form eines Steuerelements auf der UserForm.

Während die Ereignisprozeduren im Codemodul der jeweiligen Komponente angelegt werden, werden die Objekte im Designer-Bereich des UserForms erstellt.

HINWEIS Die grafische Benutzeroberfläche *Designer*

Der Designer ist eine Eigenschaft des VBComponent-Objekts 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.

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.

CD-ROM Die in diesem Abschnitt verwendeten Beispielprozeduren finden Sie auf der CD-ROM zum Buch in der Datei `\Beispiele\Kap20\Bsp20_01.docm` im Modul `modCreateNewUserForm`.

WICHTIG Damit Ihnen alle Eigenschaften und Methoden während der Entwicklung zur Verfügung stehen, müssen Sie den Objektverweis 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 Objektverweis auf das jeweilige Steuerelement zurück.

Um ein Steuerelement dem Benutzerformular hinzuzufügen, verwenden Sie die Add-Methode des angegebenen Control-Objekts. 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 20.5 aufgelistet.

Tabelle 20.5 Übersicht über die Ressource-IDs der Standard-Steuerelemente

Bezeichnung	Ressource-ID (ProgID)	Symbol
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-Objekts können Sie alle Eigenschaften des Steuerelements konfigurieren, wie Sie es auch im Visual Basic-Editor über das Eigenschaftenfenster des jeweiligen Steuerelements vornehmen können.

Das Makro in Listing 20.19 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.

ACHTUNG Die .Name-Eigenschaft einer Komponente (Modul oder UserForm) im aktiven Projekt kann entgegen der VBA-Hilfe zur Laufzeit nicht geändert werden.

Listing 20.19 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 VBCComponent
    Dim strVBC As String
    Dim strTyp As String
    Dim intZeile As Integer
    Set vbp = VBE.ActiveVBProject
    Set vbc = vbp.VBCComponents.Add(vbext_ct_MSForm)
    With vbc
        .Properties("Width") = 300
        .Properties("Height") = 200
        .Properties("Caption") = "Per Code erstellte UserForm"
    ' On Error Resume Next
    ' Die .Name-Eigenschaft kann entgegen der VBA-Hilfe zur Laufzeit nicht geändert werden
    ' .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
```

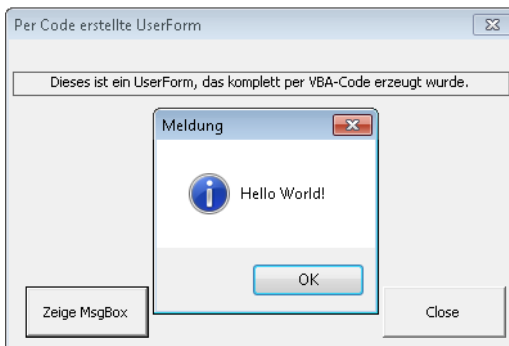
Listing 20.19 Dynamisch erstelltes Benutzerformular mit Steuerelementen (Fortsetzung)

```

.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 20.19 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. 20.19 Anzeige des dynamisch erstellten Benutzerformulars

CD-ROM

Die in diesem Abschnitt verwendeten Beispielprozeduren finden Sie auf der CD-ROM zum Buch in der Datei `\Beispiele\Kap20\Bsp20_01.docm` im Modul `modImportVBComponent`.

Entfernen von Komponenten aus einem Projekt

Zum Entfernen einzelner Komponenten aus einem nicht gesperrten Projekt steht Ihnen die `Remove`-Methode des `VBComponents`-Objekts 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 20.20 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.

CD-ROM Die verwendete Beispielprozedur finden Sie auf der CD-ROM zum Buch in der Datei `\Beispiele\Kap20\Bsp20_01.docm` im Modul `modRemoveVBComponent`.

Listing 20.20 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
    Set vbp = Nothing
End Sub
```

WICHTIG Die Remove-Methode akzeptiert nur ein VBComponent-Objekt der VBComponents-Auflistung. Wenn Sie versuchen, das VBComponents-Objekt über den Namen anzugeben, erhalten Sie die Fehlermeldung »Typen unverträglich«.

Anzeigen von dynamisch erzeugten UserForms

Mit den bisherigen Eigenschaften und Methoden können Sie jetzt zwar ein UserForm erstellen, mit Steuerelementen gestalten und Ereignisse erstellen, es steht Ihnen aber kein direkter Befehl zum Anzeigen des UserForms zur Verfügung.

Bei UserForms, die Sie direkt im Visual Basic-Editor erstellen, können Sie diese über ihren Namen und der Show-Eigenschaft aufrufen. Dazu wird über den Namen auf das entsprechende UserForm-Objekt referenziert.

PROFITIPP

Im Gegensatz zu den im Visual Basic-Editor erstellten UserForms stehen Ihnen bei dynamisch erstellten UserForms diese nicht als Referenz zur Verfügung, da das entsprechende UserForm-Objekt vor dem Ausführen des Makros noch nicht bekannt ist und somit vom Visual Basic-Editor auch nicht referenziert wurde. Da Sie auch (mit Einschränkungen) den Namen des UserForms zur Laufzeit ändern können, wodurch sich auch das UserForm-Objekt selbst ändern würde, müssen Sie für dynamisch erstellte UserForms die UserForms-Auflistung verwenden.

Über diese UserForms-Auflistung erhalten Sie Zugriff auf alle *geladenen* UserForms. Zugriff auf ein einzelnes UserForm erhalten Sie durch das UserForm-Objekt, das Ihnen die UserForms-Auflistung zurückliefert.

Um nun einen Zugriff auf eine bestimmte *UserForm*-Komponente in einem Projekt zu erhalten, müssen Sie das UserForm erst laden und somit der UserForms-Auflistung hinzufügen. Hierzu steht Ihnen die *Add*-Methode zur Verfügung, der Sie entweder den Index oder den Namen des UserForms, wie er im Projekt-Explorer angezeigt wird, als Parameter mitgeben müssen.

Anschließend stehen Ihnen alle Eigenschaften eines UserForm-Objekts 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 UserForms *nicht* zur Verfügung.

Damit Ihnen während der Entwicklung trotzdem die Eigenschaften und Methoden des UserForm-Objekts 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 bestimmt 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 20.21 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*-Objekts über die Zeile

```
Set objFrm = Nothing
```

wird das UserForm wieder aus der UserForms-Auflistung entfernt.

CD-ROM Die verwendeten Beispielprozeduren finden Sie auf der CD-ROM zum Buch in der Datei *\Beispiele\Kap20\Bsp20_01.docm* im Modul *modShowAllUserforms*.

Listing 20.21 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)
```


Listing 20.21 Anzeige aller Benutzerformulare im aktuellen Projekt (Fortsetzung)

```

        objFrm.Caption = vbc.Name
        objFrm.Show
        Set objFrm = Nothing
    End If
Next vbc
End Sub

```

Mit Listing 20.21 erhalten Sie Zugriff auf das UserForm mit normalerweise einer Reihe von Steuerelementen. In Listing 20.19 haben Sie gesehen, dass neue Steuerelemente (Controls-Objekte) über die Designer-Eigenschaft des VBComponents-Objekts 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 Objekts besitzt. So ist eine Eigenschaft die Controls-Eigenschaft mit der bekannten Name-Methode. Zur Ermittlung des Steuerelementtyps 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-Objekts einzustellen bzw. zu erhalten oder eine Methode dieses Objekts aufzurufen. So liefert folgende Zeile die Beschriftung des CommandButton-Steuerelementes *CommandButton1*:

```
MsgBox CallByName(CommandButton1,"Caption",VbGet)
```

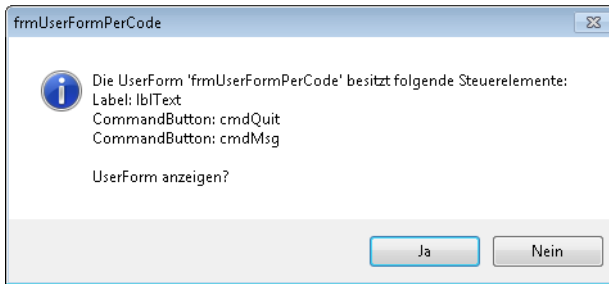
Die Prozedur *subShowUserFormControlInfos* in Listing 20.22 benutzt diese Funktion, um für alle UserForms die enthaltenen Steuerelemente mit Typ und Namen anzuzeigen (Abbildung 20.20), bevor das jeweilige UserForm angezeigt werden kann.

Listing 20.22 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. 20.20 Auflistung der Steuerelemente eines Benutzerformulars und Anzeige des *UserForm*-Objekts


Wenn Sie auf die Schaltfläche *Ja* klicken, wird das UserForm-Objekt geladen und angezeigt (siehe Abbildung 20.18).

Eigenschaften von Steuerelementen über das Designer-Objekt dynamisch ändern

Über das Designer-Objekt erhalten Sie, wie in »Die grafische Benutzeroberfläche *Designer*« ab Seite 847 angesprochen, Zugriff auf die Elemente auf einem UserForm. So können Sie über die Controls-Eigenschaft direkt ein Steuerelement über seinen Namen oder Index ansprechen. Kennen Sie die Eigenschaften des Steuerelements, wie sie im Eigenschaftenfenster angezeigt und eingestellt werden können, lassen sich darüber die Eigenschaften direkt im Designer-Objekt ändern. Im Listing 20.23 werden auf das Steuerelement *txtAnzeige* auf dem UserForm *frmChangeUFValues* zugegriffen und einige Eigenschaften (*Value*, *BackStyle*, *TextAlign*, *SpecialEffect*) gesetzt.

Listing 20.23 Setzen der Eigenschaften eines UserForm-Controls im Designer

```
Dim vbc As VBComponent
Set vbc = VBE.ActiveVBProject.VBComponents(sUFName)
With vbc.Designer.Controls("txtAnzeige")
    .Value = intVorgabe
    .BackStyle = Val(intBackStyle)
    .TextAlign = Val(intTextAlign)
    .SpecialEffect = Val(intSpecialEffect)
End With
```

Damit die Änderungen direkt angezeigt werden können, wird über die *OnTime*-Methode das UserForm geschlossen und direkt wieder angezeigt.

Listing 20.24 UserForm schließen und mit geänderten Eigenschaften neu starten

```
Sub subCallUFOnTime()
Dim oUF As Object
Dim bclose As Boolean: bclose = False
Dim bset As Boolean: bset = False
If VBA.UserForms.Count > 0 Then
    For Each oUF In VBA.UserForms
        If oUF.Name = sUFName Then
```

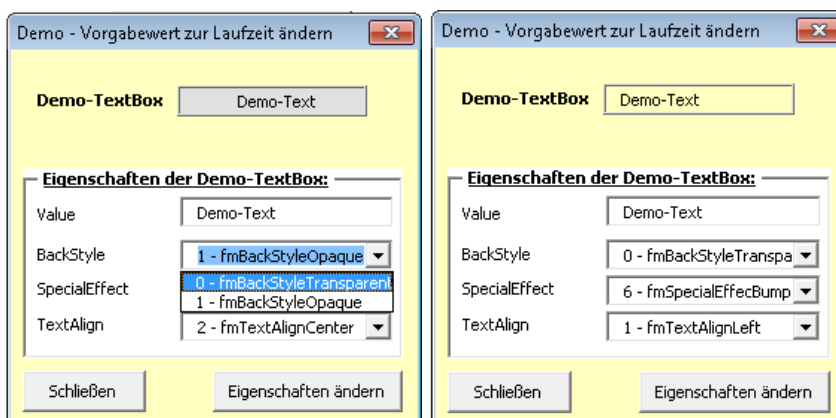
Listing 20.24 UserForm schließen und mit geänderten Eigenschaften neu starten (Fortsetzung)

```

Unload oUF
bclose = True
Exit For
End If
Next oUF
ElseIf VBA.UserForms.Count = 0 Then
    bclose = False
End If
If bclose = True Then
    Application.OnTime When:=Now + TimeValue("00:00:00"), _
        Name:="subCallUFOnTime"
    GoTo subCallUFOnTime_ende
End If
Dim vbc As VBComponent
Set vbc = VBE.ActiveVBProject.VBComponents(sUFName)
If vbc Is Nothing Then
    MsgBox "Fehler beim Zugriff auf die Komponente: " & sUFName, vbCritical, "Fehler!"
    GoTo subCallUFOnTime_ende
End If
With vbc.Designer.Controls("txtAnzeige")
    .Value = intVorgabe
    .BackStyle = Val(intBackStyle)
    .TextAlign = Val(intTextAlign)
    .SpecialEffect = Val(intSpecialEffect)
End With
vbc.Designer.Controls("txtVorgabe").Value = intVorgabe
vbc.Designer.Controls("cboBackStyle").Value = Val(intBackStyle)
vbc.Designer.Controls("cboTextAlign").Value = Val(intTextAlign)
vbc.Designer.Controls("cboSpecialEffect").Value = Val(intSpecialEffect)
frmChangeUFValues.Show vbModeless
subCallUFOnTime_ende:
End Sub

```

Abbildg. 20.21 Eigenschaften von UserForm-Controls dynamisch ändern



CD-ROM Das vollständige Beispiel finden Sie auf der CD-ROM zum Buch in der Datei `\Beispiele\Kap20\Bsp20_VorgabewertAendern.docm`.

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.

CD-ROM Die in diesem Abschnitt verwendeten Beispielprozeduren finden Sie auf der CD-ROM zum Buch in der Datei `\Beispiele\Kap20\Bsp20_01.docm` im Modul *modReferences*.

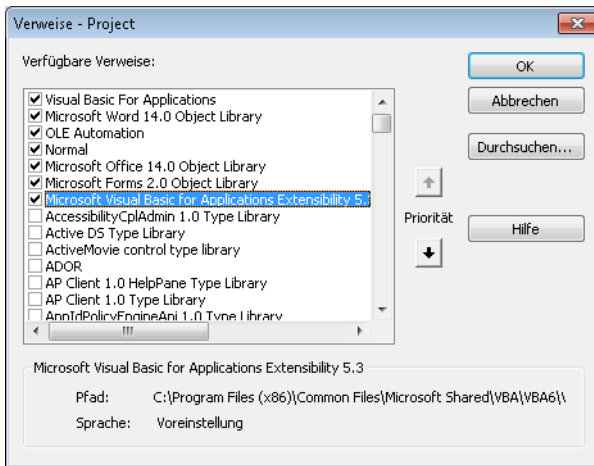
Übersicht über die gesetzten Verweise

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 Projekts ermitteln.

Das Listing 20.25 zeigt alle gesetzten Verweise, wie sie auch über den entsprechenden Menüpunkt (Abbildung 20.22) 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.

Über die References-Eigenschaft eines Projekts 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.

Abbildg. 20.22 Übersicht über die gesetzten und verfügbaren Verweise eines Projekts



Listing 20.25 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
```

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.

Verwendung der `AddFromGuid`-Methode

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 20.6 listet die GUID für die wichtigsten Verweisbibliotheken auf. Das Listing 20.26 veranschaulicht, wie diese Angaben programmtechnisch zu ermitteln sind; das Ergebnis ist in Abbildung 20.23 ersichtlich.

Tabelle 20.6 GUIDs für die Standardverweise in Office 14 (Office 2010 32-Bit und 64-Bit)

Name	GUID/Standardpfad/Beschreibung
VBA	{000204EF-0000-0000-C000-000000000046}
	C:\Programme\Gemeinsame Dateien\Microsoft Shared\VBA\VBA7\VBE7.DLL (Windows XP)
	C:\Program Files\Common Files\Microsoft Shared\VBA\VBA7\VBE7.DLL (Windows Vista / Windows 7)
	Visual Basic For Applications
Word	{00020905-0000-0000-C000-000000000046}
	C:\Programme\Microsoft Office\OFFICE14\MSWORD.OLB (Windows XP)
	C:\Program Files\Microsoft Office\Office14\MSWORD.OLB (Windows Vista)
	Microsoft Word 14.0 Object Library
stdole	{00020430-0000-0000-C000-000000000046}
	C:\WINDOWS\system32\Stdole2.tlb (Windows XP)
	C:\Windows\system32\stdole2.tlb (Windows Vista / Windows 7)
	OLE Automation
Office	{2DF8D04C-5BFA-101B-BDE5-00AA0044DE52}
	C:\Programme\Gemeinsame Dateien\Microsoft Shared\OFFICE14\MSO.DLL (Windows XP)
	C:\Program Files\Common Files\Microsoft Shared\OFFICE14\MSO.DLL (Windows Vista / Windows 7)
	Microsoft Office 14.0 Object Library
MSForms	{0D452EE1-E08F-101A-852E-02608C4D0BB4}
	C:\WINDOWS\System32\FM20.DLL (Windows XP)
	C:\Windows\system32\FM20.DLL (Windows Vista / Windows 7)
	Microsoft Forms 2.0 Object Library
VBIDE	{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 7 - 32-Bit)
	C:\Program Files (x86)\Common Files\Microsoft Shared\VBA\VBA6\VBE6EXT.OLB (Windows 7 - 64-Bit)
	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.

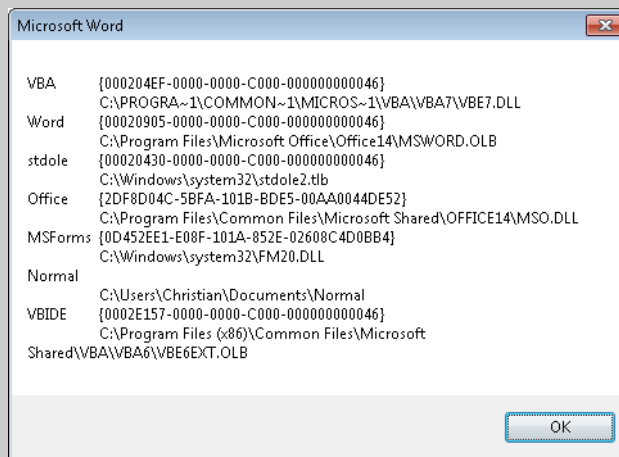
Listing 20.26 Ermitteln der gesetzten Verweise mit Pfad und GUID

```

Sub subListGUIDReferences()
    Dim strMSG As String
    Dim intRef As Integer
    With VBE.ActiveVBPProject.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. 20.23 Ausgabe der gesetzten Verweise mit Pfad und GUID



Das Listing 20.27 setzt einen neuen Verweis auf die Datei *Bsp20_Referenz.dotm*, sofern dieser noch nicht in der Liste der gesetzten Verweise aufgeführt ist. Damit das Beispiel funktioniert, kopieren Sie bitte die Datei *Bsp20_Referenz.dotm* von der CD-ROM aus dem Ordner *\Beispiele\Kap20* in den lokalen Ordner *C:\WordBuch\Beispiele\Kap20*. Wenn Sie die Datei an einen anderen Ort speichern, passen Sie bitte den vollständigen Pfad in Listing 20.27 an den gewählten Speicherort an.

Listing 20.27 Setzen eines Verweises auf eine Dokumentvorlage

```

Sub subSetReference()
    fktSetReferences "C:\WordBuch\Beispiele\Kap20\Bsp20_Referenz.dotm"
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
    End If
End Function

```

Listing 20.27 Setzen eines Verweises auf eine Dokumentvorlage (Fortsetzung)

```
Exit Function
End If
With VBE.ActiveVbProject.References
    For Each oRef In VBE.ActiveVbProject.References
        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 20.6).

ACHTUNG

Wenn Sie einem Projekt keinen eindeutigen Namen vergeben (über das Eigenschaften-Fenster im VBA-Editor), wird dieses Projekt in den Verweisen nur als „TemplateProject“ aufgeführt. Erst durch Vergabe eines Namens wird dieser auch in der Übersicht der Verweise angezeigt.

Ungültige Verweise korrigieren bzw. entfernen

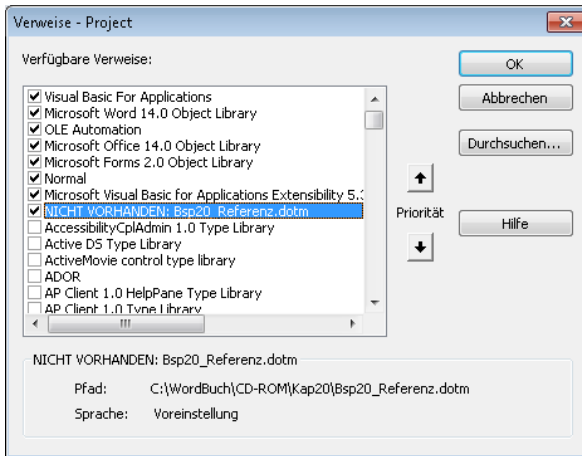
Normalerweise behalten die Standardverweise auf die Office/VBA-Bibliotheken ihre Gültigkeit, wenn Sie beispielsweise die Datei verschieben oder auf einen anderen Rechner kopieren. Auch wenn Sie die Datei mit einer älteren Word-Version öffnen, werden die Verweise durch Verweise auf die entsprechenden Bibliotheken der verwendeten Word-Version ersetzt.

Wenn Sie aber Verweise auf eigene Dokumentvorlagen setzen und dann die Datei auf einen anderen Rechner kopieren oder die Verweisdokumentvorlage entfernen, erhalten Sie ungültige Verweise in der Übersicht (siehe Abbildung 20.24). Diese ungültigen Verweise können zu Fehlern in den Makros führen und sogar die Stabilität von Word beeinträchtigen.

Die Gültigkeit der gesetzten Verweise können Sie mit der IsBroken-Eigenschaft prüfen. Diese Eigenschaft vergleicht einen Verweis mit den Verweiseinträgen in der Registry und liefert den Status der Gültigkeit zurück.

Mit dem Makro in Listing 20.28 überprüfen Sie die gesetzten Verweise. Wenn Sie die im Listing 20.27 hinzugefügte Verweisdatei *Bsp20_Referenz.dotm* entfernen oder umbenennen und das Makro ausführen, erhalten Sie den Hinweis in Abbildung 20.25, dass der Verweis ungültig sei.

Abbildg. 20.24 Fehlerhafte Einträge in der Verweisübersicht

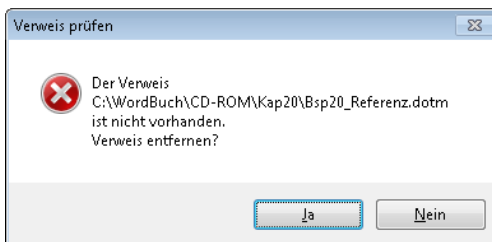


Listing 20.28 Prüfen der Verweise auf Gültigkeit

```
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. 20.25 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 20.29 überprüft alle gesetzten Verweise und versucht diese entweder anhand des Dateipfades oder der GUID wieder zu setzen.

Listing 20.29 Reparieren ungültiger Verweise

```
Sub ReSetReferences()
    Const c_Ref As String = "Verweis prüfen/setzen"
    Dim strGUID As String
    Dim strFile As String
    Dim strRef 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
                    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 Sub
```

Problematisch wird das Reparieren bei Verweisdateien (z.B. Dokumentvorlagen), wenn diese verschoben wurden. Für diesen Fall können Sie das Listing 20.29 dahingehend modifizieren, dass Sie optional den neuen Pfad zur Datei mit angeben. In Listing 20.30 werden alle ungültigen Verweise dahingehend geprüft, ob die angegebene Verweisdatei *Bsp20_Referenz.dotm* betroffen ist. Ist dies der Fall, wird nach Bestätigung (Abbildung 20.26) 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.

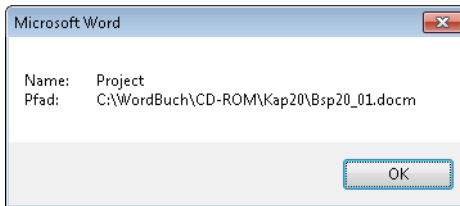
Listing 20.30 Ungültige Verweise ersetzen

```

Sub subTestAndReSetReference()
    fktReSetReferences "C:\WordBuch\Beispiele\Kap20\Bsp20_Referenz.dotm"
End Sub

Function fktReSetReferences(Optional strRefPath)
    Const c_Ref As String = "Verweis prüfen/setzen"
    Dim strGUID As String
    Dim strFile As String
    Dim strRef As String
    Dim oRef As Reference
    Dim bExists As Boolean: bExists = True
    Dim ret As Integer
    If IsMissing(strRefPath) = False Then
        If Dir(strRefPath) = "" Then
            MsgBox "Die angegebene Verweisdatei existiert nicht:" & _
                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. 20.26 Ungültigen Verweis ersetzen

WICHTIG Word reagiert mitunter sehr empfindlich, wenn ein Verweis ungültig oder nicht vorhanden ist. So werden dann einige Funktionen aus der *VBE6.DLL* bzw. *VBE7.DLL* als Fehler markiert, die ohne diesen Verweisfehler einwandfrei funktionieren. Zu diesen Funktionen gehören u.a. die *Right*-Funktion und *Chr*-Funktion.

Als Abhilfe können Sie das Elternobjekt VBA der Funktion voranstellen:

VBA.Chr

bzw.

VBA.Right

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 824) und auf das aktive VBA-Projekt (Seite 825) mit seinen darin enthaltenen Modulen und Benutzerformularen (Seite 827) zugreifen können
- Ein weiterer Schwerpunkt war der Zugriff auf den VBA-Code eines Projekts (Seite 830) und das Auslesen bestimmter Zeilen (Seite 831 und Seite 832)
- Nach dem lesenden Zugriff auf den VBA-Code wurde Ihnen gezeigt, wie Sie in einem Projekt gezielt einzelne VBA-Codezeilen suchen, ersetzen und entfernen können (Seite 837)
- 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 840), aber auch einzelne Codezeilen in ein Projekt einzufügen (Seite 843 und Seite 844).
- Neben dem Einfügen von VBA-Code haben Sie weiterhin gesehen, wie Sie Steuerelemente in ein UserForm einfügen (Seite 847) und ein so erstelltes UserForm zur Laufzeit anzeigen lassen können (Seite 851)
- Wie Sie zur Laufzeit auf die Eigenschaften einzelner UserForm-Steuerelemente zugreifen und diese ändern können, wurde auf Seite 854 angesprochen

- Abschließend wurde Ihnen in diesem Kapitel gezeigt, wie Sie während der Bearbeitung von Makros benötigte Verweise auf Bibliotheken ermitteln (Seite 856) und ggf. neu setzen können (Seite 857)

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 Onlinehilfe des Visual Basic-Editors als auch im Internet auf der MSDN-Homepage von Microsoft ([http://msdn.microsoft.com/en-us/library/bb726434\(v=office.12\).aspx](http://msdn.microsoft.com/en-us/library/bb726434(v=office.12).aspx)) im Bereich Office-Development.

Teil E

XML und Smart-Technologien

In diesem Teil:

Kapitel 21	Word und XML: Eine Einführung	869
Kapitel 22	Word Open XML-Dateiformat	915



Kapitel 21

Word und XML: Eine Einführung

In diesem Kapitel:

Was ist XML und wozu dient es?	870
Welche Aufgabe erfüllt XML in Word?	876
XML-Bestandteile	877
XML-Schema-Definitionen	883
XML-Daten manipulieren	893
XML in Word	898
WordProcessingML außerhalb von Word generieren	912
Zusammenfassung	914

Word 2003 stellte einiges an neuer Funktionalität zur Verfügung, unter anderem Werkzeuge für die Arbeit mit XML. Nachfolgeversionen haben weiter darauf gebaut. Diese sind hauptsächlich an Entwickler gerichtet, befinden sich aber teilweise auch in der Benutzerschnittstelle und können von jedem Anwender bedient werden.

XML dient mehreren Zwecken, die in diesem und in 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 Smarttag-Funktionalität (die ab Office 2010 missbilligt wurde)
- als Grundlage für die Smart Document-Funktionalität (die ab Office 2010 missbilligt wurde) und die darauf aufbauenden VSTO-Dokumentlösungen (siehe Kapitel 10)

Um diese Funktionalität einzusetzen, benötigen Sie Grundkenntnisse der XML-Technologie. Im vorliegenden Kapitel bieten wir eine kurze Übersicht dieses breit gefächerten Gebiets, gewichtet hinsichtlich der Bedeutung für die Arbeit mit Word, und weisen auf weitere Informationsquellen hin.

Ab Word 2007 ist das standardmäßige Dateiformat ab sofort XML, allerdings entspricht es nicht genau dem Dateiformat eines in Word 2003 im XML-Format gespeicherten Dokuments. Eine Variation des in Word 2003 eingeführte *WordProcessingML* definiert nach wie vor den Dokumentinhalt. Strukturelle Teile wie eingebettete Objekte, Grafiken und Formatvorlagen werden aber 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. Dieses neue Konzept heißt *OpenXML* und wird im Kapitel 22 kurz vorgestellt.

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, sodass 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 21.1 sowie in Listing 21.2 ersichtlich.

Listing 21.1 Inhalt von *Bsp21_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 21.2 Inhalt von *Bsp21_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>
```

CD-ROM Die Beispieldateien *Bsp21_01.htm* sowie *Bsp21_02.xml* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap21*.

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 bzw. Textdatei.
2. Geben Sie den Inhalt eines Listings genau so ein wie hier abgedruckt (inklusive Großschreibung).
3. Rufen Sie *Datei/Speichern* auf 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 *"Bsp21_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.

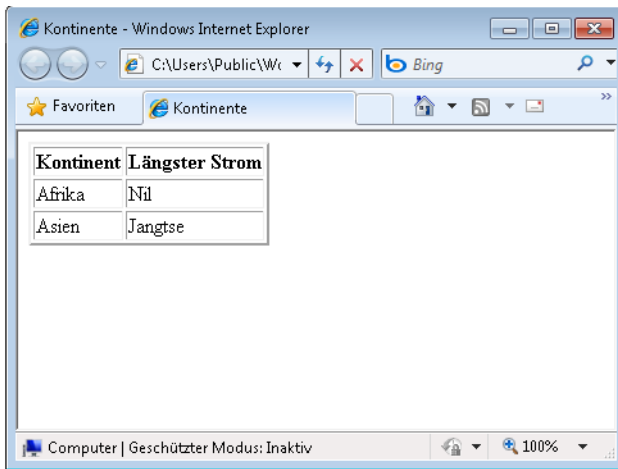
XML-Vokabular

Beim ersten Blick auf Listing 21.1 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.

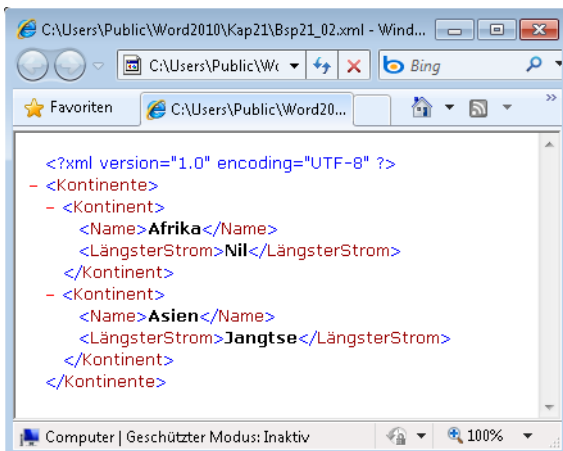
Obwohl es angeblich die gleichen Informationen enthält, sieht Listing 21.2, mit Ausnahme der ersten Zeile sowie dem Gebrauch der Zeichen `<>`, entscheidend anders aus. Wenn beide angeblich XML sind, warum gleichen sie einander so wenig?

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 21.1 wurde mit dem Vokabular XHTML, das Listing 21.2 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 21.1 sowie Abbildung 21.2) veranschaulicht.

Abbildg. 21.1 Die XML-Datei *Bsp21_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. 21.2 Die XML-Datei *Bsp21_02.xml*, interpretiert vom Internet Explorer

Die KML-Datei in Abbildung 21.2 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 »KML«-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 visuelle Darstellung, ist Sinn der »KML«-Datei. Die XHTML-Datei sagt eigentlich nichts über die Struktur der Daten aus. Der Menschenverstand erkennt, dass Kontinente in einer Spalte aufgelistet sind, während die Namen von Flüssen sich in der anderen befinden. Diese Einteilung und Zugehörigkeit geht jedoch nicht klar aus den Tags der XHTML-Datei vor, wie im Fall der »KML«-Datei.

HINWEIS

Mehr Informationen über die XML-Sprache finden Sie auf der deutschsprachigen Seite <http://edition-w3c.de/TR/2000/REC-xml-20001006/>.

XML-Daten transformieren



Diese strukturierte Darstellung der Daten macht es einfach, sie zu transformieren, sodass 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 21.3).

Listing 21.3 Inhalt von *Bsp21_03.xml* – XML-Kontinente-Datei, die auf eine XSLT-Transform verweist

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="Bsp21_03.xsl"?>
<Kontinente>
  <Kontinent>
    <Name>Afrika</Name>
    <LängsterStrom>Nil</LängsterStrom>
  </Kontinent>
  <Kontinent>
    <Name>Asien</Name>
    <LängsterStrom>Jangtse</LängsterStrom>
  </Kontinent>
</Kontinente>
```

Da die Pfadangabe für href= relativ ist (nur der Dateiname), muss sich die XSLT-Datei im gleichen Ordner mit der XML-Datei befinden. Sie enthält die Anweisungen in Listing 21.4. Wird *Bsp21_03.xml* im Internet Explorer geöffnet, muss es ähnlich aussehen wie die Abbildung 21.1.

Listing 21.4 Inhalt von *Bsp21_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>
```

Listing 21.4 Inhalt von *Bsp21_03.xsl* – XSL-Transform für die Kontinente-Datei (Fortsetzung)

```

<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>

```

CD-ROM Die Beispieldatei *Bsp21_03.xml* sowie die Transformationsdatei *Bsp21_03.xsl* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap21*.

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, sodass 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, sodass 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 in anders strukturiertes 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 *Bsp21_01.htm* ein:

```
<?xml-stylesheet type="text/xsl" href="BSP21_04.xsl"?>
```
- Entfernen Sie die Zeile

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```
- Speichern Sie die Datei unter den Namen *Bsp21_04.xml*
- Geben Sie den Code aus Listing 21.5 in eine neue Datei ein und speichern Sie diese unter dem Namen *Bsp21_04.xsl*
- Öffnen Sie *Bsp21_04.xml* im Internet Explorer. Das Resultat sollte wie in Abbildung 21.1 aussehen.

Der XSLT-Code transformiert XHTML in gewöhnliches, korrektes HTML.

Listing 21.5 Inhalt von *Bsp21_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« ab Seite 872 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 ab Word 2003 unterstützt diese Konvertierung. Zudem kann jede Version ab Word 2000 Dokumente im Word 2007-Dateiformat öffnen und speichern, sofern das Kompatibilitäts-pack installiert ist.
- 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.

HINWEIS

Diese Funktionalität wurde von einer kanadischen Firma als Patentverletzung angeklagt. Im August 2009 wurde der Firma Recht gegeben und Microsoft musste die Anzeige des XML-Inhalts, der mit einem eigenen Vokabular verbunden ist, beim Öffnen eines Dokuments für alle in Nord-Amerika verkauften Word 2007-Versionen unterbinden. Diese Unterstützung fehlt in allen Ausgaben von Office 2010. Die Benutzer- und Objektmodellschnittstellen für die Herstellung solcher Dokumente sind jedoch weiterhin vorhanden. Ein Vokabular kann in ein Dokument eingebunden, Daten damit erfasst und mit dem Dokument gespeichert werden. Sie stehen danach für die Weiterverarbeitung bereit. Die XML-Knotenpunkte gehen jedoch beim nächsten Öffnen des Dokuments in Word verloren. Da diese Funktionalität wichtige Punkte für die Diskussion über XML aufzeigt, haben wir sie in der dritten Auflage belassen.

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« ab Seite 903 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*, *.docx* 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*).
- 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 oder VSTO Dokument-Lösung (Kapitel 10).
- 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
- Ab Word 2007 können Inhaltssteuerelemente mit einem XML-Teil verbunden werden (diese sind vom erwähnten Gerichtsentscheid *nicht* betroffen)

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)).

XML-Bestandteile

Die XML-Sprache allein genügt nicht, um nützliche Lösungen bereitzustellen. Im Abschnitt »XML-Daten transformieren« ab Seite 874 sahen Sie beispielsweise, dass es einer Transformation bedarf, um die Daten auf benutzerfreundliche Art darzustellen. Die Tabelle 21.1 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 21.1 Die Mitglieder der XML-Familie

Standard	Beschreibung
XML	Die <i>Extensible Markup Language</i>
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 <i>Document Object Model</i> . 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	<i>Extensible Stylesheet Language: Transformations</i> . 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 21.4 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). Ein Attribut besteht aus drei Teilen: der Bezeichnung, gefolgt von einem Gleichheitszeichen (=) sowie dem Inhalt in 'einfachen' oder "doppelten" Anführungszeichen.

Eine gültige Element- oder Attributbezeichnung 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 verwenden 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 21.6 veranschaulicht.

Listing 21.6 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>
```

Listing 21.6 XML- Datei mit fehlenden sowie mehrfachen Datenelementen *(Fortsetzung)*

```
<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« ab Seite 872 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 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« ab Seite 883 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 & (für &) und " (für "). Fünf sind vordefiniert: & (&), ' ('), > (>) < (<) sowie " (")
- 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 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
]]>

```

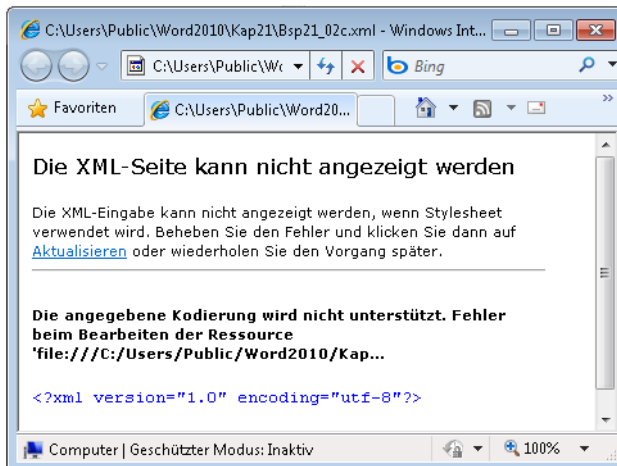
Wohlgeformte und gültige XML-Dokumente

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 21.3 veranschaulicht.

Abbildg. 21.3 Das Dokument ist nicht wohlgeformt, weil sich ein Syntaxfehler in der XML-Deklaration befindet



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

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 21.1 ist wohlgeformt und gültig, weil sie die Regeln des XHTML DTD befolgt. Die XML-Datei in Listing 21.2 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 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 21.7, das das Vokabular der »KML«-Datei aus dem Abschnitt »XML-Vokabulare« (siehe Seite 872) umschreibt. Es veranschaulicht viele Grundlagen der XSD.

Listing 21.7 Inhalt von *Bsp21_05.xsd* – XML-Schema für die »KML«-Datei (Listing 21.2). 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>
```

Listing 21.7 Inhalt von *Bsp21_05.xsd* – XML-Schema für die »KML«-Datei (Listing 21.2). Bitte beachten Sie, dass auch dieses ein XML-Dokument ist. (*Fortsetzung*)

```
<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>
```

CD-ROM

Die Beispieldatei *Bsp21_05.xsd* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap21*.

Ein Element wird durch das Tag `<xsd:element>` definiert. Das Attribut `name` ist erforderlich. Die Elemente `Kontinente` und `Kontinent` sind beide als komplexe Typen deklariert: `<xsd:complexType>`; demzufolge umfassen sie weitere Elemente. In beiden Fällen ist die nächste Anweisung `<xsd:sequence>`; was bedeutet, dass jene Elemente in der angegebenen Reihenfolge erscheinen müssen.

HINWEIS

Dieses Schema definiert zwei Elemente auf der obersten Ebene: `Kontinente` sowie `Kontinent`. Bekanntlich darf ein XML-Dokument nur ein Wurzelement haben, dieses Schema bewahrt also nicht vor diesem möglichen Fehler.

Als Nächstes werden die verschachtelten Elemente deklariert. `Kontinente` darf nur Elemente des Typs `Kontinent` enthalten, und zwar eine unbegrenzte Menge (bestimmt durch das Attribut `maxOccurs="unbounded"`) davon. Das Element `Kontinent` seinerseits darf zwei Elemente enthalten: `Name` sowie `LängsterStrom`. Mindestens einmal muss das Element `Name` vorkommen (bestimmt durch das Attribut `minOccurs="1"`); während `LängsterStrom` fehlen darf (bestimmt durch das Attribut `minOccurs="0"`).

Beide dieser Elemente sind vom Typ `OrtsNameTyp`. Es handelt sich hier um einen benutzerdefinierten Typ, der seinerseits ein einfacher Typ ist (darf nur Zeichendaten enthalten). `<xsd:restriction base="xsd:string">` bedeutet, dass der Inhalt eine Zeichenkette mit einer maximalen Länge von 20 Zeichen (`<xsd:maxLength value="20" />`) sein muss.

HINWEIS

Die Attribute `minOccurs` und `maxOccurs` sind nicht erforderlich; der standardmäßige Wert beträgt in beiden Fällen »1« (das Element muss einmal vorkommen, und nicht mehr als einmal).

XSD enthält viele einfache Datentypen für Zeichenketten, numerische Werte sowie Datenangaben. Eine Auflistung finden Sie in »XML Schema Teil 0: Einführung«. Diese können auch, wie hier veranschaulicht, in bestimmten Kombinationen mit Begrenzungen zusammengestellt werden, um benut-

zerdefinierte 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 21.8 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 21.8 Inhalt von *Bsp21_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>
```

CD-ROM

Die Beispieldatei *Bsp21_06.xsd* finden Sie auf der CD-ROM zum Buch im Ordner `\Beispiele\Kap21`.

Die bislang vorgestellten Schemas sind eher datenzentrisch ausgerichtet. Erlaubt ist eigentlich nur eine Auflistung von Kontinenten und ihren Flüssen, ähnlich wie eine Datenbanktabelle. Was ist, wenn ein weniger strukturiertes Format erwünscht ist, und die Informationen in einem Textfluss markiert sein sollen:

```
<Kontinente>Jeder Kontinent hat seine Flüsse. Der bekannteste ist meistens der größte oder
längste. <Kontinent>Der längste <Name>Afrika</Name>s ist sehr bekannt, nicht zuletzt, weil
Agatha Christie ein Stück schrieb, das sich darauf abspielt: "Tod auf dem
<LängsterStrom>Nil</LängsterStrom>"</Kontinent></Kontinente>
```

Diese Art von Markup ist HTML-ähnlich. Um es in einem Schema zu definieren, muss das Attribut `mixed` in der Element-Definition vorhanden sein:

```
<xsd:element name="Kontinente">
  <xsd:complexType mixed="true">
```

Auch mit `mixed="true"` bleibt die erlaubte Zusammensetzung von Elementen im XML-Dokument strukturiert: sie müssen immer noch in der vordefinierten Hierarchie und Reihenfolge erscheinen. Um eine zwanglose Zusammenstellung von Elementen zu erreichen, wie im folgenden Text, wird das XSD-Element `choice` benötigt.

```
<Text>...Nach einer langen Reise durch <Kontinent>Afrika</Kontinent>, fließt der
<Strom>Nil</Strom> an <Stadt>Kairo</Stadt>, der größten Stadt Aegyptens, vorbei</Text>
```

Das Element `<xsd:choice minOccurs="0" maxOccurs="unbounded" >` wird an die Stelle von `sequence` gesetzt. Die Attribute `minOccurs` und `maxOccurs` in dieser Kombination bedeuten, dass die Anzahl der Elemente unbegrenzt ist (es dürfen auch keine vorkommen):

```
<xsd:element name="Text">
  <xsd:complexType mixed="true">
    <xsd:choice minOccurs="0" maxOccurs="unbounded" >
      <xsd:element name="Kontinent" type="xsd:string" />
      <xsd:element name="Strom" type="xsd:string" />
      <xsd:element name="Stadt" type="xsd:string" />
    </xsd:choice>
  </xsd:complexType>
</xsd:element>
```

Letztlich stellt sich die Frage, wie alles erlaubt werden kann; jedes Element, jedes Attribut. Auch das ist möglich, mit dem Element `<any>` sowie dem Attribut `<anyAttribute>`. Damit wird jede Strukturierung dem Schema entzogen, was seinem Zweck zuwider spricht. Viele XML-Parser haben Probleme, Dokumente mit solchen Schemas zu validieren.

XML-Namensräume und Schemas

Das Konzept des Namensraums spielt eine wichtige Rolle bei der Implementierung von XML in Word. 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, sodass 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 Zeilen 3 bis 6 in Listing 21.9 veranschaulichen.

Listing 21.9 Inhalt von *Bsp21_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.KAP21Beispiel.com/BSP07"
  xmlns:bu="http://www.KAP21Beispiel.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>
```

CD-ROM Die Beispieldatei *Bsp21_07.xsd* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap21*.

Dieses Schema legt fest, dass sich im XML-Dokument ein Element namens *Buch* befinden muss. Verschachtelt in diesem Element ist ein weiteres Element namens *ID*, das eine Ganzzahl im Bereich zwischen 1000 und 1999 sein muss.

Das erste Attribut des `<xsd:schema>`-Elements teilt dem Prozessor (einem XML-Parser beispielsweise) mit, dass jeder Elementname, dem das Namensraumpräfix `xsd` voransteht, dem Namensraum mit dem URI `http://www.w3.org/2001/XMLSchema` gehört:

```
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
```

HINWEIS Ein URI ist eine eindeutige Bezeichnung (*Uniform Resource Identifier*). URIs sind oft Web-Adressen, müssen es aber nicht sein. Es ist nicht gesagt, dass das Schema sich dort befindet oder dass die Webadresse überhaupt existiert. Wichtig ist nur, dass die Bezeichnung eindeutig ist, was bei Webadressen 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.KAP21Beispiel.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.KAP21Beispiel.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 21.10 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 21.10 Inhalt von *Bsp21_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.KAP21Beispiel.com/BSP08"
  xmlns:cd="http://www.KAP21Beispiel.com/BSP08"
```

Listing 21.10 Inhalt von *Bsp21_08.xsd* – XML-Schema für eine einzige CD (Fortsetzung)

```

    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>

```

CD-ROM

Die Beispieldatei *Bsp21_08.xsd* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap21*.

Jetzt werden die zwei Schemas in einem »Überschema« vereint, wie Listing 21.11 veranschaulicht.

Listing 21.11 Inhalt von *Bsp21_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.KAP21Beispiel.com/BSP09"
  xmlns:bu="http://www.KAP21Beispiel.com/BSP07"
  xmlns:cd="http://www.KAP21Beispiel.com/BSP08"
  elementFormDefault="qualified">

  <xsd:import
    namespace="http://www.KAP21Beispiel.com/BSP07"
    schemaLocation="Bsp21_07.xsd"/>
  <xsd:import
    namespace="http://www.KAP21Beispiel.com/BSP08"
    schemaLocation="Bsp21_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>

```

CD-ROM Die Beispieldatei *Bsp21_09.xsd* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap21*.

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 (sodass 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 (Schemas), sind in vielen XML-Anwendungen schon eingebaut. Das Vorhandensein des korrekten URI genügt in diesen Fällen.

Das vom Schema definierte Wurzelement heißt *Sammlung*. Darunter dürfen beliebig viele `bu:Buch`- und `cd:CD`-Elemente verschachtelt werden, in beliebiger Reihenfolge (`<xsd:choice minOccurs="0" maxOccurs="unbounded" >`).

Erforschen wir nun die Wirkung dieser Schemas und Namensräume und betrachten zunächst die XML-Datei in Listing 21.12, die lediglich das Wurzelement *Sammlung* enthält. Diese Datei ist sowohl wohlgeformt als auch gültig.

Listing 21.12 Inhalt von *Bsp21_10.xml* – XML-Dokument mit einer leeren Sammlung

```
<?xml version="1.0" encoding="UTF-8" ?>
<Sammlung
  xmlns="http://www.KAP21Beispiel.com/BSP09" >
</Sammlung>
```

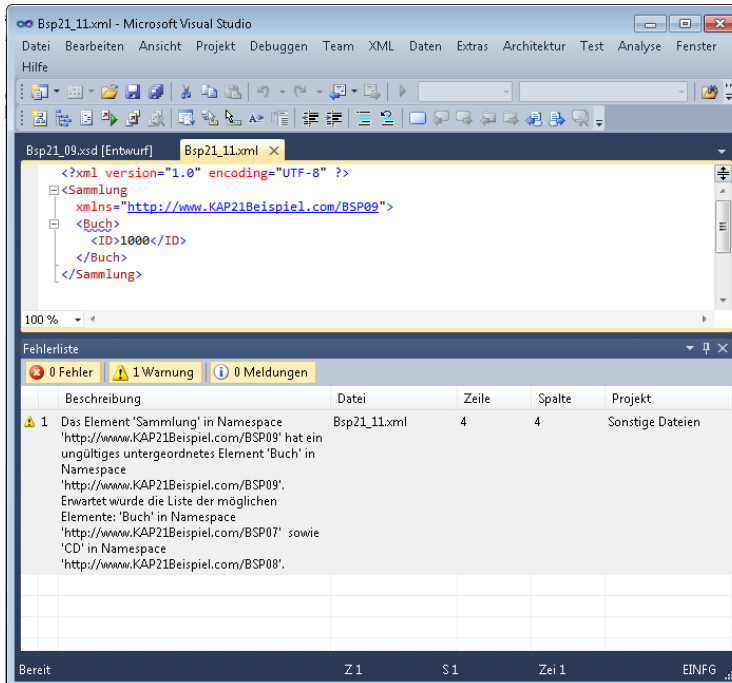
HINWEIS Da kein Namensraumpräfix Teil der `xmlns`-Deklaration ist, wird dieser Namensraum als standardmäßiger der XML-Datei in Listing 21.12 betrachtet. Der XML-Parser wird annehmen, dass alle Elemente ohne Namensraumpräfix (unqualified) diesem Namensraum angehören.

Wird diese Datei um ein *Buch*-Element und seine ID erweitert, wie in Listing 21.13, wird es ungültig, weil das Element *Buch* nicht dem standardmäßigen Namensraum (dem *Sammlung*-Schema) angehört (Abbildung 21.4).

Listing 21.13 Inhalt von *Bsp21_11.xml* – XML-Dokument mit einem *Buch*-Element; erster Versuch

```
<?xml version="1.0" encoding="UTF-8" ?>
<Sammlung
  xmlns="http://www.KAP21Beispiel.com/BSP09" >
  <Buch>
    <ID>1000</ID>
  </Buch>
</Sammlung>
```

Abbildg. 21.4 Die XML-Datei ist nicht gültig, wenn sie mit dem Schema 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. Wir können »XML Notepad 2007« (<http://www.microsoft.com/downloads/details.aspx?familyid=72d6aa49-787d-4118-ba5f-4f30fe913628&displaylang=en>) und Architag's »XRay XML Editor« (<http://architag.com/xray/>) empfehlen. 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 21.5 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 haben, lohnt es sich, die mitgelieferten Werkzeuge anzuschauen. Im Menü *XML* stehen Befehle, um ein Schema zu generieren, Schemas einzubinden sowie das Ergebnis einer Transformation zu betrachten.

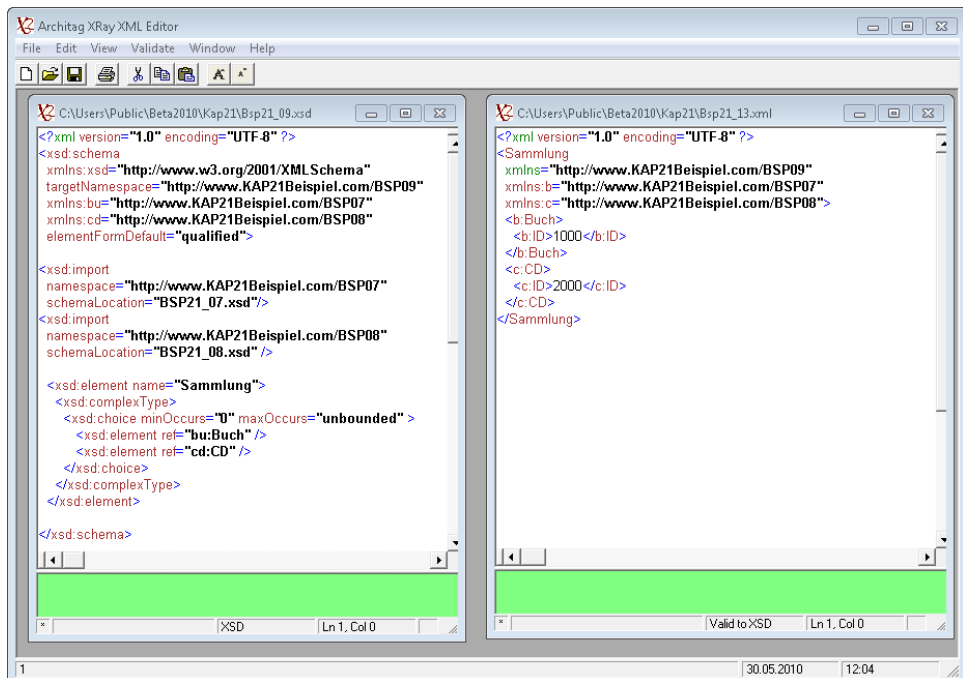
Der nächste Versuch, in Listing 21.14, enthält zusätzlich den Namensraum für Buch-Elemente und den Elementnamen wurden Namensraumpräfixe vorangestellt. Diese Version ist gültig. Bitte beachten Sie, dass es erlaubt ist, in der XML-Datei ein anderes Namensraumpräfix als das im Schema definierte zu verwenden.

Listing 21.14 Inhalt von *Bsp21_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.KAP21Beispiel.com/BSP09"
  xmlns:b="http://www.KAP21Beispiel.com/BSP07">
  <b:Buch>
    <b:ID>1000</b:ID>
  </b:Buch>
</Sammlung>
```

Schließlich veranschaulichen Listing 21.15 und Abbildung 21.5 ein gültiges XML-Dokument mit *Buch*- sowie *CD*-Element.

Abbildg. 21.5 »Alles im grünen Bereich«: Das XML-Dokument (rechts) erfüllt die Bedingungen des Schemas (links) und ist gültig



Listing 21.15 Inhalt von *Bsp21_13.xml* – Gültiges XML-Dokument mit je einem *Buch*- sowie *CD*-Element

```
<?xml version="1.0" encoding="UTF-8" ?>
<Sammlung
  xmlns="http://www.KAP21Beispiel.com/BSP09"
  xmlns:b="http://www.KAP21Beispiel.com/BSP07"
  xmlns:c="http://www.KAP21Beispiel.com/BSP08" >
  <b:Buch>
    <b:ID>1000</b:ID>
  </b:Buch>
  <c:CD>
    <c:ID>2000</c:ID>
  </c:CD>
</Sammlung>
```

XML-Daten manipulieren

Wie im Abschnitt »XML-Daten transformieren« ab Seite 874 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 im Speicher 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 aufwendigere 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 installierte Version und Hilfedatei ist MSXML 5.0 für Microsoft Office Applications; die gegenwärtig neueste Version ist MSXML 6.0 SP2.

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. Die SDK der neuesten Version kann von <http://www.microsoft.com/downloads/details.aspx?familyid=D21C292C-368B-4CE1-9DAB-3E9827B70604&displaylang=de> heruntergeladen werden.

.NET Framework stellt System.XML-Klassen für die Arbeit mit XML-Dokumenten, XML-Schemas, XSLT, XPath und das XML-DOM zur Verfügung. Diese sind in der MSDN-Bibliothek dokumentiert.

Das Word-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, indem er seinem Projekt eine Referenz auf die Bibliothek *Microsoft XML* zufügt.

XSLT

Viele Aufgaben können mit XSLT (Extensible Stylesheet Language Transformation) statt auf Basis des DOM gelöst werden. Im Zusammenspiel mit XPath wird es zum schlagkräftigen Werkzeug, das die Daten eines XML-Dokuments in beliebiger Reihenfolge herauslesen und neu zusammenstellen kann. Die Grundregeln für Transformationen bilden ein XML-Vokabular. Eine *.xsl*-Datei ist ein wohlgeformtes und gültiges XML-Dokument, das beschreibt, wie die Daten einer *.xml*-Datei dynamisch darzustellen sind. In diesem Abschnitt werden wir lediglich einen Überblick zu dieser großen und komplexen Sprache darstellen.

Mittlerweile gibt es die Versionen 1.0 und 2.0. Der Microsoft XML-Parser, MSXML, 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.

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. Kon-

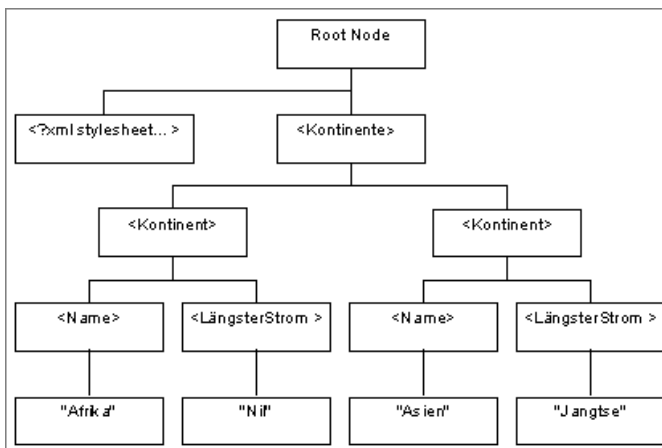
zeptuell kann eine Transformation mit einer SQL-Anweisung verglichen werden, die beschreibt, welche Daten auszuwählen sind. Es können beispielsweise alle Adresse-Elemente, deren *Art* »Geschäft« ist, ausgewählt werden, oder nur die Flüsse, die sich in Afrika befinden.

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 21.6).

Abbildg. 21.6 Hierarchie der Knotenpunkte des XML-Dokuments *Kontinente*



Die XSLT in Listing 21.3 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 unterstell-

ten 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 21.16 veranschaulicht. Darin basiert die Transformation des Kontinente-XML-Dokuments auf Mustervergleichen statt auf programmtechnischen Konstrukten.

Listing 21.16 Inhalt von *Bsp21_14.xsl* – Eine Transformation des Kontinent-XML-Dokuments, basierend auf Mustervergleichen

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:template match="/">
    <html>
      <head>
        <title>Kontinente</title>
      </head>
      <body>
        <table border="2">
          <tr>
            <th>Kontinent</th>
            <th>Längster Strom</th>
          </tr>
          <xsl:apply-templates select="Kontinente/Kontinent" />
        </table>
      </body>
    </html>
  </xsl:template>

  <xsl:template match=" Kontinent" >
    <tr>
      <xsl:apply-templates />
    </tr>
  </xsl:template>

  <xsl:template match="Name|LängsterStrom" >
    <td><xsl:apply-templates /></td>
  </xsl:template>

</xsl:stylesheet>
```

CD-ROM Die Beispieldateien *Bsp21_14.xsl* mit der Transformation sowie *Bsp21_15.xml*, die auf diese hinweist, finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap21*.

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 21.6), 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 21.1) 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.

XML in Word

Dieser Abschnitt bietet einen Überblick des WordProcessingML-Vokabulars und stellt die Werkzeuge für die Arbeit mit Word-fremden XML-Vokabularen vor. Das erste von Word unterstützte Vokabular war *WordProcessingML* in Version 2003. Word 2007 und Word 2010 arbeiten standardmäßig mit einer Variation davon, das jedoch neue Funktionalität (wie Inhaltssturelemente) unterstützt und als Teil des neuen Dateiformats *OpenXML* integriert ist.

Da es im Vergleich etwas einfacher ist, stellen wir hier kurz die 2003 WordProcessingML vor, um die Grundlagen besser zu veranschaulichen. Spätere Versionen von Word können dieses Dateiformat lesen und schreiben. Dies dient als Basis für eine vertiefte Diskussion des neueren Vokabulars in Kapitel 22.

HINWEIS

Alle in diesem Abschnitt vorgestellten Beispiele mit Word 2003 WordProcessingML funktionieren weiterhin in Word 2007 und Word 2010. Auch wenn die rosa-roten XML-Knotenpunkte beim Öffnen eines XML-Dokuments nicht angezeigt werden (Abbildung 21.15), funktioniert eine Transformation.

WordProcessingML

Bei Microsofts WordProcessingML handelt es sich um ein XML-Vokabular, das ein Word 2003-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 21.17; wie Word dieses auf dem Bildschirm darstellt, sehen Sie in Abbildung 21.7.

Listing 21.17 Inhalt von *Bsp21_16.xml* – Die *Kontinente*-Tabelle in WordProcessingML

```

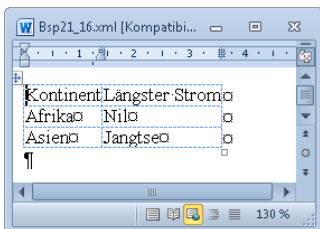
<?xml version="1.0" encoding="UTF-8"?>
<w:wordDocument
  xmlns:w="http://schemas.microsoft.com/office/word/2003/wordml" >
<w:body>
  <w:tbl>
    <w:tr>
      <w:tc><w:p><w:r><w:t>Kontinent</w:t></w:r></w:p></w:tc>
      <w:tc><w:p><w:r><w:t>Längster Strom</w:t></w:r></w:p></w:tc>
    </w:tr>
    <w:tr>
      <w:tc><w:p><w:r><w:t>Afrika</w:t></w:r></w:p></w:tc>
      <w:tc><w:p><w:r><w:t>Nil</w:t></w:r></w:p></w:tc>
    </w:tr>
    <w:tr>
      <w:tc><w:p><w:r><w:t>Asien</w:t></w:r></w:p></w:tc>
      <w:tc><w:p><w:r><w:t>Jangtse</w:t></w:r></w:p></w:tc>
    </w:tr>
  </w:tbl>
</w:body>
</w:wordDocument>

```

CD-ROM

Die Beispieldatei *Bsp21_16.xml* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap21*.

Abbildg. 21.7 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 würden.

CD-ROM Eine bearbeitete Version des Ergebnisses (mit Zeilenschaltungen und Einzügen, um die Strukturen hervorzuheben) befindet sich in der Datei *Bsp21_17.xml* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap21*.

PROFITIPP

Wie im Abschnitt »XML-Dokumentinstanz, Markup und Inhalt« ab Seite 881 erwähnt, veranlasst die Verarbeitungsanweisung den Internet Explorer, eine WordProcessingML-Datei in Word zu öffnen. Das macht es schwierig, sich einen Überblick über den Dateiinhalt zu verschaffen. Entweder muss die Verarbeitungsanweisung entfernt oder der für die Verknüpfung verantwortliche Eintrag in der Registry geändert werden:

1. Geben Sie *regedit* in die Suchbox von Windows ein und klicken auf  drücken.
2. Navigieren Sie zum Eintrag `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Office\[nn].0\Common\Filter\text\xml`, wobei »nn« der Versionsnummer von Word entspricht (11 für 2003; 12 für 2007; 14 für 2010).

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. Es folgen weitere »Dokumenteigenschaften« (Einstellungen) wie die aktuelle Ansicht beim Speichern des Dokuments und die angehängte Vorlage. Erst dann folgt der Dokumentkörper.

Eine detaillierte Behandlung des WordProcessingML-Vokabulars würde den Rahmen dieses Buchs sprengen, wir möchten aber auf einige Besonderheiten aufmerksam machen.

HINWEIS Die Einzelheiten von und der Umgang mit WordProcessingML werden in dem Buch »Office 2003 XML« von Evan Lenz, Mary McRae und Simon St. Laurent (O'Reilly, ISBN-13: 978-0-596-00538-2) 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 dazugehörigen, lästigen Nachteilen.
- Dabei ist es nicht unbedingt notwendig, alle Tags und Attribute einzufügen, die Word beim Speichern eines Dokuments einsetzt. Wie das Listing 21.17 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. Das folgende Codefragment, das 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 Elementzusammensetzung benötigt. Bitte beachten Sie die Verwendung der Textläufe (`<w:r>`)

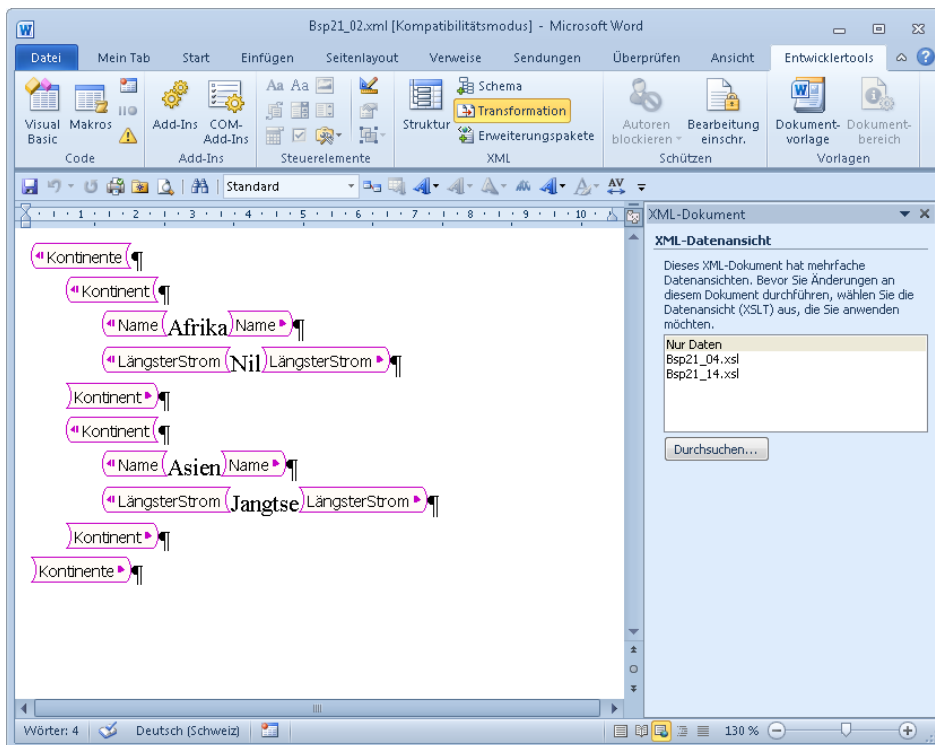
sowie Textlaufeigenschaften (<w:rPr>), um die direkte Formatierung festzulegen. Der Formatierungsbehl wird in einem leeren Tag (<w:b/>) angegeben.

```
<w:p>
  <w:r>
    <w:t>Dieses Wort ist</w:t>
  </w:r>
  <w:r><rPr><w:b/><w:rPr>
    <w:t>fett</w:t>
  </w:r>
  <w:r>
    <w:t>.</w:t>
  </w:r>
</w:p>
```

Benutzerdefiniertes XML

Zusätzlich zum XML-Dateiformat enthält Word 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; sie macht keinen benutzerfreundlichen XML-Editor aus Word.

Abbildg. 21.8 Die in Word geöffnete KML-Datei



Wenn Sie die Beispieldatei für Listing 21.2 (*Bsp21_02.xml* auf der Buch-CD) in Word öffnen, wird das Ergebnis in Word 2003 und Word 2007 ähnlich wie in Abbildung 21.8 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.

HINWEIS

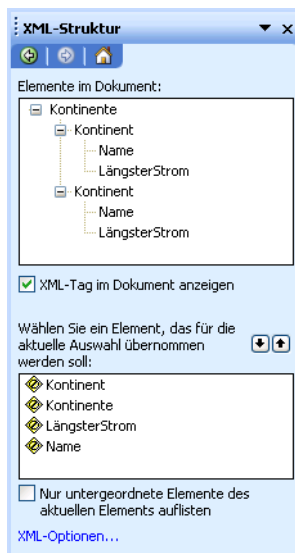
In Word 2010 werden die rosaroten Tags nicht angezeigt, da diese Funktionalität wegen des erwähnten Gerichtsentscheids nicht mehr vorhanden ist. Sie können jedoch in ein Dokument eingefügt und gespeichert werden, was wir für Abbildung 21.8 getan haben.

Klicken Sie auf die Schaltfläche *Struktur* in der Gruppe *Entwicklertools/XML* des Menübands.

Dieser Aufgabenbereich (Abbildung 21.9) ist in zwei Abschnitte unterteilt: Der obere zeigt eine hierarchische Auflistung der im Dokument *enthaltenen* Elemente sowie ein Kontrollkästchen, womit diese ein- und ausgeblendet werden können; der untere listet die *verfügbaren* Elemente auf. Falls das XML-Dokument mit einem Schema verknüpft ist, wird diese Information dem Schema entnommen, sonst werden (wie hier) nur die sich im Dokument befindenden angezeigt. Das dazugehörige Kontrollkästchen filtert die Liste, um nur kontextgültige Elemente anzubieten. (Da die Gültigkeit nur anhand eines Schemas geprüft werden kann, bleibt die gefilterte Liste leer, wenn kein Schema mit dem aktuellen Dokument verbunden ist.)

Da das abgebildete XML-Dokument mit keinem Schema verbunden ist, können überall im Text beliebig Elemente und Text eingefügt werden. Es kann als Word-Dokument (*.doc*), WordProcessingML-Datei oder als XML-Datei ohne Word-eigene Elemente (*Nur Daten speichern*) gespeichert werden.

Abbildg. 21.9 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 21.18 für das »KML«-Vokabular geladen und mit einem Dokument (im vorgestellten Szenario mit der Beispieldatei *Bsp21_02.xml*) verbunden. (Dieses Schema unterscheidet sich von den vorangehenden durch die Deklaration eines `targetNamespace` und die Unterstützung von gemischtem Inhalt (`mixed="true"`).)

Listing 21.18 Inhalt von *Bsp21_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.KAP21Beispiel.com/BSP18"
  elementFormDefault="qualified">
  <xsd:element name="Kontinente">
    <xsd:complexType mixed="true">
      <xsd:sequence>
        <xsd:element name="Kontinent" maxOccurs="unbounded" >
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="Name" type="xsd:string" minOccurs="1" />
              <xsd:element name="LängsterStrom" type="xsd:string" minOccurs="0" />
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Listing 21.18 Inhalt von *Bsp21_18.xsd* – Ein XML-Schema für das KML-Vokabular (Fortsetzung)

```
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:schema>
```

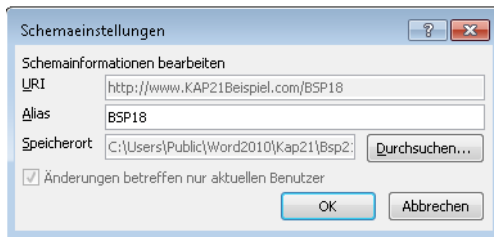
CD-ROM

Die Beispieldatei *Bsp21_18.xsd* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap21*.

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 *Bsp21_02.xml* aus Listing 21.2). Klicken Sie neben dem ersten Tag (Kontinente).
2. Blenden Sie das Dialogfeld *XML-Schema* über die Schaltfläche *Schema* auf der Registerkarte *Entwicklertools*.
3. Klicken Sie auf die Schaltfläche *Schema hinzufügen*, um ein Schema in die Liste aufzunehmen (Abbildung 21.10). Navigieren Sie zum Ordner und wählen Sie die *.xsd*-Datei aus.

Abbildg. 21.10 Das Dialogfeld *Schemaeinstellungen*



4. Bei der Aufforderung für ein »Alias« (Namensraumpräfix) geben Sie einen kurzen, beschreibenden Ausdruck ein.

HINWEIS

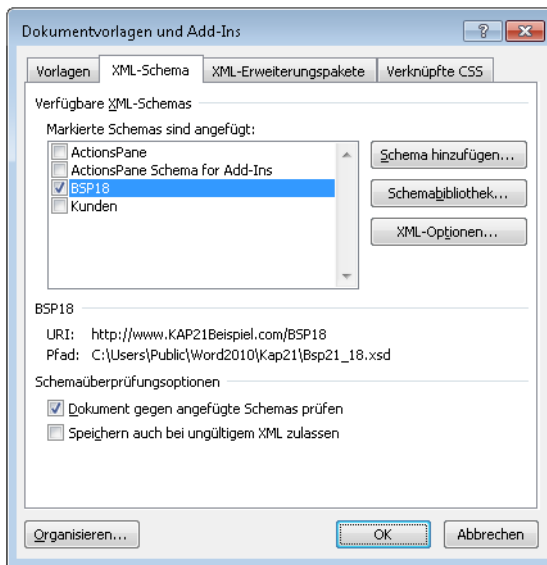
Word wird beim Speichern des XML-Dokuments das eingegebene Namensraumpräfix nicht berücksichtigen und wird ein eigenes generieren (wie beispielsweise *ns0*). Das von Ihnen eingegebene dient nur der Anzeige im Aufgabenbereich *XML-Struktur*.

5. Das Namensraumpräfix wird der Schemabibliothek (Abbildung 21.11) 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*.

HINWEIS

Die eigentliche Schemabibliothek, in der Schemas und Lösungen verwaltet werden, wird durch Anklicken der Schaltfläche *Schemabibliothek* erreicht (Abbildung 21.17). In der oberen Hälfte können Schemas hinzugefügt, entfernt sowie andere Namensraumpräfixe zugewiesen werden.

Abbildg. 21.11 Schemas werden auf der Registerkarte *XML-Schema* mit einem Dokument verbunden

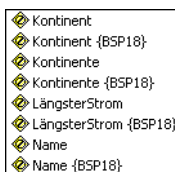


Beim Betrachten des Aufgabenbereichs *XML-Struktur* wäre zu erwarten, dass der Eintrag *Kontinent* in der unteren Liste steht, da laut Schema nach *Kontinente* das einzig erlaubte Element *Kontinent* ist. Dies ist jedoch nicht der Fall. Beim Deaktivieren des Kontrollkästchens *Nur untergeordnete Elemente des aktuellen Elements auflisten* wird der Grund klar, wie in Abbildung 21.12 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 *Bsp21_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.KAP21Beispiel.com/BSP18* in *Bsp18*).

Abbildg. 21.12 Liste der verfügbaren Elemente im Aufgabenbereich *XML-Struktur*



Es ist auch möglich, durch ein Makro wie in Listing 21.19 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 21.19 Den Namensraum *URI* in einem Dokument ersetzen

```
Sub NamensraumAnpassen()  
    Const strURI = "http://www.KAP21Beispiel.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
```

CD-ROM

Die Beispieldatei *Bsp21_19.doc* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap21*.

Wenn Sie nun das XML-Dokument unter einem anderen Namen speichern (beispielsweise *Bsp21_02a.xml*), schließen und dann im Windows-Editor öffnen, erkennen Sie, dass der Namensraum *URI http://www.KAP21Beispiel.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 *XML-Schema*-Dialogfeld bestätigt, dass Word das Schema mit dem XML-Dokument verbunden hat.

HINWEIS

In Word 2010 werden alle Informationen in der XML-Datei gespeichert. Beim Wiedereröffnen erscheinen jedoch die XML-Tags nicht. Die Verbindung zum Schema ist jedoch noch vorhanden.

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 21.13).

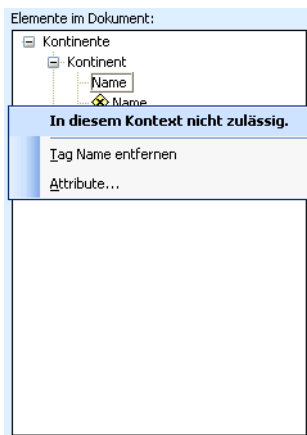
Standardmäßig kann ein ungültiges XML-Dokument nicht gespeichert werden. Wollen Sie es trotzdem speichern, muss entweder

- die Verbindung zum Schema getrennt werden,
- in Dialogfeld *XML-Schema* das Kontrollkästchen *Speichern auch bei ungültigem XML zulassen* aktiviert werden oder
- das XML-Dokument als Word-Dokument (*.doc*) gespeichert werden.

HINWEIS

Unter Umständen kann Word ein nicht wohlgeformtes XML-Dokument nicht öffnen. In diesem Fall müssen Sie die Fehler zuerst in einer anderen Umgebung (wie dem Windows-Editor) beheben.

Abbildg. 21.13 Kontextfehler, weil ein »Name«-Element nicht in einem »Name« verschachtelt sein darf

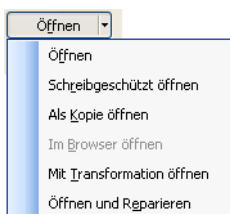


Transformationen und Lösungen (Solutions)

Beim Öffnen 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.

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 21.14). (Die Transformation eines WordProcessingML-Dokuments in ein anderes WordProcessingML-Dokument ist normalerweise recht komplex.)

Abbildg. 21.14 Ein XML-Dokument mit einer Transformation öffnen



Es ist auch möglich, das Aussehen eines XML-Dokuments zu ändern (es nochmals zu transformieren), 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 (Abbildung 21.15):

1. Öffnen Sie das XML-Dokument, das im Abschnitt »XML-Vokabulare« ab Seite 872 vorgestellt wurde (*Bsp21_02.xml*).
2. Im Aufgabenbereich *XML-Dokument* klicken Sie auf den Eintrag *Durchsuchen*.

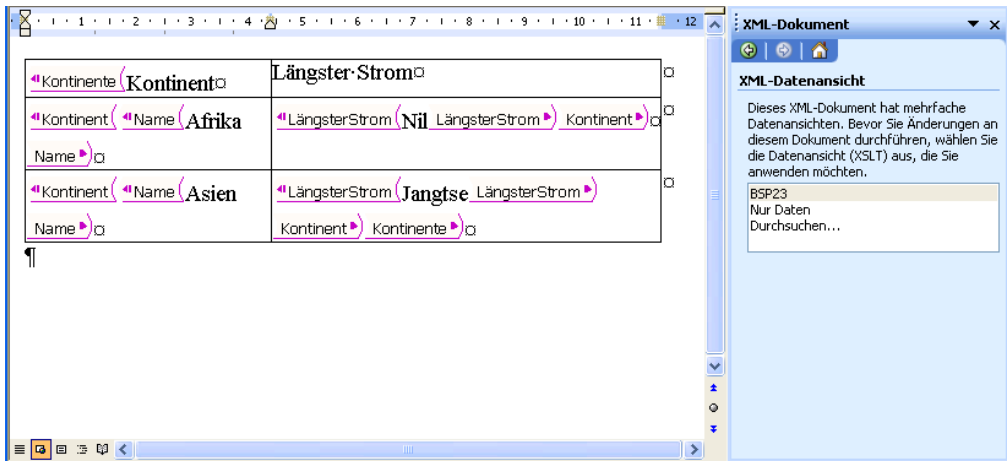
3. Navigieren Sie zum Ordner mit den Beispieldateien und wählen Sie eine Transformation (beispielsweise *Bsp21_14.xsl*), die im Abschnitt »XML-Daten transformieren« ab Seite 874 beschrieben wurde.

Sie sollten eine ähnliche Tabelle wie in Abbildung 21.1 sehen. Wenn Sie die gleiche Transformation mit dem im Abschnitt »Die Word-Schemabibliothek« ab Seite 903 erstellten Dokument (*Bsp21_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 21.20 veranschaulicht eine Transformation, die dieses XML-Dokument in ein WordProcessingML-Dokument transformiert, wie in Abbildung 21.15. Sie deklariert sowohl den Namensraum des *Bsp21_18.xsd*-Schemas als auch den für das Word-Vokabular:

```
xmlns:w="http://schemas.microsoft.com/office/word/2003/wordml"
xmlns:ns2="http://www.KAP21Beispiel.com/BSP18"
```

Abbildg. 21.15 Ein in WordProcessingML transformiertes XML-Dokument



Listing 21.20 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.KAP21Beispiel.com/BSP18">
  <xsl:output method="xml" encoding="UTF-8" standalone="yes" />
  <xsl:template match="/">
    <w:wordDocument
      xmlns:w="http://schemas.microsoft.com/office/word/2003/wordml"
      xmlns:ns2="http://www.KAP22Beispiel.com/BSP18"
      xml:space="preserve">
      <w:body>
        <xsl:apply-templates select="ns2:Kontinente" />
      </w:body>
    </w:wordDocument>
  </xsl:template>
```


Listing 21.20 Transformation, um aus einem KML- ein WordProcessingML-Dokument zu erzeugen (Fortsetzung)

```

<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>

<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>

```

CD-ROM Die Beispieldatei *Bsp21_23.xsl* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap21*.


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.KAP21Beispiel.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.

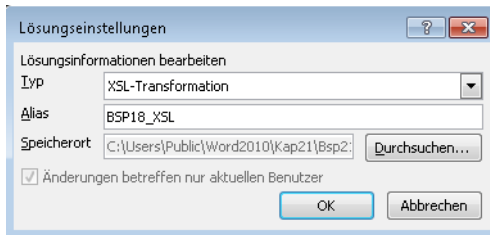
Eine Transformation darf mit mehr als einem Schema verbunden werden.

Fügen Sie die Transformation *Bsp21_23.xsl* wie folgt der Schemabibliothek hinzu:

1. Blenden Sie das Dialogfeld *Schemabibliothek* über die Menüfolge *Entwicklertools/XML/Schema* mit einem Klick auf die Schaltfläche *Schemabibliothek* ein.
2. Wählen Sie das Schema (*Bsp18*) in der Liste aus.
3. Klicken Sie auf die Schaltfläche *Solution hinzufügen*.

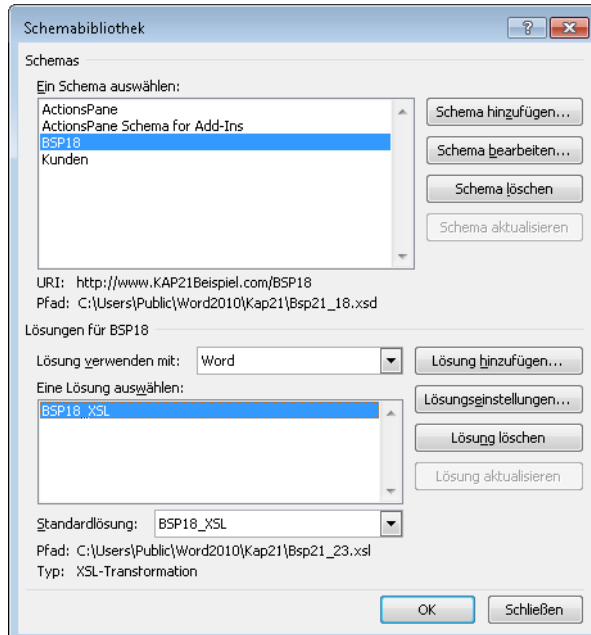
4. Navigieren Sie zum Beispiel-Ordner, wählen Sie die Transformation (*Bsp21_23.xsl*) und betätigen Sie dann *Öffnen*. Das Dialogfeld *Solution-Einstellungen* (Abbildung 21.16) wird eingeblendet.
5. Geben Sie ein Namensraumpräfix (»Alias«) in das *Alias*-Feld ein (falls nicht, erstellt Word eine GUID und benutzt diese).

Abbildg. 21.16 Beim Einbinden einer Transformation in die Schemabibliothek ein Namensraumpräfix zuweisen



Beachten Sie in Abbildung 21.17, 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. 21.17 Eine Solution (Transformation) mit einem Schema verbinden und verwalten im Dialogfeld *Schemabibliothek*



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), den 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 Beispielschema mit dem Namensraum »BSP18« in die Schemabibliothek:

```
Dim objSchema As Word.XMLNamespace
Set objSchema = Application.XMLNamespaces.Add(
    Path:="C:\Wordbuch\Beispiele\Kap21\Bsp21_18.xsd", NamespaceURI:="", _
    Alias:="BSP18", InstallForAllUsers:=False)
```

WordProcessingML außerhalb von Word generieren

Im Abschnitt »Transformationen und Lösungen (Solutions)« ab Seite 907 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 und sogar auf einem Server stattfinden.

Als Beispiel zeigen wir, wie die Transformation programmtechnisch mit dem DOM des MSXML-Parsers bewerkstelligt wird. Das Listing 21.21 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 *Bsp21_25.xml*. Sie müsste ebenso aussehen wie das Ergebnis der Transformation im Abschnitt »Transformationen und Lösungen (Solutions)« ab Seite 907.

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. 21.18 XML in WordProcessingML transformieren – ohne Word

```
Sub XML2WML()
    'Passen Sie die Pfadangaben Ihrem System an
    Const strPath = "C:\\Wordbuch\\Beispiele\\Kap21\\"
    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 & "\\Bsp21_23.xsl") Then
        MsgBox "XSL-Transformationsdatei konnte nicht gefunden werden"
    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 & "\\Bsp21_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 & "\\Bsp21_25.xml"
                End If
            End If
        End If
    End If

    Set objWordMLDocument = Nothing
    Set objXMLDocument = Nothing
    Set objXSLTransform = Nothing
End Sub
```

CD-ROM

Die Beispieldatei *Bsp21_25.doc* sowie *Bsp21_25.txt* mit dem Code finden Sie auf der CD-ROM zum Buch im Ordner *\\Beispiele\\Kap21*.

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 870 ff.) vorgestellt
- Eine Übersicht der XML-Funktionalität in Word 2003 folgte im Abschnitt »Welche Aufgabe erfüllt XML in Word?« (Seiten 876 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 877 ff.) kurz erläutert

HINWEIS

Alle in diesem Abschnitt vorgestellten Beispiele mit Word 2003 WordProcessingML funktionieren weiterhin in Word 2007 und Word 2010. Auch wenn die rosa-roten XML-Knotenpunkte beim Öffnen eines XML Dokuments nicht angezeigt werden (Abbildung 21.15), funktioniert eine Transformation.

- Die Grundzüge von WordProcessingML und Words eigenem XML-Vokabular wurden in Abschnitt »WordProcessingML« (Seiten 912 ff.) präsentiert
- Wie Schemas und Transformationen in Word integriert werden, wurde im Abschnitt »Die Word-Schemabibliothek« (Seiten 903 ff.) behandelt
- Schließlich wurde im Abschnitt »WordProcessingML außerhalb von Word generieren« (Seiten 912 ff.) veranschaulicht, wie ein WordProcessingML-Dokument unabhängig von der Word-Anwendung erstellt werden kann

Kapitel 22

Word Open XML- Dateiformat

In diesem Kapitel:

Die Word-Dateiformate	916
Erstellung eines Open XML-Dokuments	917
Das OPC und dessen Bestandteile	918
Teile, Beziehungen und Inhaltstypen	934
Die programmtechnische Arbeit mit Open XML: eine Übersicht	944
Open XML-Dokumente & XSLT	965
Zusammenfassung	968

In Office 2007 hat Microsoft neue Dateiformate für mehrere Office-Anwendungen eingeführt. Diese tragen die Bezeichnung *Office Open XML*, was oft auf *Open XML* oder *OOXML* abgekürzt wird. Das vorliegende Kapitel bietet einen Überblick zum neuen Dateiformat in Bezug auf Microsoft Word. Es baut dabei auf die Diskussion im Kapitel 21 auf. Falls Sie über keine XML-Vorkenntnisse verfügen, wäre es hilfreich, jenes Kapitel zuerst durchzulesen.

Die Word-Dateiformate

Bis einschließlich Word 2003 war die in Office 97 eingeführte binäre *.doc*-Datei das standardmäßige Dateiformat. Es stellt eine Momentaufnahme der von Word im Speicher gehaltenen Strukturen dar. Die Arbeit mit diesem Format gestaltet sich nicht einfach; Dritthersteller haben Schwierigkeiten, aus einer Word-Datei Informationen zu lesen, geschweige denn Word-Dokumente zu erstellen, die erfolgreich in Word geöffnet werden können.

HINWEIS

Lange Zeit war das binäre Dateiformat nicht öffentlich verfügbar. Der Entwickler musste bei Microsoft ein Gesuch stellen und ging gewisse Verpflichtungen ein. Seit Einführung von Office Open XML sind die Einzelheiten beider Formate offen zugänglich. Das binäre Format kann unter [http://msdn.microsoft.com/en-us/library/cc313105\(office.12\).aspx](http://msdn.microsoft.com/en-us/library/cc313105(office.12).aspx) heruntergeladen werden. Es gibt zudem ein Diskussionsforum auf MSDN unter http://social.msdn.microsoft.com/Forums/en-US/os_binaryfile/threads/.

Wie im Kapitel 21 erwähnt, gibt es zwei Hauptgründe für den Wechsel zu einem XML-Dateiformat: verbesserte Wiederverwendung des Dokumentinhalts sowie erhöhte Zugänglichkeit. Ein mit XML strukturiertes Dokument kann auf mehrere Arten wiederverwendet werden. Zudem ist es für Dritthersteller um einiges einfacher geworden, Word-Dokumente zu manipulieren bzw. herzustellen.

Die ersten Vorboten sahen wir in Office XP (2002) mit SpreadsheetML für Excel. In Office 2003 folgte WordProcessingML (Word 2003 XML) für Word. Obwohl beide Formate verbesserte Wiederverwendung sowie erhöhte Zugänglichkeit anboten, waren die XML-Vokabulare zu wenig umfangreich, um die alten Dateiformate zu ersetzen.

Microsoft wollte ein Dateiformat, welches

- das binäre als standardmäßiges Dateiformat ersetzt;
- die Konvertierung von Dokumenten im binären Format ohne Inhaltsverlust ermöglicht;
- robuster ist;
- von internationalen Standard-Organisationen anerkannt wird, beispielsweise ISO (Internationale Organisation für Normung).

Das Resultat ist das in Office 2007 eingeführte Office Open XML-Dateiformat.

Office Open XML – das standardmäßige Dateiformat

In Office 2003 hat Microsoft entschieden, die bisherigen, binären Dateiformate als die standardmäßigen beizubehalten. Ihre Zeit (die über eine Dekade dauerte) ist mit der Veröffentlichung von Office 2007 zu Ende gegangen. Seither haben drei weitere Änderungen diesen Wechsel gefestigt:

- Die Standardisierung des Dateiformats durch ISO in 2008 (siehe auch den Kasten »Standardisierung der Dateiformate« auf Seite 925)
- Missbilligung des RTF-Dateiformats (bislang das für den Mensch lesbare Dateiformat für Word-Dokumente). Dieses Dateiformat wird nicht länger unterhalten oder ausgebaut.
- Einführung einer neuen, COM-basierten Konversionsschnittstelle für Textdateien (Word 2007 SP2). Diese benutzt Open XML statt RTF als das Zwischenformat und entfernt somit die letzte Abhängigkeit vom älteren Format.

Diese Open XML-Dateiformate sind die standardmäßigen Dateiformate für Office 2007, Mac Office 2008 und Office 2010. Zudem bietet Microsoft einen kostenlosen Satz Konvertierfilter zum Herunterladen an. Diese ermöglichen Benutzern von Word 2000, 2002 (XP), 2003, Mac Word v.X und Mac Word 2004 die Arbeit mit Open XML-Dokumenten. Damit können sie das neue Dateiformat öffnen sowie binäre Dateien im neuen Dateiformat speichern.

Erstellung eines Open XML-Dokuments

Betrachten wir zuerst, wie ein Open XML Word-Dokument aufgebaut ist. Wir erstellen ein solches von Grund auf im Texteditor. Der erste Versuch ist eine Variation von Listing 21.17 aus Kapitel 21 (ohne Tabelle). Das Resultat aus Listing 22.1 ist ein leeres Dokument mit einer einzigen Absatzmarke. Es handelt sich hier um Word 2003 XML.

Listing 22.1 Inhalt von *Bsp22_01.xml* – Word 2003-WordProcessingML, das ein leeres Dokument definiert

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<w:wordDocument
  xmlns:w="http://schemas.microsoft.com/office/word/2003/wordml">
  <w:body>
    <w:p/>
  </w:body>
</w:wordDocument>
```

Geben Sie diese Zeilen in einen Texteditor ein und speichern die Datei als *Bsp22_01.xml* ab. Das Resultat lässt sich in Word 2003, 2007 und 2010 als Word-Dokument öffnen. Das Dokument ist »leer«.

WICHTIG

Denken Sie daran, dass die XML-Sprache auf die Großschreibung genau achtet!

CD-ROM

Die Beispieldatei *Bsp22_01.xml* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap22*.

Das neuere Open XML hat mit seinem Vorgänger einige Gemeinsamkeiten. Viele der Elemente und Attribute bleiben gleich, wie beispielsweise `<w:p>` für einen Absatz (Paragraph), andere wurden jedoch geändert. Open XML benutzt beispielsweise `<w:document>` statt `<w:wordDocument>`.

Da Open XML ein neues XML-Vokabular eingeführt hat, benötigt es auch einen anderen XML-Namensraum URI.

Probieren wir also zunächst, diese Änderungen im vorherigen Beispiel vorzunehmen (Listing 22.2). Wir speichern die Datei als *Bsp22_02.xml* und beobachten, wie sich das Dokument verhält.

Listing 22.2 Inhalt von *Bsp22_02.xml* – Ein leeres Dokument im WordProcessingML von Word 2003

```
<?xml version="1.0" encoding="UTF-8"?>
<w:document
  xmlns:w="http://schemas.openxmlformats.org/wordprocessingml/2006/main">
  <w:body>
    <w:p/>
  </w:body>
</w:document>
```

Wird die Datei in Word 2007 oder 2010 geöffnet, erscheint eine Fehlermeldung: der Dateiinhalt wird nicht korrekt erkannt (Abbildung 22.1). Ob Word 2003 die Datei öffnen kann, kommt darauf an, ob das Kompatibilitätspack installiert ist. Bestenfalls öffnet Word die Datei als reine XML-Datei und zeigt die Elemente als Tags eines benutzerdefinierten Namensraums gehörend an.

Abbildg. 22.1 Ein mit dem Namensraum für Open XML erstelltes XML-Dokument verursacht eine Fehlermeldung



Was fehlt?

Ein Open XML-Dokument besteht aus mehreren, in einem Paket (»Package«) zusammengefassten XML-Dateien (»Teilen«) statt einer einzelnen XML-Datei. Die Strukturen dieses Pakets müssen den Richtlinien und Regeln der *Open Packaging Conventions* (OPC) entsprechen.

CD-ROM

Die Beispieldatei *Bsp22_02.xml* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap22*.

Das OPC und dessen Bestandteile

Ein OPC-Paket enthält mehrere Teile (Parts). Einer dieser Teile enthält das Dokument-XML. Weitere Teile stellen Informationen über den Inhalt des Pakets oder wie die einzelnen Teile miteinander zusammenhängen bereit. Das XML für ein solches Paket ist in Listing 22.3 ersichtlich. Wenn Sie die-

ses Listing in einen Texteditor eingeben, die Datei als *Bsp22_03.xml* speichern und anschließend in Word 2007 oder 2010 öffnen, liegt ein »leeres« Dokument vor.

Listing 22.3 Inhalt von *Bsp22_03.xml* – Office Open XML für ein leeres Word 2007/2010-Dokument

```
<?xml version="1.0" encoding="UTF-8"?>
<pkg:package
  xmlns:pkg="http://schemas.microsoft.com/office/2006/xmlPackage">

  <pkg:part
    pkg:name="/rels/.rels"
    pkg:contentType="application/vnd.openxmlformats-package.relationships+xml">
    <pkg:xmlData>
      <Relationships
        xmlns="http://schemas.openxmlformats.org/package/2006/relationships">
        <Relationship
          Id="rId1"
          Type=
"http://schemas.openxmlformats.org/officeDocument/2006/relationships/officeDocument"
          Target="bsp22/03.xml"/>
        </Relationships>
      </pkg:xmlData>
    </pkg:part>

    <pkg:part
      pkg:name="/bsp22/03.xml"
      pkg:contentType="application/vnd.openxmlformats-officedocument.wordprocessingml.document.main+xml">
      <pkg:xmlData>
        <w:document
          xmlns:w="http://schemas.openxmlformats.org/wordprocessingml/2006/main">
          <w:body>
            <w:p/>
          </w:body>
        </w:document>
      </pkg:xmlData>
    </pkg:part>
  </pkg:package>
```

CD-ROM

Die Beispieldatei *Bsp22_03.xml* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap22*.

Das »flache« OPC-Format

Nehmen wir dieses Paket – das im so genannten *Flat OPC-Format* erfasst ist – etwas genauer unter die Lupe.

Dieses Paket besteht aus zwei Teilen (Parts). Jeder Teil hat einen Namen (Name), der wie eine Pfadangabe aussieht. Der zweite enthält das uns aus Listing 22.2 bekannte XML für ein Word-Dokument. Bezeichnen wir ihn als den Teil »Hauptdokument«. Der erste Teil definiert eine Beziehung (Relationship) zum zweiten Teil und hat einen Typ (Type) sowie ein Ziel (Target).

Der Typ der Beziehung ist `http://schemas.openxmlformats.org/officeDocument/2006/relationships/officeDocument` und ihr Ziel hat den gleichen Namen wie der Teil Hauptdokument (ohne das vorangehende »/«). Diese Tatsachen lassen darauf schließen, dass der erste Teil, die Beziehung, darauf hinweist, dass das Objekt `officeDocument` sich im Teil `bsp22/03.xml` (also im Teil Hauptdokument) befindet.

Bislang haben wir nichts gesehen, was darauf hindeutet, welche Art Dokument vorliegt. Die beschriebenen Informationen sind in allen Open XML-Dokumenten vorhanden. Was unterscheidet dieses von einem Excel- oder PowerPoint-Dokument? Darauf gibt es einige Hinweise:

- Der Teil Hauptdokument deklariert und setzt den Namensraum

```
xmlns w="http://schemas.openxmlformats.org/wordprocessingml/2006/main"
```

- Zudem enthält jeder Teil ein Inhaltstyp (`contentType`). Für das zur Diskussion stehende Hauptdokument handelt es sich um

```
application/vnd.openxmlformats-officedocument.wordprocessingml.document.main+xml
```

Diese URI legt fest, dass wir es mit einem Open XML Word-Dokument ohne Makros (`.docx`) zu tun haben. Dokumente mit Makros (`.docm`), sowie Vorlagen mit (`.dotm`) und ohne (`.dotx`) Makros haben eigene URIs für den Inhaltstyp.

Ein Open XML-Dokument umfasst meist mehr als nur zwei Teile. Fügen wir dem einfachen Dokument eine Kopfzeile zu und schauen, was passiert.

Im Word 2003 XML wird die Informationen für die Kopfzeile innerhalb des Elements `wordDocument` verschachtelt. Im Open XML-Paket befindet sie sich in einem eigenen Teil, wie dies Listing 22.4 veranschaulicht.

Listing 22.4 Inhalt von `Bsp22_04.xml` – Ein Word 2007/2010 Open XML-Dokument mit einer Kopfzeile

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<pkg:package
  xmlns:pkg="http://schemas.microsoft.com/office/2006/xmlPackage">
  <pkg:part
    pkg:name="/rels/.rels"
    pkg:contentType="application/vnd.openxmlformats-package.relationships+xml">
    <pkg:xmlData>
      <Relationships
        xmlns="http://schemas.openxmlformats.org/package/2006/relationships">
        <Relationship Id="rId1"
          Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/officeDocument"
          Target="word/document.xml" />
        </Relationships>
      </pkg:xmlData>
    </pkg:part>
    <pkg:part
      pkg:name="/word/document.xml"
      pkg:contentType="application/vnd.openxmlformats-officedocument.wordprocessingml.document.main+xml">
      <pkg:xmlData>
        <w:document
          xmlns:r="http://schemas.openxmlformats.org/officeDocument/2006/relationships"
```

Listing 22.4 Inhalt von *Bsp22_04.xml* – Ein Word 2007/2010 Open XML-Dokument mit einer Kopfzeile (Fortsetzung)

```

    xmlns:w="http://schemas.openxmlformats.org/wordprocessingml/2006/main">
    <w:body>
      <w:p />
      <w:sectPr>
        <w:headerReference w:type="default" r:id="rId1" />
      </w:sectPr>
    </w:body>
  </w:document>
</pkg:xmlData>
</pkg:part>
<pkg:part
  pkg:name="/word/_rels/document.xml.rels"
  pkg:contentType="application/vnd.openxmlformats-package.relationships+xml">
  <pkg:xmlData>
    <Relationships
      xmlns="http://schemas.openxmlformats.org/package/2006/relationships">
      <Relationship Id="rId1"
        Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/header"
        Target="header1.xml" />
      </Relationships>
    </pkg:xmlData>
  </pkg:part>
<pkg:part
  pkg:name="/word/header1.xml"
  pkg:contentType="application/vnd.openxmlformats-officedocument.wordprocessingml.header+xml">
  <pkg:xmlData>
    <w:hdr xmlns:w="http://schemas.openxmlformats.org/wordprocessingml/2006/main">
      <w:p>
        <w:r>
          <w:t>Kopfzeilentext</w:t>
        </w:r>
      </w:p>
    </w:hdr>
  </pkg:xmlData>
</pkg:part>
</pkg:package>

```

Dieses neue Paket hat vier statt zwei Teile:

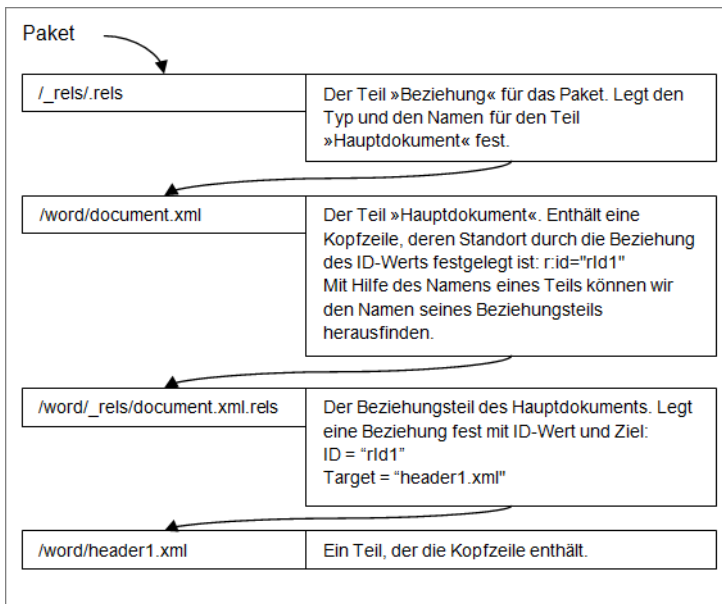
- Der erste Teil, /_rels/.rels, zeigt weiterhin auf den Teil Hauptdokument
- Der zweite Teil ist immer noch das Hauptdokument, mit einigen Änderungen gegenüber zuvor:
 - Sein Name ist nun /word/document.xml
 - Das Element <w:document> enthält einen Namensraum, der das Namensraumpräfix r definiert
 - Ein neues Element <w:sectPr> wurde zugefügt mit einem verschachtelten Element, das auf die Kopfzeile weist:

```
<w:headerReference w:type="default" r:id="rId1" />
```

- Der dritte Teil namens `/word/_rels/document.xml.rels` ist neu und definiert eine Beziehung `rId1` mit dem Ziel `header1.xml`
- Der vierte Teil `/word/header1.xml` ist ebenfalls neu und enthält das Element `<w:hdr>` mit dem XML für einen einzigen Absatz mit Text für die Kopfzeile

Das ergibt die in Abbildung 22.2 ersichtliche Verkettung von Teilen und Beziehungen.

Abbildg. 22.2 Die Struktur eines Pakets mit einem Hauptdokument- sowie Kopfzeileteilen

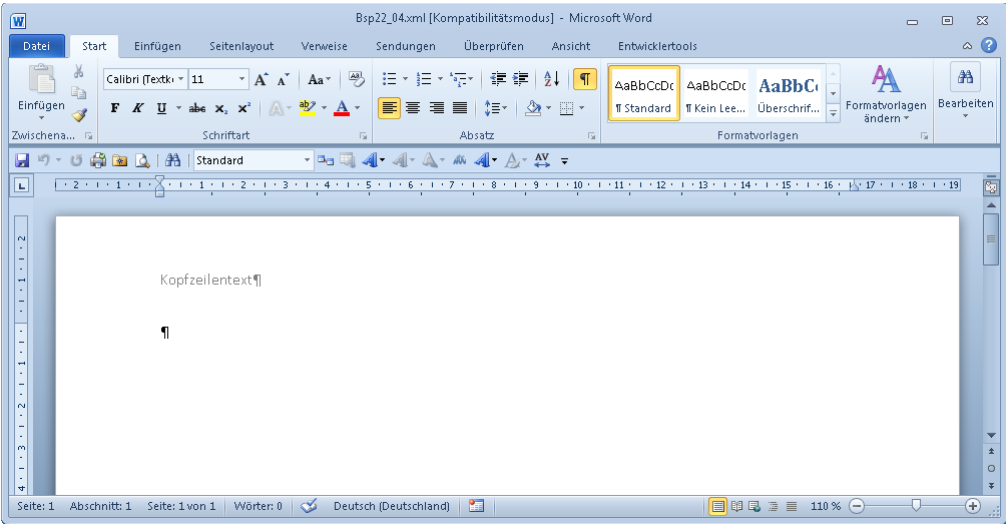


HINWEIS

Falls Sie sich fragen, ob das mehrmalige Vorkommen des ID-Werts `rId1` innerhalb des Pakets zu Problemen führt, können wir Sie beruhigen. Paket-bezogene Beziehungen befinden sich immer im Beziehungsteil für das Paket, und solche für das Dokument im entsprechenden Teil für das Dokument.

Wenn Sie dieses Beispiel im Texteditor eingeben, speichern und in Word 2007 oder Word 2010 öffnen – oder die Beispieldatei von der CD-ROM zum Buch verwenden –, sehen sie ein Dokument mit dem Text »Kopfzeilentext« in der Kopfzeile (Abbildung 22.3).

Abbildg. 22.3 Das Open XML-Dokument mit einer Kopfzeile



CD-ROM Die Beispieldatei *Bsp22_04.xml* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap22*.

Das vollumfängliche OPC-Format

Bis hierher waren die Beispiele im flachen OPC XML-Format. Alle Informationen befinden sich in einer einzigen Datei; ein Element mit seinen Unterelementen definieren das Paket und seine Teile. Diese Sichtweise entspricht mehr oder weniger unserem gewöhnten »Alltag« und ist überschaubar, was den Einstieg in das Konzept des »echten« Open XML-Formats vereinfacht.

Laut der Open Package Convention besteht eine Datei im Open XML-Format richtigerweise aus einem Ordner, der als Paket dient, mit einer Datei für jeden Teil. Dieser Ordner muss dann in eine ZIP-Datei konvertiert und die Dateiendung vorzugsweise in (wie im Falls dieses Beispiels) *.docx* umbenannt werden.

Allzu weit davon entfernt sind wir mit dem letzten Beispiel nicht. Durch die Namen der Teile in Abbildung 22.2 zwingt sich die Struktur in Abbildung 22.4 förmlich auf.

Abbildg. 22.4 Der Inhalt eines Open XML-Pakets, ausgelegt als Ordner und Dateien

Bsp22_04	(Ordner)
rels	(Ordner)
rels	(Datei)
word	(Ordner)
rels	(Ordner)
document.xml.rels	(Datei)
document.xml	(Datei)
header1.xml	(Datei)

Sie können eine solche ZIP-Datei von Hand erstellen. Da aber kaum jemand ernsthaft ein Open XML-Dokument auf diese Art erstellen wird, werden wir diese Struktur am Beispiel einer *docx*-Datei lediglich bestätigen.

1. Kopieren Sie die Datei *Kap22_04.docx* von der Buch-CD.
2. Führen Sie die Schritte 1 bis 4 im Kapitel 16, Abschnitt »Die Menüband-Erweiterung in das Dokument einbinden« aus, um das Dokument in eine ZIP-Datei umzuwandeln. (Ersetzen Sie dabei in der Anleitung den Dateinamen *customUI* durch *Bsp22_04.docx*.)
3. Sie können die darin enthaltenen *.xml*- und *.rels*-Dateien im Texteditor öffnen und ihren Inhalt mit dem entsprechenden Teil in Listing 22.4 vergleichen. In *\word\header1.xml*, beispielsweise, sollen Sie den folgenden XML-Code sehen:

```
<?xml version="1.0" encoding="UTF-8"?>
<w:hdr xmlns:w="http://schemas.openxmlformats.org/wordprocessingml/2006/main">
  <w:p>
    <w:r>
      <w:t>Kopfzeilentext</w:t>
    </w:r>
  </w:p>
</w:hdr>
```

Mit Ausnahme der XML-Deklaration in der ersten Zeile handelt es sich hier um das gleiche XML, das sich im Element *<pkgData>* von Listing 22.4 befindet.

4. Letztlich fällt eine zusätzliche Datei im Ordner der obersten Ebene auf: *[Content_Types].xml*. Diese enthält die Informationen des Inhaltstyps (*contentType*) aus jedem Teil des flachen OPC-XML-Pakets, mit einem zusätzlichen Eintrag, wie dies in Listing 22.5 ersichtlich ist.

Listing 22.5 Inhalt von *[Content_Types].xml* – Die Datei mit den Inhaltstypen im ZIP-Ordner *Bsp22_04*

```
<?xml version="1.0"?>
<Types
  xmlns="http://schemas.openxmlformats.org/package/2006/content-types">
  <Default
    Extension="rels"
    ContentType="application/vnd.openxmlformats-package.relationships+xml"/>
  <Default
    Extension="xml"
    ContentType="application/xml"/>
  <Override
    PartName="/word/document.xml"
    ContentType="application/vnd.openxmlformats-officedocument.wordprocessingml.document.main+xml"/>
  <Override
    PartName="/word/header1.xml"
    ContentType="application/vnd.openxmlformats-officedocument.wordprocessingml.header+xml">
  </Types>
```

CD-ROM Der Beispielfolder *Bsp22_04* sowie die Dateien *Bsp22_04.zip* und *Bsp22_04.docx* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap22*. Die Beispieldatei *[Content_Types].xml* befindet sich in diesem Beispielfolder.

Somit liegt ein komplettes Open XML Word-Dokument vor. Es kann in Word 2007 oder Word 2010 und Word für Macintosh 2008 geöffnet werden sowie in jeder weiteren Version von Word mit installiertem Konvertierungspack.

Standardisierung der Dateiformate

Die *Open Packaging Conventions* (OPC) waren ursprünglich Teil der Office Open XML-Dateiformate, die im Jahr 2006 durch die ECMA als ECMA-376 standardisiert wurden. (Dieser Standard ist ebenso als ECMA-376 1st edition (2006) bekannt.)

Eine neue Version des Standards wurde in 2008 durch die ISO als ISO/IEC-29500:2008 abgesegnet. Eine aktualisierte Version des ECMA-Standards, die technologisch auf dem ISO-Standard ausgerichtet wurde, wurde als ECMA-376 2nd edition (2008) herausgegeben.

Die vier Hauptdokumente des ISO-Standards umfassen um die 7000 Seiten. Der Standard definiert »Übergangs«-Bestimmungen, sodass Dokumente, die sich nach dem früheren Standard richten, als gültige »Übergangs«-Dokumente des ISO-Standards gelten. Vorgesehen ist, dass die Übergangsbestimmungen später entfernt werden.

Die Office 2007- sowie Mac 2008-Anwendungen (Word, Excel und PowerPoint) erstellen und erkennen Dokumente, die dem Standard »ECMA-376 1st edition (2006)« entsprechen.

Die Dokumente der Office 2010-Anwendungen sind eher auf der Linie des ISO-Standards.

Die ISO/IEC 29500:2008-Standarddokumente umfassen vier Hauptdokumente mit Anhängen, wie folgt:

- Part 1 (Fundamentals and Markup Language Reference)
Enthält Definitionen der Übereinstimmungen (Conformance Definitions) sowie Referenzmaterial, u.a. XML-Schemas für die XML-Dokument Markup Sprachen. Dieser Teil umfasst mehr als 5.500 Seiten.
- Part 2 (Open Packaging Convention)
(Wie in diesem Abschnitt vorgestellt.) Beschreibt das OPC, Kern-Eigenschaften und andere paketbezogene Punkte, wie die Handhabung digitaler Signaturen und Miniaturen. Es stellt die XML-Schemas für die OPC bereit. Dieser Teil umfasst mehr als 1000 Seiten.
- Part 3 (Markup Compatibility and Extensibility)
Beschreibt die Erweiterungsmöglichkeiten und umfasst ungefähr 40 Seiten
- Part 4 (Transitional Migration Features)

Beschreibt Teile aus früheren Versionen, die aus Gründen der Rückwärtskompatibilität weiter unterstützt werden, wie VML (Grafiken). Es listet die Unterschiede zwischen »ISO/IEC 29500:2008« und »ECMA-376 1st edition« auf.

Diese Hauptdokumente für das ISO/IEC 29500:2008 Standard stehen unter <http://standards.iso.org/ittf/PubliclyAvailableStandards/index.html> zur Verfügung.

Verwandte Standards inklusive einige Korrekturen für 2010 finden Sie unter http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=54999. Diese sind jedoch nicht kostenlos.

Microsoft stellt Informationen zu den Office Open XML-Formaten unter [http://msdn.microsoft.com/en-gb/library/cc313118\(office.12\).aspx](http://msdn.microsoft.com/en-gb/library/cc313118(office.12).aspx) bereit.

Vom besonderen Interesse ist das Microsoft-Dokument, das seine Erfüllung der Standards beschreibt: [MS-OI29500]: Office Implementation Information for ISO/IEC 29500 Standard Compliance. Dieses Dokument können Sie unter [http://msdn.microsoft.com/en-us/library/ee908652\(office.12\).aspx](http://msdn.microsoft.com/en-us/library/ee908652(office.12).aspx) lesen.

Wo ist der passende Einstiegspunkt, um bei so vielen Informationen die Übersicht nicht zu verlieren? Wir empfehlen, Section 8 – Overview (Abschnitt 8 – Übersicht) der ISO/IEC 29500-1 Fundamentals and Markup Language Reference als Einstieg.

Anschließend ist es sinnvoll, über das Inhaltsverzeichnis für die vier Hauptdokumente zu schweifen und Ausschau auf die Themen, die relevant für die gegenwärtig zu bewältigende Aufgabe sind, zu halten.

Obwohl diese Dokumentation zuerst überwältigend erscheint, enthält es eine überproportionale Menge an Erklärungstext und hilfreiche Beispiele. Sind Sie hauptsächlich an Word interessiert, reduziert sich der Fokus auf die Word-relevanten Teile.

Beispiel: Einen *customUI*-Teil in ein Dokument einbinden

Nun werden die bis hierher gewonnen Kenntnisse anhand eines einfachen Codebeispiels programmtechnisch umgesetzt. Im Kapitel 16 haben Sie erfahren, wie Schritt für Schritt eine Anpassung des Menübands durch Integration eines Menüband-XML-Teils im Dokumentpaket erfolgt. In diesem Abschnitt zeigen wir auf, wie dies programmtechnisch mit VBA vollzogen wird.

Das vorliegende Makro ist einfach und versucht nicht, alle möglichen Szenarien abzuhandeln. Es nimmt beispielsweise an, dass das Zieldokument weder einen *customUI.xml*-Teil noch eine entsprechende Beziehung enthält.

Was macht das Beispiel?

Der VBA-Code dieses Beispiels führt folgende Handlungen aus, die den Schritten des Kapitels 16 entsprechen.

- packt die Datei *Bsp22_05rb.dotm* in einen Arbeitsordner aus,
- fügt die benötigten Beziehungen in den entsprechenden Teil für das Paket ein,
- kopiert *Bsp22_05ui.xml* in den korrekten Ordner der OPC-Struktur und nennt sie in *customUI.xml* um,
- packt den Ordner wieder in eine ZIP-Datei und ersetzt die ursprüngliche *.dotm*-Datei damit.

Diese Aufgaben werden ausgeführt, ohne das Word-Objektmodell zu bemühen; d.h. sie können genauso gut in Excel oder in einem klassischen VB-Projekt laufen. Die zwei Prozeduren, welche die Office Open XML-Datei aus- und einpackt, werden in einem späteren Beispiel nochmals verwendet.

HINWEIS

Eine ausführlichere Diskussion über die programmtechnische Arbeit mit dem Open XML-Format finden Sie weiter unten in diesem Kapitel.

Das Beispiel ausführen

CD-ROM Die Beispieldateien *Bsp22_05.docm*, *Bsp22_05rb.dotm* und *Bsp22_05ui.xml* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap22*.

1. Kopieren Sie alle vier Beispieldateien in einen Ordner auf Ihrer Festplatte mit der Pfadangabe *C:\Beispiele\Kap22*.
2. Öffnen Sie in Word die Datei *Bsp22_05.docm*.
3. Öffnen Sie den VB-Editor (**Alt** + **F11**) und blenden Sie das Modul *Bsp22_05* ein.
4. Passen Sie – falls notwendig – die Pfadangaben an Ihr System an.
5. Führen Sie die Hauptprozedur aus.

Wurde die Prozedur erfolgreich ausgeführt, können Sie *Bsp22_05rb.dotm* in Word 2007 oder Word 2010 öffnen und die neue Registerkarte im Menüband sehen.

Es folgt eine Diskussion des Makro-Codes für die Bearbeitung des Open XML. Die Prozeduren der Hilfsmodule werden nicht behandelt.

Das Hauptmodul *Bsp22_05*

Am Anfang des Moduls *Bsp22_05* sind **Konstanten** für folgende Informationen definiert (Listing 22.6):

- Pfadangaben und Dateinamen der Quelldateien sowie des Arbeitsordners. Diese können Ihren Bedürfnissen entsprechend angepasst werden.
- Die URI des Beziehungstyps für den Teil des Typs *customUI*
- Datiname und Pfadangabe des Teils *customUI* innerhalb des Pakets
- Eine ID und die URI des Namensraums für den Beziehungsteil XML
- Pfadangabe und Datiname für den Teil Beziehungen des Pakets

Listing 22.6 Konstanten in *Bsp22_05.docm*

```
Option Explicit

Const sCustomUIFullName As String =
    "c:\Beispiele\Kap22\Bsp22_05ui.xml"
Const sTargetDotmFullName As String =
    "c:\Beispiele\Kap22\Bsp22_05rb.dotm"
Const sWorkPath As String =
    "c:\Beispiele\Kap22\Bsp22_05"

Const CUSTOMUIREL As String =
    "http://schemas.microsoft.com/office/2006/relationships/ui/extensibility"
Const CUSTOMUIPATH As String =
    "/customUI/customUI.xml"
Const CUSTOMUIRELID As String =
    "customUIRelId"
Const RELSNS As String =
    "http://schemas.openxmlformats.org/package/2006/relationships"
Const RELPARTPATH As String = "_rels"
Const RELPARTEXTENSION As String = ".rels"
```

HINWEIS

Bsp22_05ui.xml verweist auf die Schemas von Office 2007 und funktioniert somit für Word 2007 sowie für Word 2010. Falls Sie eine Datei erstellen möchten, welche die Menüband-Erweiterungen von Word 2010 nutzen soll, dann müssen die folgenden Anpassungen zum in Listing 22.6 enthaltenen Code vorgenommen werden:

```
Const sCustomUIFullName As String =
    "c:\Beispiele\Kap22\Bsp22_05ui14.xml"
Const CUSTOMUIREL As String =
    "http://schemas.microsoft.com/office/2007/relationships/ui/extensibility"
Const CUSTOMUIPATH As String = _
    "/customUI/customUI14.xml"
Const CUSTOMUIRELID As String = _
    "customUIRelId14"
```

Die Hauptprozedur ist `addCustomUI` (Listing 22.7). Sie führt folgende Handlungen aus:

- Stellt sicher, dass die zwei Quelldateien vorliegen
- Erstellt, wenn nötig, den Arbeitsordner
- Packt das *.dotm* in den Arbeitsordner aus
- Öffnet die Datei mit den Beziehungen für das Paket in einem `MSXML2.DOMDocument60` Objekt im Speicher
- Fügt eine Beziehung für den Teil *customUI.xml* hinzu
- Kopiert diesen Teil in das Paket
- Packt den Ordnerinhalt wieder in eine ZIP-Datei und ersetzt die ursprüngliche *.dotm*

Der Arbeitsordner wird *nicht* entfernt, um Ihnen Gelegenheit zu geben, den Inhalt zu inspizieren.

TIPP

Es gibt verschiedene Möglichkeiten, über MSXML einem XML-Dokument im Speicher XML-Elemente und -Attribute hinzuzufügen. Etwas fremd für den Entwickler, der hauptsächlich mit Office-Objektmodellen arbeitet, ist die Erstellung eines neuen Knotenpunkts sowie das Hinzufügen dieses Elements in das XML-Dokument. Zwei Schritte sind erforderlich:

```
Set objXMLENode =
    objXMLDoc.createElement(tagDOMNodeType.NODE_ELEMENT, "Relationship", RELSNS)
...
objXMLDoc.DocumentElement.appendChild objXMLENode
```

Zuerst wird der Knotenpunkt erstellt. Danach werden seine Eigenschaften festgelegt. Erst dann wird der Knotenpunkt dem XML-Dokument angefügt.

Bei Office-Objektmodellen sind wir gewohnt, dass ein Objekt zuerst in das Dokument eingefügt und nachträglich bearbeitet wird. Bei der Arbeit mit MSXML ist es umgekehrt.

Listing 22.7 Bsp22_05 – *addCustomUI*

```
Sub addCustomUI()

    Dim bContinue As Boolean
    Dim objFSO As Scripting.FileSystemObject
```

Listing 22.7 Bsp22_05 – addCustomUI (Fortsetzung)

```

Dim objXMLANode As MSXML2.IXMLDOMNode
Dim objXMLENode As MSXML2.IXMLDOMNode
Dim objXMLDoc As MSXML2.DOMDocument60
Dim strPackageReIsFullName As String

bContinue = True
Set objFSO = New FileSystemObject
If Not objFSO.FileExists(sCustomUIFullName) Then
    bContinue = logError("Die Datei mit dem XML für das Menüband konnte " & _
        "nicht gefunden oder nicht geöffnet werden.")
End If

If bContinue Then
    If Not objFSO.FileExists(sTargetDotmFullName) Then
        bContinue = logError("Das Zieldokument konnte nicht gefunden oder " & _
            "nicht geöffnet werden.")
    End If
End If

If bContinue Then
    If Not objFSO.FolderExists(sWorkPath) Then
        objFSO.CreateFolder sWorkPath
    End If
    If Not objFSO.FolderExists(sWorkPath) Then
        bContinue = logError("Der Arbeitspfad konnte nicht gefunden oder nicht " & _
            "erstellt werden.")
    End If
End If

If bContinue Then
    bContinue = logPack(unpack(sTargetDotmFullName, sWorkPath))
End If

If bContinue Then
    strPackageReIsFullName = _
        objFSO.BuildPath(sWorkPath, _
            objFSO.BuildPath(RELPARTPATH, RELPARTEXTENSION))
    If objFSO.FileExists(strPackageReIsFullName) Then
        Set objXMLDoc = New MSXML2.DOMDocument60
        objXMLDoc.validateOnParse = True
        If objXMLDoc.Load(strPackageReIsFullName) Then
            Set objXMLENode = _
                objXMLDoc.createElement(tagDOMNodeName.NODE_ELEMENT, "Relationship", RELSNS)
            With objXMLENode
                Set objXMLANode = _
                    objXMLDoc.createElement(tagDOMNodeName.NODE_ATTRIBUTE, "Type", "")
                objXMLANode.NodeValue = CUSTOMUIREL
                .Attributes.setNamedItem objXMLANode
                Set objXMLANode = _
                    objXMLDoc.createElement(tagDOMNodeName.NODE_ATTRIBUTE, "Target", "")
                objXMLANode.NodeValue = CUSTOMUIPATH
                .Attributes.setNamedItem objXMLANode
                Set objXMLANode = _
                    objXMLDoc.createElement(tagDOMNodeName.NODE_ATTRIBUTE, "Id", "")
                objXMLANode.NodeValue = "customUIRelID"
            End With
        End If
    End If
End If

```

Listing 22.7 Bsp22_05 – *addCustomUI* (Fortsetzung)

```

        .Attributes.setNamedItem objXMLANode
        Set objXMLANode = Nothing
    End With
    objXMLDoc.DocumentElement.appendChild objXMLENode
    Set objXMLENode = Nothing
    objXMLDoc.Save strPackageRelsFullName
Else
    bContinue = logError("Der Beziehungsteil des Pakets konnte nicht geöffnet werden.")
End If
Set objXMLDoc = Nothing
Else
    bContinue = logError("Der Beziehungsteil des Pakets konnte nicht gefunden werden.")
End If
createPath
objFSO.GetParentFolderName( _
    objFSO.BuildPath(sWorkPath, CUSTOMUIPATH)), _
objFSO
objFSO.CopyFile
sCustomUIFullName, _
objFSO.BuildPath(sWorkPath, CUSTOMUIPATH), _
overwritefiles:=True
bContinue = logPack(pack(sWorkPath, sTargetDotmFullName))
End If
End Sub

Function createPath( _
    strPath As String, _
    objFSO As scripting.FileSystemObject _
) As Boolean
createPath = False
If Not objFSO.FolderExists(strPath) Then
    If createPath(objFSO.GetParentFolderName(strPath), objFSO) Then
        createPath = True
        Call objFSO.createFolder(strPath)
    End If
Else
    createPath = True
End If
End Function

```

Das Modul *modPackUnpack*

Dieses Modul enthält Prozeduren für das Aus- sowie Einpacken eines Open XML-Pakets (ZIP-Datei).

Solche Werkzeuge muss der VBA-Entwickler selber erstellen, weil VBA dafür keine Methoden zur Verfügung stellt. Deren Entwicklung gestaltet sich schwierig, weil weder die Windows API noch sonst irgendwelche Microsoft COM-Objekte mit Automatisierungsschnittstelle die Fähigkeit bieten, mit ZIP-Dateien arbeiten zu können.

Im Modul wird die einfache Dateiverwaltung mit dem Scripting File System-Objekt (FSO) ausgeführt. Um Dateien in und aus einem ZIP-Paket zu kopieren, bedarf es des Shell32-Objekts. Beide Objekte gehören zum Lieferumfang von Windows XP und später. Die vorliegende Methode hat jedoch einige Nachteile. Das Kopieren von Dateien in das ZIP-Paket mit dem Shell32-Objekt erfolgt

asynchron. Es ist deshalb notwendig, eine Methode auszuarbeiten, die ermittelt, ab wann dieser Vorgang beendet wurde.

Es handelt sich hier also um ein Konzeptbeispiel, nicht um Produktionscode. Entsprechend haben wir einige Vorgänge der Klarheit halber vereinfacht:

- Der ganze Inhalt des Open XML-Dokuments wird ausgepackt, bearbeitet und wieder eingepackt. Das Resultat ersetzt dann das ursprüngliche Dokument.
- Wird ein Ordner erstellt, nimmt der Code an, dass der nächst höhere Ordner in der Hierarchie bereits existiert

Um kommerzielle Anwendungen zu erstellen, müsste der Code um einiges robuster programmiert werden.

Die Funktion *unpack*

Die Funktion *unpack* (Listing 22.8) packt die im Parameter *strPack* bestimmte Datei in den im Parameter *strFolder* angegebenen Ordner aus. Sie gibt einen der *pckError*-Enumwerte zurück. Diese Prozedur

- verifiziert, dass die in *strPack* angegebene Datei tatsächlich existiert,
- fügt derem Namen die Dateierdung *.zip* zu, nachdem eine allfällig vorhandene Datei gleichen Namens gelöscht wurde,
- löscht einen allfällig vorhanden Ordner mit dem in *strFolder* angegebenen Namen und erstellt einen neuen,
- kopiert den Inhalt der ZIP-Datei in den neuen Ordner,
- entfernt die Dateierdung *zip* vom Dateinamen des Ursprungdokuments.

TIPP

Die Methode *Copyhere* des *Shell32*-Objekts hat ein Parameter *CopyOptions*. Diese legt das gewünschte Verhalten für Meldungen fest. Dies können Sie Ihren Wünschen anpassen. Die folgenden Werten dürfen (wie für die in Kapitel 2 vorgestellte *MsgBox*-Funktion) in beliebiger Zusammenstellung zusammen addiert werden, um mehrere der Optionen zu aktivieren.

4	Kein »Progress«-Dialogfeld einblenden
16	Automatisch »Yes to all« antworten auf jedes eingeblendete Dialogfeld
512	Das Erstellen eines neuen Ordners <i>nicht</i> mit einer Meldung bestätigen
1024	Keine Meldung einblenden, falls ein Fehler vorkommt

Listing 22.8 Bsp22_05.docm: *modPackUnpack* – gemeinsame Strukturen sowie die Funktion *unpack*

```
Option Explicit
Public Enum pckError
    pckErrorOther = -1
    pckErrorSuccess = 0
    pckErrorPackNotFound = 1
    pckErrorFolderNotFound = 2
    pckErrorItemNotFound = 3
End Enum
```

Listing 22.8 Bsp22_05.docm: *modPackUnpack* – gemeinsame Strukturen sowie die Funktion *unpack* (Fortsetzung)

```

Declare Sub Sleep Lib "kernel32" _
    (ByVal dwMilliseconds As Long) _

Function unpack( _
    strPack As String, _
    strFolder As String _
) As Integer

Const CopyOptions As Integer = 1556
Dim objFSO As Scripting.FileSystemObject
Dim objShell As Shell
Dim strTemp As String
On Error GoTo problem

Set objFSO = CreateObject("Scripting.FileSystemObject")
If objFSO.FileExists(strPack) Then
    strTemp = strPack & ".zip"
    If objFSO.FileExists(strTemp) Then
        objFSO.GetFile(strTemp).Delete
    End If
    Name strPack As strTemp

    If objFSO.FolderExists(strFolder) Then
        objFSO.GetFolder(strFolder).Delete
    End If
    objFSO.CreateFolder strFolder

    Set objShell = CreateObject("Shell.Application")
    objShell.NameSpace(strFolder).CopyHere _
        objShell.NameSpace(strTemp).Items, _
        CopyOptions
    Set objShell = Nothing

    Name strTemp As strPack
    unpack = pckError.pckErrorSuccess
Else
    unpack = pckError.pckErrorPackNotFound
End If
Set objFSO = Nothing
Exit Function

problem:
    unpack = pckError.pckErrorOther
    On Error Resume Next
    If strTemp <> "" Then
        Name strTemp As strPack
    End If
    Set objShell = Nothing
    Set objFSO = Nothing
    Err.Clear
End Function

```


Die Funktion *pack*

Wie ihr Name verrät, packt die Funktion *pack* den Inhalt des im Parameter *strFolder* angegebenen Ordners in eine Datei mit dem im Parameter *strFile* festgelegten Namen ein. Sie gibt einen der *pckError* Enum-Werte zurück. Die Prozedur führt die folgenden Handlungen aus:

- Verifiziert, dass der in *strFolder* angegebene Ordner tatsächlich existiert
- Fügt dem in *strDatei* angegebenen Namen die Endung *.zip* zu, und nachdem eine eventuell vorhandene Datei gleichen Namens gelöscht wurde, erstellt eine neue ZIP-Datei
- Kopiert den Inhalt des Ordners *strFolder* in die ZIP-Datei und wartet, bis die Anzahl an Dateien im Ordner der obersten Ebene der Anzahl an Dateien im Ordner *strFolder* entspricht. Die Wartezeit wird auf jeden Fall nach Erreichen der durch *MaxWait* bestimmten Anzahl Millisekunden abgebrochen.
- Löscht eine eventuell vorhandene Datei mit dem in *strFile* angegebenen Namen und benennt die ZIP-Datei mit diesem Namen um

WICHTIG Windows erkennt nicht jede Datei mit der Endung *.zip* als ZIP-Datei. Am Dateianfang muss eine entsprechende »Signatur« vorhanden sein. Hier wird dies mit den folgenden Codezeilen erzielt:

```
Open strTemp For Output As #1
Print #1, "PK" & Chr$(5) & Chr$(6) & String(18, 0)
Close #1
```

Diese Funktion unterstützt ebenfalls den Parameter *CopyOptions*, welcher bereits im vorangegangenen Abschnitt vorgestellt wurde.

Listing 22.9 Bsp22_05.docm: *modPackUnpack* – die Funktion *pack*

```
Function pack(
    strFolder As String, _
    strPack As String _
) As Integer

Const CopyOptions As Integer = 1556
Const EachWait As Long = 100 ' 100ms
Const MaxWait As Long = 10000 ' 10s
Dim lngWait As Long
Dim objShell As Shell
Dim objFSO As Scripting.FileSystemObject
Dim strTemp As String
On Error GoTo problem

Set objFSO = CreateObject("Scripting.FileSystemObject")
If objFSO.FolderExists(strFolder) Then
    strTemp = strPack & ".zip"
    If objFSO.FileExists(strTemp) Then
        objFSO.GetFile(strTemp).Delete
    End If
    Open strTemp For Output As #1
    Print #1, "PK" & Chr$(5) & Chr$(6) & String(18, 0)
    Close #1
```

Listing 22.9 Bsp22_05.docm: *modPackUnpack* – die Funktion *pack* (Fortsetzung)

```

Set objShell = CreateObject("Shell.Application")
objShell.Namespace(strTemp).CopyHere _
    objShell.Namespace(strFolder).Items, _
    CopyOptions

On Error Resume Next
lngWait = 0
Do Until
    ((objShell.Namespace(strTemp).Items.Count >= _
        objShell.Namespace(strFolder).Items.Count) _
    Or (lngWait > MaxWait))
    DoEvents
    Sleep EachWait
    lngWait = lngWait + EachWait
Loop
On Error GoTo 0
Set objShell = Nothing

If objFSO.FileExists(strPack) Then
    objFSO.GetFile(strPack).Delete
End If
Name strTemp As strPack
pack = pckError.pckErrorSuccess
Else
    pack = pckError.pckErrorFolderNotFound
End If
Set objFSO = Nothing
Exit Function

problem:
pack = pckError.pckErrorOther
On Error Resume Next
If strTemp <> "" Then
    Name strTemp As strPack
End If
Set objShell = Nothing
Set objFSO = Nothing
Err.Clear
End Function

```

Teile, Beziehungen und Inhaltstypen

Durch die bisherigen Beispiele haben Sie einige Grundkenntnisse des Open XML-Formats gesammelt. Wie Sie sich vorstellen können, gestaltet sich der Aufbau eines »echten« Open XML-Dokuments meistens etwas komplexer. In diesem Abschnitt stellen wir die Konzepte, Teile, Inhaltstypen und Beziehungen eingehender vor. Somit haben Sie die nötigen Werkzeuge in der Hand, die Open XML-Dokumentation anzuwenden.

Teile und ihre Typen

Den Inhalt des Beispieldokuments *Bsp22_04.xml* im Abschnitt »Das »flache« OPC-Format« ab Seite 919 bilden die beiden Teile Hauptdokument und Kopfzeile. Sogar ein neues, »leeres«, von Word erstelltes Dokument enthält einige Teile mehr, wie

```
/rels/.rels
/word/_rels/document.xml.rels
/word/document.xml
/word/theme/theme1.xml
/word/settings.xml
/word/fontTable.xml
/word/webSettings.xml
/docProps/app.xml
/docProps/core.xml
/word/styles.xml
```

Im Anbetracht dieser Liste ist anzunehmen, dass es viele verschiedene Teil-Typen gibt. Darin wird der Inhalt für Kopf- und Fußzeilen, Fußnoten, Kommentare, Grafiken usw. verwaltet.

Was den Open XML-Standard betrifft, stehen für Word-Dokumente eine große Anzahl Typen für Teile bereit. Diese können in drei verschiedenen Gruppen unterteilt werden:

- Word-spezifische (Tabelle 22.1)
- Allgemeine, die dem Open XML-Format aller Office Anwendungen gemeinsam sind (Textverarbeitung, Tabellenblätter, usw.) (Tabelle 22.2)
- Funktionalitätsspezifische, wie für Diagramme oder Zeichnungen (werden hier nicht aufgelistet)

Zudem verfügt Word über einige Typen, die nicht in den ISO/ECMA-Standards definiert sind. (Tabelle 22.3)

In diesen Tabellen gibt die Spalte »Anzahl« an, wieviele Teile eines bestimmten Typs sich im Paket eines Dokuments befinden dürfen. Einige Typen dürfen nur einmal vorkommen, während andere höchstens zweimal vorhanden sein können. Weitere Typen dürfen beliebig oft integriert werden. In diesem Fall werden die Teile fortlaufend nummeriert: *header1.xml*, *header2.xml* usw. Folgend einige Beispiele:

- Ein, und nur ein einziger Hauptdokumentteil ist vorgeschrieben (Anzahl = 1)
- 0 oder 1 Glossarteil ist erlaubt (Anzahl=0–1)
- 0, 1 oder 2 Kommentarteile (Anzahl=0–2). Einer dieser Teile enthält die Kommentare für das Hauptdokument. Der andere verwaltet Kommentare für das Glossar.
- 0 oder mehr Kopfzeileteile dürfen vorhanden sein (Anzahl=mehrere)

Im Fall von Anzahl=0–2 gehört immer ein Teil dem Hauptdokument und der andere dem Glossar.

Jeder Teiltyp akzeptiert ein oder mehrere URI für Inhaltstypen (contentType). In Teilen, die XML enthalten, werden ein oder mehrere URI für Namensräume definiert oder zumindest ein URI für einen Wurzelnamensraum beinhalten.

HINWEIS Diese URIs führen wir im Text nicht auf, Sie werden jedoch viele der häufig vorkommenden URIs in *Kap22_CT.txt* und in *Kap22_NS.txt* auf der CD-ROM zum Buch im Ordner *\Beilagen\XMLPackages* finden.

Tabelle 22.1 Word-spezifische Teiltypen

Typname	Anzahl	Beschreibung
Alternative Format Import	mehrere	Diese Teile sind für das Importieren von nicht-Open XML-Dokumenten vorgesehen. Sie unterstützen die Konvertierung des Inhalts in das neue Format für die Verwendung im Zieldokument. Beispiel: Ein HTML-Fragment wird in das Dokument eingefügt. Word wird es vermutlich in WordProcessingML umwandeln und gleichzeitig die ursprüngliche Information in einem eigenen Teil speichern.
Comments (Kommentare)	0–2	Die Kommentare für alle Kommentar-Referenzen im Haupt- oder Glossar-Dokument
Document Settings (Dokumenteinstellungen)	0–2	Verwaltet verschiedene Informationen, wie die allgemeinen Einstellungen für Fußnoten oder Datenquelleninformationen für den Seriendruck
Endnotes (Endnoten)	0–2	Die Endnoten für alle Endnoten-Referenzen im Haupt- oder Glossar-Dokument
Font Table (Schriftart-Tabelle)	0–2	Informationen zu den benutzten Schriftarten (wird beispielsweise für der Substitution einer Schriftart, wenn auf der Maschine nicht vorhanden ist, verwendet).
Footer (Fußzeile)	mehrere	Je ein Fußzeilenteil für jeden Fußzeilentyp (erste Seite, ungerade Seite, gerade Seite) pro Dokumentabschnitt. Je ein Satz für das Hauptdokument sowie das Glossar-Dokument.
Footnotes (Fußnoten)	0–2	Alle Fußnoten des Haupt- oder des Glossar-Dokuments
Glossary Document (Glossar-Dokument)	0–1	Ein Speicherplatz für im Hauptdokument referenzierte Dokumentfragmente
Header (Kopfzeile)	mehrere	Je ein Kopfzeilenteil für jeden Kopfzeilentyp (erste Seite, ungerade Seite, gerade Seite) pro Dokumentabschnitt. Je ein Satz für das Hauptdokument sowie das Glossar-Dokument.
Main Document (Hauptdokument)	1	Der Textkörper des Word-Dokuments oder der Word-Dokumentvorlage
Numbering Definitions (Definitionen der Nummerierungen)	0–2	Eine Definition für jedes benutzte Nummerierungs- bzw. Aufzählungsschema
Style Definitions (Definitionen der Formatvorlagen)	0–2	Definitionen für benutzerdefinierte und Word-eigene Formatvorlagen.
Web Settings (Einstellungen für Internet-Dokumente)	0–2	Einstellungen für Webseitendokumente, wie beispielsweise Einstellungen für ein Frameset-Objekt

Tabelle 22.2 Gemeinsame Office-Teiltypen

Typname	Anzahl	Beschreibung
Additional Characteristics (erweiterte Eigenschaften)	mehrere	Speicher für erweiterte Informationen (beispielsweise die Höchstzahl Spalten oder Zeilen eines Kalkulationsblatts)
Audio	mehrere	Ein Teil, der Audio-Information speichert
Bibliography (Bibliographie)	0–1	Ein Teil, der Bibliographie-Informationen speichert
Content (Inhalt)	mehrere	Enthält jedes unterstützte XML-Vokabular. Aus den Standards ist zu entnehmen, dass dieser Teiltyp für Objekte wie OMath-XML ist.
Custom XML Data Storage (Speicher für Benutzerdefiniertes XML)	mehrere	Beliebiges XML. In Word werden diese als »Custom XML Parts« bezeichnet.
Custom XML Data Storage Properties (Eigenschaften des Custom XML Data Storage)	mehrere	Informationen zum Inhalt/Speicher eines Custom XML Storgeteils. Meistens 1 pro Custom XML Storgeteil.
Digital Signature Origin (Ursprung der digitalen Signatur)	0–1	Ein leerer Teil, der als Anfangspunkt für die Aufspürung von Informationen zu einer digitalen Signatur dient
Digital Signature XML Signature	0–1	Ein Teil für die digitale Signatur des Pakets
Embedded Control Persistence (Speicher für Daten von eingebetteten Steuer- elementen)	mehrere	Für ActiveX-Steuerelemente, die Daten im XML-Format speichern. Ein Teil pro Steuerelement mit diesem Bedürfnis.
Embedded Object (Eingebettetes Steuerelement)	mehrere	Ein beliebiges, eingebettetes OLE-Objekt
Embedded Package (Eingebettetes Paket)	mehrere	Ein beliebiges Open XML-Paket, das im Dokument eingebettet wurde. Beispiel: eine xlsx-Excel-Arbeitsmappe. Das gesamte Paket, im ZIP-Format, wird im Word-Dokument-Paket gespeichert.
File Properties, Extended (Erweiterte Dokument- eigenschaften)	0–1	Erweiterte Eigenschaften sind Eigenschaften, die spezifisch sind für den Typ des Open XML-Dokuments. Diese umfassen beispielsweise bei einem Word-Dokument den Namen der verbundenen Dokumentvorlage, statistische Informationen wie die Anzahl Seiten, Zeichen, Wörter und Absätze usw.
File Properties, Core (Kern-Dokument- eigenschaften)	0–1	Kern-Eigenschaften sind Eigenschaften, die allen Office-Anwendungen gemeinsam sind. Dazu gehören Author (Autor), Title (Titel) usw.
File Properties, Custom (Benutzerdefinierte Dokumenteigenschaften)	0–1	Benutzerdefinierte Eigenschaften

Tabelle 22.2 Gemeinsame Office-Teiltypen (Fortsetzung)

Typname	Anzahl	Beschreibung
Font (Schriftart)	mehrere	Ein Teil für jede Schriftart, die im Paket <i>eingebettet</i> ist. (Normalerweise ist dies nicht der Fall, die Schriftarten werden dynamisch geladen. Wird eine TrueType-Schriftart benutzt, kann sie mit dem Dokument verteilt werden, indem die Information im Dokument eingebettet ist. Die Option dafür befindet sich in der Registerkarte <i>Speichern</i> der Word-Optionen. Die Dateigröße wird dadurch erheblich erhöht.)
Image (Grafik)	mehrere	Ein Teil pro Grafik
Printer Settings (Druckereinstellungen)	mehrere	Je nach Einstellungen könnte das Paket einen solchen Teil pro Dokumentabschnitt enthalten
Thumbnail (Miniatur)	0–1	Ein Thumbnail des Dokuments
Video	mehrere	Eine Videodatei wie AVI oder MPEG

Tabelle 22.3 Zusätzliche Microsoft Teil-Typen

Typname	Anzahl	Beschreibung
Attached Toolbars (Angehängte Symbolleisten)	0–1	Ein Teil mit binären Daten für die Unterstützung von benutzerdefinierten Symbolleisten
Customizations (Anpassungen)	0–2	Tastenkombinationen und Anpassungen von Symbolleisten
CustomUI	0–1	Anpassungen des Menübands
CustomUI2	0–1	Anpassungen des Menübands sowie der Backstage-Ansicht
Embedded Control Persistence Binary Data (Binärdaten eines eingebetteten Steuerelements)	mehrere	Unterstützt (ActiveX) Steuerelemente, die, anstelle von XML, binäre Daten speichern müssen. Ein Teil pro Steuerelement, das binäre Daten speichert.
Mail Merge Recipient Data (Daten über Seriendruckempfänger)	0–1	Enthält die Listen der eingebundenen bzw. ausgeschlossenen Seriendruckempfänger. Der Teiltyp ist in der Dokumentation des Standards erwähnt, jedoch nicht dokumentiert.
Quick Access Toolbar Customizations (Anpassungen zur Symbolleiste für den Schnellzugriff)	0–1	Anpassungen zur Symbolleiste für den Schnellzugriff
Styles With Effects (Formatvorlagen mit Effekten)	0–2	Speichert eine Kopie des Teils <i>Styles</i> (wird mit der Word 2010-Funktionalität für Effekterweiterungen benutzt)

Mehr zum Thema Beziehungen

In diesem Abschnitt werden die Typen für Quellen und Ziele einer Beziehung behandelt. Wir beschreiben zudem die Unterschiede zwischen impliziten und expliziten Beziehungen. Letztlich werden erlaubte Beziehungen, Eigenschaften und das Handhaben der Beziehungen für Custom XML Parts vorgestellt.

Quellen und Ziele

Jede »Relationship« im Open XML umschreibt eine Beziehung zwischen zwei Objekten: einer Quelle und einem Ziel. Die Quelle kann das Paket sein oder ein Teil innerhalb des Pakets. Das Ziel darf ein Teil im Paket sein (eine »interne Quelle«) oder könnte auf etwas außerhalb des Pakets zeigen (eine »externe Quelle«).

Wir sahen in *Bsp22_04* beispielsweise Folgendes:

- Wie das Paket eine Beziehung definiert, die als Ziel den Teil Hauptdokument hat
- Falls im Dokument eine Kopfzeile vorkommt, wird diese Beziehung im Teil Hauptdokument definiert, die als Ziel den Teil Kopfzeile (»header«) hat

Jede Quelle, die eine Beziehung definieren muss, hat einen Teil Beziehungen (Relationships). Dieser enthält alle Beziehungen für den Teil. Umgekehrt ist das nicht der Fall: Ein Ziel führt keine Informationen mit, die Auskunft darüber geben, welcher Teil seine Quelle ist. Es ist also allgemein gesehen einfacher, die Beziehungen eines Pakets von Quelle zum Ziel zu verfolgen, als herauszufinden, welcher Teil die Quelle eines Zielteils ist.

Das Auffinden eines Beziehungsteils gestaltet sich relativ einfach, weil sein Name auf dem des Quellteil basiert. Hat ein Teil den Namen `/abc/def.ghi`, heißt der damit verbundenen Beziehungsteil `/abc/_rels/def.ghi.rels`.

Wenn wir annehmen, dass das Paket selber den Namen `/` hat, ist logischerweise der Name des Teils mit allen Beziehungen der obersten Ebene `/_rels/.rels`.

Es müsste also möglich sein, durch alle Teile des Pakets zu navigieren, indem

1. der Beziehungsteil `/_rels/.rels` geöffnet wird und
2. für jede interne Beziehung
 - den Zielnamen lesen
 - den Namen für den Beziehungsteil daraus zusammenstellen
 - falls der Beziehungsteil vorhanden ist, ihn öffnen und Schritt (2) ausführen

Bitte beachten Sie, dass dies alles ausgeführt wird, ohne einen Zielteil zu öffnen. Nur sein Beziehungsteil wird angesprochen (sofern vorhanden).

Die Mehrzahl der Beziehungstypen haben interne Ziele (Teile innerhalb des Pakets). Aus diesem Grund befinden sich die meisten Zieltypen in Tabelle 22.1 bis Tabelle 22.3. Weitere Beziehungstypen für externe Ziele sind in Tabelle 22.4 aufgelistet. Einige Beziehungstypen werden für interne sowie externe Beziehungen eingesetzt, wie etwa Audioteile (bzw. -dateien), Hyperlinks, Grafiken oder Videoteile (bzw. -dateien).

Tabelle 22.4 Externe Zieltypen

Zieltyp	Bemerkungen
Attached Template (Angehängte Dokumentvorlage)	Darf auch im Teil <i>settings</i> definiert sein
FrameSet	Darf auch im Teil <i>WebSettings</i> definiert sein
SubDocument (Filialdokument)	Falls es sich um ein Master-Dokument handelt, werden die Beziehungen für die Subdokumente im Beziehungsteil des Hauptdokuments definiert
Mail Merge Data Source (Seriendruckdatenquelle)	Darf auch im Teil <i>settings</i> definiert sein
Mail Merge Header Source (Seriendrucksteuersatz)	Darf auch im Teil <i>settings</i> definiert sein
XSL Transformation	Wird eine Transformation beim Speichern automatisch ausgeführt, wird sie im Teil <i>settings</i> definiert

Implizite und explizite Beziehungen

Es gibt zwei Arten Beziehung zwischen Quell- und Zielteilen: die explizite und die implizite.

Ist die Beziehung *explizit*, enthält das XML des Quellteils den Namen eines *r:id*-Werts, wie *rId5*. Der *settings*-Teil könnte beispielsweise eine Beziehung zu einer Seriendruckdatenquelle wie folgt definieren:

```
<w:mailMerge>
. . .
<w:dataSource r:id="rId5"/>
. . .
</w:mailMerge>
```

Im Beziehungsteil des Teils *settings* wird die Beziehung aufgesucht, deren ID *rId5* ist. Daraus werden den Beziehungstyp und das Ziel ermittelt.

Ist die Beziehung *implizit*, enthält der Quellteil kein *r:id*-Attribut für das Ziel. Die Referenz für eine Fußnote im Hauptdokumentteil wird beispielsweise durch eine »gewöhnliche« Id-Nummer für den Zielteil definiert:

```
<w:footnoteReference w:id="5"/>
```

Da alle Fußnotentexte in einem einzigen Teil aufgelistet werden, wäre es überflüssig, *r:id="rId3"* in jedem *footnoteReference*-Element einzugeben. (Und es kommt auch nie vor.)

Folgende Fragen stellen sich

1. Ist eine Beziehung zwischen der Quelle Hauptdokumentteil und dem Ziel Fußnotenteil wirklich notwendig?
Ja, weil diese den Namen des Fußnotenteils bereitstellt.
2. Enthält der Hauptdokumentteil kein Element mit einem Attribut für einen Id-Wert der Referenz (wie *r:id="rId3"*), wie kann die Beziehung im Beziehungsteil nachgeschlagen werden?

Ein Beziehungselement (Relationship) mit einem Type-Attribut wird gesucht, wie beispielsweise für Fußnoten:

```
Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/footnotes"
```

Erlaubte Beziehungstypen und Typen-URLs

Die Open XML-Standards definieren, welche Arten von Beziehungen erlaubt, welche fakultativ und welche für eine bestimmte Kombination von Quellteil- und Zieltypen erforderlich sind. Erlaubte Beziehungen sind in Tabelle 22.5 aufgelistet.

CD-ROM Jeder Beziehungstyp hat eine eigene URL. Eine Liste der meistverwendeten finden Sie auf der CD-ROM im Ordner *\Beilagen\XMLPackages* in der Datei *Kap22_RT.txt*.

Tabelle 22.5 Erlaubte Beziehungen

Quelle(n)	Implizit/Explizit	Ziel(e)
Package (Paket)	E	Main Document (erforderlich) Digital Signature Origin Properties (Core, Custom, Extended)
Comments (Kommentare) Endnotes, Footnotes (End-, Fußnoten) Header, Footer (Kopf-, Fußzeile) Glossary Document (Glossardokument) Main Document (Hauptdokument)	E	Alternative Format Import Content Various diagram parts Embedded Control Persistence Embedded Object Embedded Package Hyperlink Image Video
Glossary Document (Glossar- Dokument) Main Document (Hauptdokument)	I	Comments Document Settings Endnotes Font Table Footnotes Numbering Definitions Style Definitions Web Settings
Glossary Document (Glossar- Dokument) Main Document (Hauptdokument)	E	Footer Header Printer Settings

Tabelle 22.5 Erlaubte Beziehungen (Fortsetzung)

Quelle(n)	Implizit/Explizit	Ziel(e)
Main Document (Hauptdokument)	I	Additional Characteristics Bibliography Custom XML Data Glossary Document Theme Thumbnail
Main Document (Hauptdokument)	E	SubDocument
Font Table (Schriftart-Tabelle)	E	Font
Numbering Definitions (Definitionen für die Nummerierung)	E	Image
Settings (Einstellungen)	E	Document Template Mail Merge Data Source Mail Merge Header Source Recipient Data XSL Transformation
Web Settings (Einstellungen für Webseiten)	E	FrameSet
Additional Characteristics (zusätzliche Eigenschaften) Bibliography (Bibliographie) Custom XML Data (Benutzerdefinierte XML-Daten)	E	Custom XML Data Properties
Digital Signature Origin (Ursprung der digitalen Signatur)	E	Digital Signature XML-Signatur
Embedded Object (eingebettetes Objekt) Embedded Package (eingebettetes Paket)	E	Hyperlink

Keine anderen Teiltypen können die Quelle einer Beziehung bilden (wie Alternative Format Import, Style Definitions, Audio, Custom XML Data Properties, Digital Signature XML Signature, Embedded Content Persistence, Properties (Core, Custom, Extended), Font, Image, Printer Settings, Thumbnail, Video, Hyperlink).

Beziehungen zu Teilen für Custom XML Parts (Daten)

Der Tabelle 22.5 ist zu entnehmen, dass nur ein Hauptdokumentteil ein Ziel des Typs »Custom XML (Data) Part« enthalten kann. Im Kapitel 7 wurde veranschaulicht, wie ein Custom XML Part mit einem Inhaltssteuerelement verknüpft wird. Inhaltssteuerelemente dürfen in Kopf- und Fußzeilen eingefügt werden. Eigentlich würden wir meinen, ein in der Kopfzeile eingefügtes Objekt müsste eine Referenz auf einen Teil haben, der außerhalb der Kopfzeile liegt. Folglich müsste ein Beziehungsteil für die Kopfzeile vorliegen, worin eine Beziehung zu finden ist, die einen Custom XML Part als Ziel hat. Aber laut Tabelle 22.5 ist dies nicht erlaubt. Wie wird es dann gemacht?

Datenverbindungen für Inhaltssteuerelemente sind ein Spezialfall. Kommt ein Inhaltssteuerelement vor, in irgendeinem Teil, der diese Objekte unterstützt (also Kopf-, Fußzeile, Fuß-, Endnote oder Hauptdokument), hat das Hauptdokument einen Beziehungsteil, der die Beziehung zum Custom XML Part definiert.

Der Teil, der das Inhaltssteuerelement mit Datenverknüpfung enthält, benutzt den Namensraum, einen Xpath-Ausdruck sowie ein StoreItemID-Wert, um den Custom XML Part zu referenzieren. Folgende Codezeilen dienen als Beispiel:

```
w:prefixMappings="xmlns:ns0='http://www.beispiele.com/kap22'"
w:xpath="/ns0:Adresse[1]/ns0:Ort[1]"
w:storeItemID="{55AF091B-3C7A-41E3-B477-F2FDAA23CFDA}"
```

Der Namensraum und die Xpath-Angabe können in weiteren Custom XML Parts desselben Pakets vorkommen und sind somit nicht unbedingt eindeutig. Der StoreItemID-Wert ist der einzige, der garantiert eindeutig ist. Dieser ist jedoch im getrennten Custom XML Properties-Teil des Custom XML Parts gespeichert und ist daher keine direkte Referenzierung.

HINWEIS

Weiteres zu Custom XML Parts lesen Sie im Abschnitt »Dokumenteigenschaften, Custom XML Parts und Inhaltssteuerelemente« ab Seite 948.

Standards für die Benennungen von Teilen

Im Beispiel *Bsp22_03.xml* haben wir */bsp22/03.xml* als den Namen des Hauptdokumentteils eingesetzt. Würden Sie dieses Beispiel in Word öffnen, unter einem anderen Namen abspeichern und danach das XML des Dateiinhalts anschauen, würden Sie sehen, dass Word einen anderen Namen eingesetzt hat. Word benutzt für diesen Teil den im Open XML-Standard vorgesehenen Standardnamen */word/document.xml*.

Word hat eine interne Liste von Ordner- und Dateinamen für jeden Teiltyp. Wie Sie jedoch gesehen haben, bleibt ein Dokument mit anderen Namen gültig. Es ist nicht auszuschließen, dass Ihr Code es einmal mit einem solchen Dokument zu tun haben wird, weshalb er entsprechend programmiert werden soll. Das Beispiel im Abschnitt »Beispiel: Seriendruck-Datenquelleninformationen abfragen und entfernen« ab Seite 948 bietet einige Hinweise, wie die Teile und Beziehungen eines Pakets durchzuschleifen sind, um festzustellen, wo sich etwas befindet und wie es in Wirklichkeit heißt.

TIPP

Um herauszufinden, welche Standardnamen Word benutzt, ist es am einfachsten, ein Beispiel in Words Benutzerschnittstelle zu erstellen, dieses zu speichern und anschließend die Ordnerstruktur zu analysieren.

WICHTIG

Eine wichtige Regel gilt es zu beachten: Ist das Ziel in einer Beziehung eine relative URL innerhalb des Pakets, ist diese Pfadangabe relativ zum Standort der *Quelle der Beziehung* und nicht relativ zur Datei *.rels*, worin sich die Beziehung befindet.

Die programmtechnische Arbeit mit Open XML: eine Übersicht

Vor der Einführung des Word 2003 XML-Dateiformats war die Automatisierung des Objektmodells die einzige zuverlässige Methode, Word-Dokumente zu erstellen oder zu bearbeiten. Ab Word 97 wurde dies mit jeder COM-fähigen Programmiersprache möglich, sei sie VBA, VB6, Delphi oder eine .NET-Sprache (über die Schnittstelle der »Interop Assemblies«).

Dank den XML-Dateiformaten von Word 2003 und später ist es nun möglich, Word-Dokumente ohne installierte Word-Anwendung zu erstellen und zu bearbeiten.

Nachfolgend stellen wir die Vor- und Nachteile der zwei Methoden gegenüber.

Das XML-Dateiformat

Einige Gründe sprechen für die direkte Manipulation einer Dokument-XML-Datei:

- Die Dokumente können durch eine beliebige Programmiersprache, (vorausgesetzt, sie verfügt über die notwendigen Werkzeuge) auf einem beliebigen Betriebssystem bearbeitet werden. XML-Dateien sind reine Textdateien. Open XML-Dateien sind gewöhnliche ZIP-Behälter.
- Die Word-Anwendung muss dabei nicht installiert sein
- Zudem erfolgt die Ausführung der Bearbeitung schneller als über das Objektmodell

Diese Überlegungen sprechen für die direkte Bearbeitung des XML für Dokumente, die auf einem Server erstellt oder bearbeitet werden müssen. Die Word-Anwendung wurde nicht für die Ausführung in einer Server-Umgebung konzipiert. Wird sie so automatisiert, entstehen Probleme, sobald Word eine Antwort vom Benutzer verlangt. Ein Web-Server könnte beispielsweise Berichte erstellen, die auf Daten von einem SQL-Server basieren.

Das Word-Objektmodell

Auch die Arbeit mit dem Objektmodell hat ihre Vorteile, wie etwa

- Office-Anwendungen stellen einige Dienstleistungen zur Verfügung, die ohne die Anwendung nur mit großer Mühe angeboten werden können. Ein wichtiges Beispiel ist das Ausdrucken eines Dokuments.

Weder Windows noch Crystal Reports oder sonst ein Entwicklertool bietet die Möglichkeit an, ein Word-Dokument ohne Hinzuziehen seiner Anwendung auszudrucken. Der Entwickler müsste mühsam jede Einzelheit der Word-Dokumentstruktur und -paginierung kennen und herausfinden, wie dies für die Ausgabe auf jedem möglichen Drucker zu codieren ist. Das Open XML des Dokuments müsste in Detail abgearbeitet und die Abbildung für den Ausdruck zusammengesetzt werden. Dieses Vorhaben wäre extrem kostspielig und zeitintensiv.

- Auch die direkte Bearbeitung eines Dokuments verlangt eine vertiefte Kenntnis der Word- sowie Open XML-Strukturen. Wird ein Absatz beispielsweise samt seiner Objekte durch die Word-Anwendung gelöscht, stellt Word sicher, dass alle Beziehungen und Referenzen nachgeführt werden. Diese Handlungen müssen bei der Bearbeitung mit Open XML explizit durch den Programmiercode ausgeführt werden.

Enthält der gelöschte Bereich beispielsweise eine Referenz zu einer Fußnote, muss der entsprechende Teil geöffnet und auch diese Fußnote gelöscht werden.

Das Löschen einer Grafik gestaltet sich noch komplexer, da die Grafik unter Umständen sonst wo im Dokument eingesetzt werden könnte. Wenn nicht, muss die Beziehung und evtl. sein Teil gelöscht werden. Zudem müssten weitere Referenzen geprüft und angepasst werden.

Das Zusammenspiel der zwei Methoden

Obwohl die Bearbeitung eines Dokuments über Automatisierung der Word-Anwendung in eine Open XML-Datei resultiert, es ist nicht möglich, über Automatisierung die Paket- und XML-Strukturen genau zu bestimmen. Das Objektmodell bietet eine Dienstleistung an, die für die ausgeführten Handlungen Open XML schreibt – wie Word es für gut hält.

Als Beispiel nehmen wir ein Logo, das im Paket gespeichert und später für den Gebrauch in Kopf- und Fußzeilen verwendet werden soll. Mit Open XML kann der Entwickler genau festlegen, wo die Grafik zu speichern und wie sie in den verschiedenen Teilen einzubinden und zu referenzieren ist.

Über das Objektmodell ist diese Feinabstimmung nicht möglich. Word entscheidet beim Einfügen, wo im Paket die Grafik gespeichert und wie sie in das Dokument eingebunden wird. Vielleicht wird es die Grafikdaten direkt in die Zeile mit dem XML der Fußzeile schreiben. Auch möglich wäre, dass Word die Grafik in einem getrennten Teil speichert, eine Beziehung erstellt und in die Fußzeile eine Referenz dazu einfügt. Der Entwickler kann das nicht beeinflussen.

Nur eine eingehende Analyse des Projekts kann entscheiden, welche Methode besser zum Ziel führt.

Was *nicht* möglich ist, ist die gleichzeitige Bearbeitung eines Dokuments durch das Objektmodell sowie durch Manipulieren des Open XML-Pakets. Die Datei wird beim Öffnen von Windows gesperrt, sodass nur der eine Prozess damit arbeiten kann. Falls Sie ein Dokument auf beide Arten bearbeiten wollen, müssen diese Handlungen nacheinander erfolgen. Das Dokument muss geschlossen und von Windows freigegeben werden, bevor die Bearbeitung mit der anderen Methode erfolgen kann.

Zugang zum Open XML durch das Objektmodell

Das Word-Objektmodell bietet einige Methoden, die den Zugriff auf den XML-Inhalt eines Dokuments sowie auf gewisse Teile ermöglichen:

- Custom XML Parts können zugefügt, bearbeitet und mit Inhaltssteuerelemente verknüpft werden (im Kapitel 7 beschrieben)
- Das Word 2003 XML für die aktuelle Markierung oder eines Dokumentbereichs kann einer Zeichenkette zugewiesen werden (die Eigenschaft XML des Range- bzw. Selection-Objekts)
- Das Open XML für die aktuelle Markierung oder eines Dokumentbereichs kann einer Zeichenkette zugewiesen werden (die Eigenschaft WordOpenXML des Range- bzw. Selection-Objekts)
- Der Inhalt eines bestimmten Bereichs kann als eine Datei jedes unterstützten Dateiformats (wie Word 2003 XML, flaches OPC oder Open XML) gespeichert werden (die Methode ExportFragment des Range-Objekts)
- Der Inhalt einer in einem unterstützten Format gespeicherten Datei kann importiert werden (die Methode ImportFragment des Range-Objekts)
- Inhalt in der Form von WordProcessingXML oder Open XML kann als Zeichenkette in ein Dokument eingefügt werden (die Methode InsertXML des Range- bzw. Selection-Objekts)
Diese Methode setzt voraus, dass der Bereich oder die Markierung eine zulässige Stelle ist. Die Methode verursacht eine Fehlermeldung, wenn beispielsweise der Zielbereich eine Grafik ist.

Beim Importieren oder Exportieren von XML in ein bzw. aus einem in Word geöffneten Dokument spielt es keine Rolle, ob das Dokument selber als *.doc*, *.docx*, *.rtf* oder überhaupt auf der Festplatte gespeichert wurde. Word hält eine Repräsentation des Dokuments im Speicher bereit und benutzt diese Informationen für Umwandlung der Information und deren Einfügen bzw. für das Bereitstellen des XML.

Dabei behandelt es sich nicht um ein kleines XML-Fragment, wie `<w:t>abc</w:t>`, sondern um das benötigte XML für ein gültiges Word-Dokument, das als selbstständige Datei in Word geöffnet werden könnte. Im Fall von Word-Open XML bedeutet dies, dass es in der Form des flachen OPC-Formats vorhanden ist (siehe den Abschnitt »Das »flache« OPC-Format« ab Seite 919).

Des Weiteren verweigert Word das Einfügen über InsertXML, wenn das XML mit einem BOM (Byte Order Mark) anfängt. (Ein BOM trägt Informationen über Unicode.) Word kann eine Datei mit einem BOM problemlos öffnen, es meckert nur, wenn es in der Zeichenkette vorhanden ist, die über InsertFile in das Dokument eingefügt werden soll.

Werkzeuge für die direkte Bearbeitung eines Open XML-Pakets

Da Open XML-Pakete aus einem ZIP-Behälter mit reinen Textdateien bestehen, können sie mit beliebigen Programmiersprachen oder Systemen bearbeitet werden, die diese einfache Voraussetzungen unterstützen. Wie für die Arbeit mit XML vereinfacht ein gutes Werkzeug die Aufgabe erheblich. Eine Programmiersprache oder Bibliothek/Add-On soll die folgenden Voraussetzungen erfüllen. Sie muss ...

- mit ZIP-Dateien und deren Inhalt arbeiten können. Noch vorteilhafter ist die Fähigkeit, mit jeder Art Paket und dessen Teilen arbeiten zu können. Sie muss den standardmäßigen Umgang mit Ordnern und Dateien unterstützen, wie das Kopieren, Öffnen und Löschen.
- mit XML-Dateien arbeiten können. Welche Funktionalitäten benötigt werden, hängt jeweils vom einzelnen Projekt ab. Auf jedem Fall müssen XML-Dokumente geladen und gespeichert werden können, sowie

- das Erstellen, Löschen und Navigieren einer Hierarchie von XML-Knotenpunkten, die Zählung von Knotenpunkten, sowie das Lesen deren Daten
- die Verwendung von Xpath, um einzelne sowie Sammlungen von Knotenpunkten anzusprechen
- den Einsatz von XSLT unterstützen, um das XML zu transformieren.

Für das Windows-Betriebssystem stellt .NET Framework die vollumfänglichste Umgebung für solche Arbeiten zur Verfügung. Es umfasst Bereiche für die Arbeit mit XML sowie (ab Version 3.0) mit Paketen. Die Klassen unter System.IO.Packaging, beispielsweise, bieten Werkzeuge für die allgemeine Arbeit mit ZIP-Dateien ab Windows XP SP3.

Zudem hat Microsoft ein Open XML SDK entwickelt, das eine strengere Typisierung des Zugangs zu Open XML-Paketen bereitstellt.

Viele Informationsquellen stehen für die Arbeit mit Open XML in .NET Framework zur Verfügung. (Was auch der Grund ist, warum die Beispiele in diesem Kapitel VBA sind – diese Dokumentation fehlt.)

HINWEIS

Die Dokumentation für das Open XML Format SDK 2.0 (»Software Development Kit«) für Microsoft .NET Framework 3.5 in Zusammenarbeit mit Office 2007 sowie Office 2010 steht auf <http://msdn.microsoft.com/en-us/library/bb448854.aspx> bereit.

Die Dokumentation für das Open XML Format SDK 1.0 für Microsoft .NET Framework 3.0 in Zusammenarbeit mit Office 2007 steht auf [http://msdn.microsoft.com/en-us/library/bb448854\(office.12\).aspx](http://msdn.microsoft.com/en-us/library/bb448854(office.12).aspx) bereit.

Der Namensraum System.IO.Packaging, welcher mit ZIP-Dateien arbeitet, ist Bestandteil von Microsoft .NET Framework 3.0 und später. Dokumentation für die Funktionalität von .NET Framework 4.0 sowie Links für frühere Versionen finden Sie auf <http://msdn.microsoft.com/en-us/library/system.io.packaging.aspx>.

Diskussionsforen finden Sie auf <http://www.OpenXMLDeveloper.org> sowie <http://social.msdn.microsoft.com/Forums/en-US/oxmlsdk/threads>.

Word-VBA und die Arbeit mit Open XML

Nur weil VBA ein Teil von Word (und Office) ist, bedeutet das nicht, dass es auf der Arbeit mit dem eigenen Objektmodell begrenzt ist. VBA basiert auf der klassischen Visual Basic und ist eine allgemein einsetzbare, prozedurale Programmiersprache. Es kann mit Dateien, Ordnern und sogar der Windows API umgehen, wie dies in mehreren Kapiteln dieses Buchs beschrieben ist. Damit kann auch mit Open XML-Dateien gearbeitet werden.

Einziger Stolperstein ist die Tatsache, dass Microsoft keine entsprechenden Werkzeuge in der Sprache eingebaut hat. Die benötigte Bibliothek muss der Entwickler in das Projekt einbinden und allenfalls ergänzen.

HINWEIS

Wie ein Verweis auf eine Bibliothek gesetzt wird, wurde in Kapitel 9 beschrieben.

Für die Arbeit mit XML stellt Microsoft die Bibliothek MSXML zur Verfügung. Diese bietet Funktionalität für das Laden, Speichern und Navigieren von XML-Dokumenten und unterstützt Xpath

Version 1.0 sowie XSLT Version 1.0. Die Beispiele in diesem Kapitel wurden mit MSXML Version 6.0 erstellt.

Leider gibt es kein Gegenstück für die Arbeit mit ZIP-Paketen. In Microsoft-Produkten wird ein solches Werkzeug einzig in .NET Framework angeboten. Das Betriebssystem Windows 7 wird mit einer Packaging API ausgeliefert, welches Windows Vista ebenfalls zur Verfügung steht, wenn das »Platform Update for Windows Vista« installiert wurde. Leider stellt diese COM-Bibliothek keine Automatisierungsschnittstelle zur Verfügung, die einen Einsatz mittels VBA ermöglichen würde.

Deshalb machen die VBA-Beispiele in diesem Kapitel vom Windows Scripting-Objekt und Windows Shell-Objekt Gebrauch, um

- eine Open XML-ZIP-Paket zu öffnen, analog wie eine *.docx*-Datei im Windows-Explorer als ZIP-Datei geöffnet werden kann
- einen Ordner mit seinen Dateien in eine Open XML-ZIP-Paket (*.docx*) zu packen

HINWEIS

Falls Sie vorhaben, intensiv mit dem Open XML-Format zu programmieren, wäre es empfehlenswert, in bessere Werkzeuge für die Arbeit mit ZIP-Dateien zu investieren. Verschiedene Hersteller von ZIP-Software für den Benutzer verkaufen APIs für ihre Anwendungen, wie beispielsweise WinZip.

Die Beispiele in den folgenden Abschnitten veranschaulichen, wie mit VBA Open XML-Dateien direkt bearbeitet werden können.

Beispiel: Seriendruck-Datenquelleninformationen abfragen und entfernen

Ein Seriendruck-Hauptdokument ist im Prinzip eine Vorlage, woraus mehrfache Kopien erstellt werden, mit variierendem Inhalt aus einer Datenquelle. Das klassische Beispiel ist ein Standardbrief, der an mehrere Empfänger gesandt wird.

Viele Organisationen benutzen die Seriendruckfunktionalität für die verschiedensten Aufgaben. Mit der Zeit sammeln sich eine Menge Seriendruck-Hauptdokumente an, in Benutzer- sowie freigegebenen, geteilten Ordnern.

Ein großes Problem für die Verwaltung dieser Dokumente stellt die Verbindung zur Datenquelle dar. Die genauen Verbindungsinformationen stehen nur dann zur Verfügung, wenn das Dokument erfolgreich geöffnet werden kann. Ist die Datenquelle nicht verfügbar, werden die Datenquelleninformationen beim Öffnen des Dokuments durch Word entfernt.

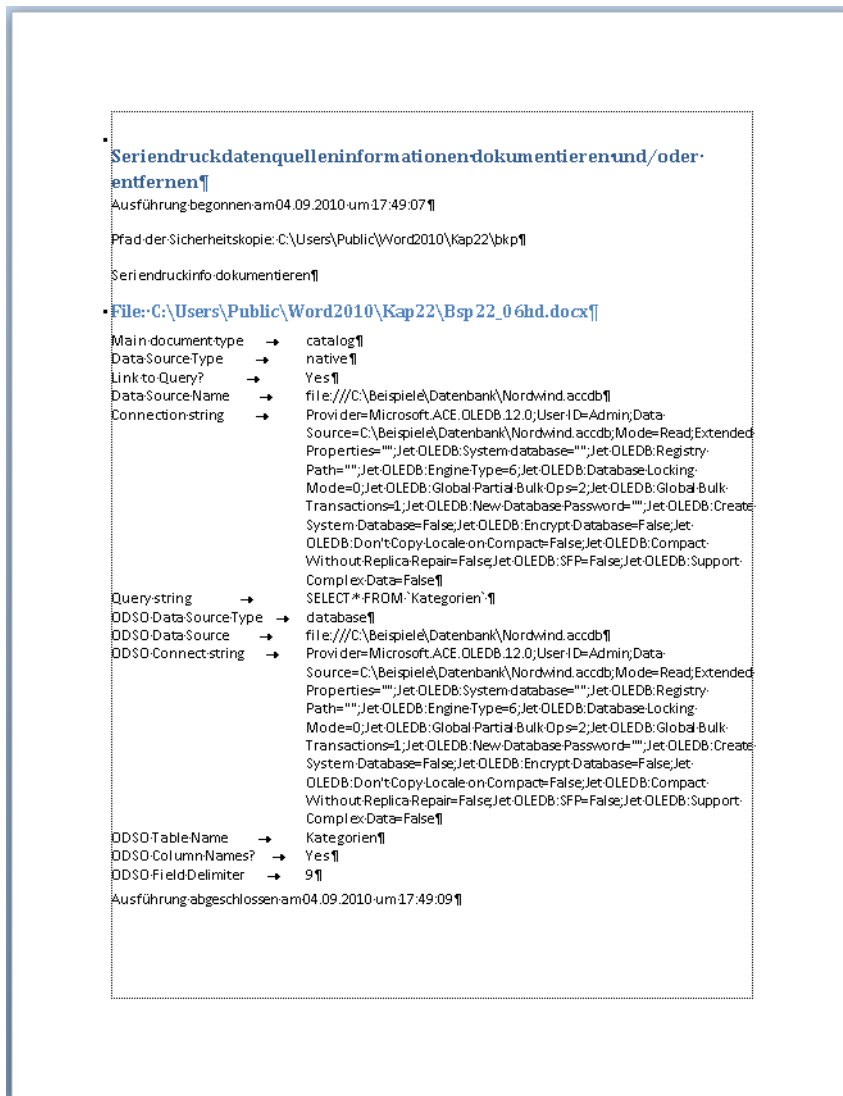
Des Weiteren wird die Datenquelle automatisch aus dem Seriendruck-Hauptdokument entfernt, wenn es programmtechnisch (statt durch den Benutzer) geöffnet wird. Der Code muss die gleiche Datenquelle wieder anhängen und dafür braucht es ebenfalls diese Informationen.

Das vorliegende Beispiel zeigt eine mögliche Lösung auf, indem es die Datenquelleninformationen aus dem Open XML-Dokument liest und dokumentiert (Abbildung 22.6). Es zeigt auf

- die Verfolgung einer Verkettung von Beziehungen und Teilen innerhalb eines Pakets
- das Extrahieren von Daten aus einem Paket
- das Entfernen von Teilen und die Aktualisierung zwischen Beziehungsteilen und dem Inhalt von *[Content_Types].xml*

Abbildg. 22.5

Bericht über die Seriendruckdatenquelleninformationen im Seriendruck-Hauptdokument



Wozu dient das Beispiel?

Neben der Dokumentation ermöglicht dieses Beispiel das Entfernen der Datenquelle aus dem Seriendruck-Hauptdokument. Der Code führt folgende Handlungen aus. Er

- macht eine Sicherheitskopie der ausgewählten *.docx*-Datei und speichert sie in einem dafür bestimmten Ordner
- packt diese *.docx*-Datei in einen Unterordner dieses Ordners aus
- navigiert durch die Strukturen der Teile und Beziehungen, bis er den Teil *settings* aufspürt, worin der Hauptteil der Datenquelleninformationen sich befindet

- falls diese Informationen gefunden wurden
 - sucht er den entsprechenden Teil mit den Beziehungen für den Teil *settings.xml*. Dieser könnte mehrere Beziehungen für den Seriendruck enthalten, die auf Teile für die externe Datenquelle und eventuell für eine Steuersatzdatei zeigen. Möglich ist auch eine Beziehung auf einen internen Teil *recipientData* (enthält Informationen über welche Empfänger der Benutzer ausgegrenzt oder eingebunden hat).
 - kann fakultativ die Dokumentation in das gegenwärtig aktives Dokument ausgeben
 - kann fakulatitv die Seriendruckinformationen aus dem Teil *settings*, aus den entsprechenden Beziehungen sowie aus dem Teil *recipientData* entfernen. Falls im Teil mit den Beziehungen für den Teil *settings* keine Beziehungen mehr vorliegen, wird auch der Teil für die Beziehungen gelöscht. Der Teil *[Content_Types].xml* wird entsprechend angepasst.
- packt das geänderte Dokument wieder ein und ersetzt das ursprüngliche Dokument

Das Beispiel ausführen

CD-ROM

Das Beispieldokument *Bsp22_06.docm* finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap22*. Sie brauchen zudem ein Seriendruck-Hauptdokument. Sie können ein eigenes Dokument oder die Datei *Bsp22_06hd.docx* im gleichen Ordner auf der CD-ROM verwenden. Dieses ist mit der Datenbank *Nordwind2007.accdb* als Datenquelle verbunden, die sich im Ordner *C:\Beispiele\Datenbank* befinden soll, wenn das Dokument ohne Verlust der Datenquelle geöffnet werden soll.

1. Öffnen Sie das Dokument *Bsp22_06.docm*.
2. Öffnen Sie den VBA-Editor (**Alt** + **F11**) und wechseln zum Modul *Bsp22_06*. (Dieses Modul ist der Einstiegspunkt für die Funktionalität und wird hier nicht wiedergegeben.)
3. Passen Sie die Pfadangaben in der Prozedur für Ihr System an und führen sie aus.

Ein Log sowie die Dokumentation zum Seriendruck wird in das aktuelle Dokument (*Bsp22_06.docm*) geschrieben. Falls Sie das Beispiel nochmals von vorne ausführen möchten, können Sie diesen Text einfach löschen.

Die weiteren Module, die im Folgenden vorgestellt werden, sind:

- *Bsp22_06_Main* enthält Prozeduren für
 - die Erstellung der Sicherheitskopie der angegebenen Datei
 - das Auspacken dieser Datei
 - das Navigieren und das Bearbeiten der Datei
- *clsContentTypesPart* ist ein Klassenmodul mit Funktionen für die Verwaltung des Teils *[Content_Types].xml*
- *modLog* enthält einige Hilfsfunktionen und -methoden für das Loggen von Informationen und Fehlern und wird hier nicht detailliert aufgeführt
- *modPackUnpack*: Beachten Sie dazu die Beschreibungen im Abschnitt »Beispiel: Einen *customUI*-Teil in ein Dokument einbinden« ab Seite 926

Das Hauptmodul: Bsp22_06

Deklaration der Konstanten

Dieser Abschnitt hält die URIs für die Beziehungsteile, sowie Text für Logs und Fehlermeldungen fest.

Zudem befinden sich Speicherort und Dateiname des Beziehungsteils für das Paket hier.

Letztlich wird eine globale Variable deklariert, der ein ContentTypes-Objekt zugewiesen wird. Dieses wird durch mehrere Prozeduren eingesetzt.

Listing 22.10 Hauptmodul in *Bsp22_06.docm* – die Konstante

```
Option Explicit

Const PACKAGEDESC As String = _
    "Package"
Const DATASOURCEREL As String = _
    "http://schemas.openxmlformats.org/officeDocument/2006/relationships/mailMergeSource"
Const DATASOURCEDESC As String = _
    "Data Source"
Const ODSODATASOURCEDESC As String = _
    "Data Source (ODSO)"
Const HEADERSOURCEREL As String = _
    "http://schemas.openxmlformats.org/officeDocument/2006/relationships/
    mailMergeHeaderSource"
Const HEADERSOURCEDESC As String = _
    "Header Source"
Const MAINDOCREL As String = _
    "http://schemas.openxmlformats.org/officeDocument/2006/relationships/officeDocument"
Const MAINDOCDESC As String = _
    "Main Document Part"
Const RECIPIENTDATAREL As String = _
    "http://schemas.openxmlformats.org/officeDocument/2006/relationships/recipientData"
Const RECIPIENTDATADESC As String = _
    "Recipient Data Part"
Const SETTINGSREL As String = _
    "http://schemas.openxmlformats.org/officeDocument/2006/relationships/settings"
Const SETTINGSDESC As String = _
    "Settings Part"

Const RELPARTPATH As String = "rels"
Const RELPARTEXTENSION As String = ".rels"
Dim objCTP As clsContentTypesPart
```

Die Prozedur *Process1OpenXMLFile*

Diese Prozedur hat vier Arbeitsgänge:

1. Sicherstellen, dass die Datei der Parameter *StrFullName* und die Pfadangabe der Parameter *strBackupPath* existieren und, dass die Sicherheitskopie nicht schon vorhanden ist.
2. Eine Sicherheitskopie der vorhandenen Datei erstellen und diese in einen Arbeitsordner auspacken.
3. Den Namen des Hauptdokumentteils ermitteln, den Namen seines Beziehungsteils bestimmen und den Teil *settings* mit den Seriendruckinformationen öffnen. Danach wird die Prozedur *processSettingsPart* aufgerufen, um einen Bericht zu erstellen bzw. die Seriendruckinformationen zu löschen, in Abhängigkeit der booleschen Parameter *bDocument* und *bRemove*.

4. Packt den Arbeitsordner ein und ersetzt die ursprüngliche Datei.

Falls Sie den Inhalt des Arbeitsordners anschauen möchten, kommentieren Sie die folgende Codezeile aus:

```
objFS0.DeleteFolder strPackagePath, True
```

Ein einzelnes FileSystemObject objFS0 wird erstellt und an alle Unterprozeduren übergeben. Mit diesem Objekt wird die allgemeine Dateiverwaltung ermöglicht, wie das Erstellen von Ordnerstrukturen oder die Prüfung von Dateien und Ordnern auf deren Vorhandensein.

Listing 22.11 Hauptmodul in *Bsp22_06.docm* – Hauptprozedur

```
Sub Process1OpenXMLFile( _
    strFullName As String, _
    strBackupPath As String, _
    bDocument As Boolean, _
    bRemove As Boolean _
)

    Dim bContinue As Boolean
    Dim objFS0 As Scripting.FileSystemObject
    Dim objXMLDoc As MSXML2.DOMDocument
    Dim strExtension As String
    Dim strPartFullName As String
    Dim strMainPartFullName As String
    Dim strPackagePath As String
    Dim strSettingsPartFullName As String
    Dim strTargetFullName As String
    Dim strFileName As String

    ' Phase 1

    bContinue = logHeader(strFullName)

    Set objFS0 = New FileSystemObject
    If Not objFS0.FileExists(strFullName) Then
        bContinue = logError("Die Datei konnte nicht gefunden oder nicht geöffnet werden.")
    End If

    If bContinue Then
        strExtension = objFS0.GetExtensionName(strFullName)
        Select Case UCase(strExtension)
            Case "DOCX", "DOCX", "DOTM", "DOTX"
                '
            Case Else
                bContinue = logError( _
                    "Die Dateierweiterung muß .docx, .docx, .dotm oder .dotx sein.")
        End Select
    End If

    If bContinue Then
        If Not objFS0.FolderExists(strBackupPath) Then
            bContinue = logError( _
                "Konnte den Ordner '" & _
                strBackupPath & "' für die Sicherheitskopie nicht finden.")
        End If
    End If
End Sub
```

Listing 22.11 Hauptmodul in *Bsp22_06.docm* – Hauptprozedur (Fortsetzung)

```

End If

If bContinue Then
    strTargetFullName = _
        objFSO.BuildPath( _
            strBackupPath, _
            objFSO.GetFileName(strFullName))
    If objFSO.FileExists(strTargetFullName) Then
        bContinue = logError( _
            "Die Datei ist schon im Ordner für die Sicherheitskopie vorhanden. " & _
            "Die Ausführung für diese Datei wurde abgebrochen.")
    End If
End If

' Phase 2

If bContinue Then
    objFSO.CopyFile
        Source:=strFullName, _
        Destination:=strTargetFullName, _
        overwritefiles:=True
    strPackagePath = _
        objFSO.BuildPath( _
            strBackupPath, _
            objFSO.GetBaseName(strFullName))
    bContinue = _
        logPack(modPackUnpack.unpack(strFullName, strPackagePath))
End If

' Phase 3

If bContinue Then
    Set objCTP = New clsContentTypesPart
    objCTP.openPart strPackagePath
    bContinue =
        getSingleTarget( _
            strPackagePath, _
            PACKAGEDESC,
            "[@Type='" & MAINDOCREL & "']", _
            MAINDOCDESC,
            strMainPartFullName, _
            objFSO, _
            BuildTargetFullName:=True, _
            RelPartShouldExist:=True, _
            RelShouldExist:=True, _
            RemoveRel:=False)
    If bContinue Then
        bContinue = _
            getSingleTarget( _
                strMainPartFullName, _
                MAINDOCDESC,
                "[@Type='" & SETTINGSREL & "']", _
                SETTINGSDESC,
                strSettingsPartFullName, _
                objFSO, _

```

Listing 22.11 Hauptmodul in *Bsp22_06.docm* – Hauptprozedur (Fortsetzung)

```

        BuildTargetFullName:=True, _
        RelsPartShouldExist:=False, _
        RelShouldExist:=False, _
        RemoveRel:=False)
    If bContinue Then
        If strSettingsPartFullName <> "" Then
            processSettingsPart
                strSettingsPartFullName, _
                bDocument, _
                bRemove, _
                objFSO
            End If
        End If
    End If
    objCTP.closePart
End If

' Phase 4

If bContinue Then
    bContinue = logPack(modPackUnpack.pack(strPackagePath, strFullName))
End If

If objFSO.FolderExists(strPackagePath) Then
    objFSO.DeleteFolder strPackagePath, True
End If

Set objFSO = Nothing
End Sub

```

Dokumenteigenschaften, Custom XML Parts und Inhaltssteuerelemente

Dieser Abschnitt bietet eine Übersicht der Benutzung und der Speicherorte von Dokumenteigenschaften in Word Open XML-Dokumenten an. Custom XML Parts und ihre Verwaltung in Open XML-Paketen werden zudem näher vorgestellt.

Dokumenteigenschaften speichern kleine Informationseinheiten über das Dokument.

- Ihr Inhalt kann oft mittels einer *DocProperty*-Feldfunktion innerhalb des Dokumenttexts angezeigt werden (beispielsweise der Name des Autors in einer Kopf- oder Fußzeile)
- Sie können gelesen und teilweise beschrieben werden, ohne das Objektmodell zu bemühen
- Sie werden oft für die Dokumentverwaltung benutzt

Dokumenteigenschaften vor der Ära Open XML

Vor Office 2007 gab es zwei Arten von Dokumenteigenschaften, die Office-eigenen und die benutzerdefinierten Eigenschaften.

Ungefähr 30 Office-eigene Eigenschaften werden bereit gestellt, wie Autor, Titel oder die Anzahl Wörter im Dokument. Einige davon wurden durch Word verwaltet. Andere konnten durch die Benutzerschnittstelle bzw. das Objektmodell festgelegt und bearbeitet werden. Der Inhalt vieler Eigenschaften kann wie erwähnt über *DocProperty*-Feldfunktionen im Dokument angezeigt werden. Benutzerdefinierte Dokumenteigenschaften werden durch den Benutzer oder den Entwickler erstellt. Ihren Inhalt konnte ebenfalls mit *DocProperty*-Feldfunktionen angezeigt, aber nicht aktualisiert werden. Ihnen wird ein bestimmter Datentyp (wie Zeichenkette, Zahl, Datum) zugewiesen. Der Wert der Eigenschaft konnte über das Objektmodell ausgelesen und beschrieben werden.

HINWEIS Die Anzeige einer Feldfunktion muss meistens ausdrücklich aktualisiert werden, um sicher zu stellen, dass sie die korrekte Information widerspiegelt.

Dokumenteigenschaften bei Open XML

Ab Office 2007, gleichzeitig mit der Einführung des Open XML-Dateiformats, gibt es sechs Objekte, die als »Eigenschaften« betrachtet werden können. Im Gegensatz zum alten binären Dateiformat ist es relativ einfach, mit der Hilfe von XML-Funktionalität wie Xpath, ein beliebiges Stück XML aus einem Dokument zu lesen. Diese Tatsache hat die Grenzen ein wenig verwischt. Jede Art von Objekt wird ein wenig anders gehandhabt, manchmal werden sogar verschiedene Eigenschaften der gleichen Art anders gehandhabt. Diese Unterschiede sorgen für etwas Verwirrung.

Die sechs Objektarten sind:

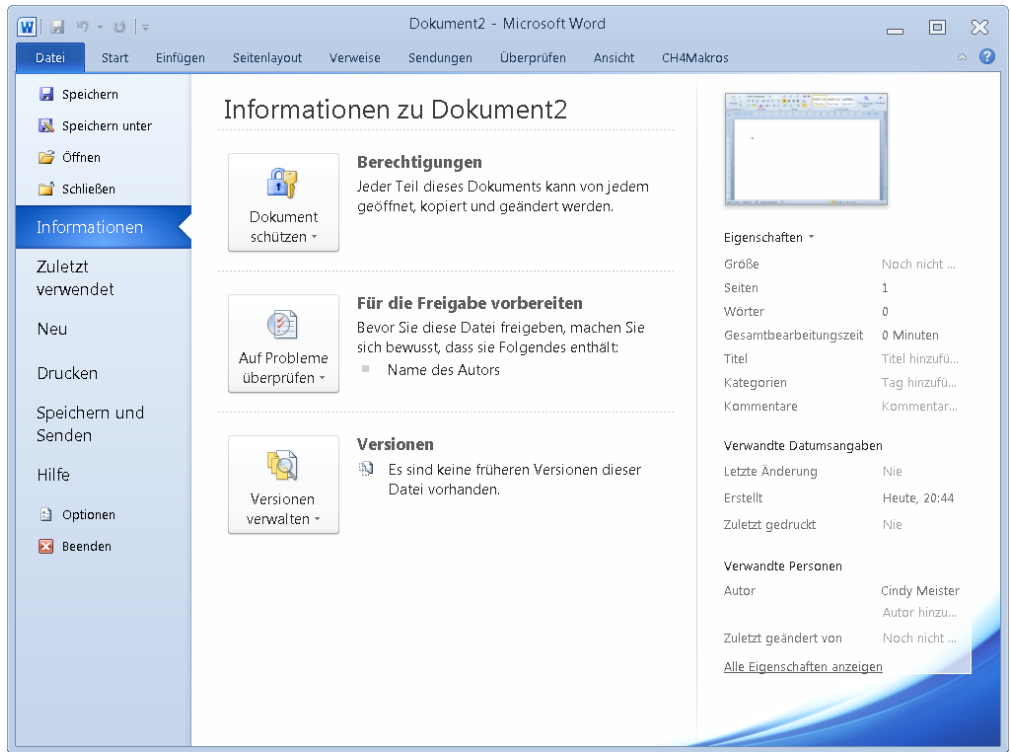
- Kerneigenschaften
- Erweiterte Eigenschaften
- Benutzerdefinierte Eigenschaften
- Deckblatteigenschaften
- »Server«-Eigenschaften
- Entwicklerdefinierte Eigenschaften, die in einem Custom XML Part gespeichert werden

Die Benutzerschnittstelle ermöglicht die Einsicht in und Bearbeitung von einigen Eigenschaftsarten. Office 2010 stellt dafür den Eigenschaftenbereich in der Registerkarte *Datei/Informationen* der Backstage-Ansicht zur Verfügung (Abbildung 22.6). In Office 2007 muß der Dokumentbereich über *Office-Schaltfläche/Vorbereiten/Dokumenteigenschaften* eingeblendet werden. Das Dialogfeld mit weiteren Eigenschaften wird durch den Menübefehl *Dokumenteigenschaften/Erweiterte Eigenschaften* eingeblendet. (Dieser Bereich sowie das Dialogfeld stehen auch in Word 2010 zur Verfügung und werden über das Dropdownmenü *Eigenschaften* in *Datei/Informationen* eingeblendet.)

Über *Einfügen/Text/Schnellbausteine/Dokumenteigenschaften* ist es möglich, Inhaltssteuerelemente einzufügen, die mit einer Dokumenteigenschaft verknüpft sind.

HINWEIS Mehr zum Thema Schnellbausteine sowie eine Abbildung des Menüs sind im Kapitel 6 beschrieben.

Abbildg. 22.6 Der Eigenschaftenbereich, rechts; im oberen Teil ist der Dokumentbereich im Dokumentfenster sichtbar



Kerneigenschaften

Diese entsprechen zum größten Teil den Office-eigenen Eigenschaften früherer Versionen. Sie sind im Open XML-Standard definiert. Wenn vorhanden, sind sie im Teil `/docProps/core.xml` gespeichert. (Word erstellt diesen Teil auch beim Speichern eines leeren Dokuments.)

Inhaltssteuerelemente für sieben dieser Kern-Eigenschaften stehen für das Anzeigen und die Aktualisierung ihrer Inhalte zur Verfügung: Autor, Betreff, Kategorie, Kommentare, Schlüsselwörter, Status sowie Titel. Dies sind die einzigen Dokumenteigenschaften, die standardmäßig im Dokumentbereich angezeigt werden und über den Menübefehl *Einfügen/Text/Schnellbausteine/Dokumenteigenschaften* als verknüpftes Inhaltssteuerelement eingefügt werden können.

Erweiterte Eigenschaften

Diese Eigenschaften sind spezifisch für ein Open XML-Dokument. Words erweiterte Eigenschaften sind nicht die gleichen wie jene von Excel oder PowerPoint. Diese Eigenschaften sind ebenso im Open XML-Standard definiert. Wenn vorhanden, werden erweiterte Eigenschaften im Teil `/docProps/app.xml` gespeichert. (Word erstellt diesen Teil auch beim Speichern eines leeren Dokuments.)

Inhaltssteuerelemente stehen zur Verfügung für das Anzeigen und die Aktualisierung von zwei dieser erweiterten Eigenschaften: Manager und Firma. Dies sind die einzigen erweiterten Eigenschaften,

die über den Menübefehl *Einfügen/Text/Schnellbausteine/Dokumenteigenschaften* als verknüpftes Inhaltssteuerelement eingefügt werden können.

Allgemein betrachtet, entsprechen die Kern- und die erweiterten Dokumenteigenschaften den Word-eigenen Eigenschaften aus früheren Versionen.

Benutzerdefinierte Eigenschaften

Die Benutzerdefinierten Eigenschaften entsprechen denjenigen aus früheren Versionen. Der Teil »Custom Document Properties« ist im Open XML-Standard definiert, es gibt jedoch keine Angaben über die Anzahl oder Namen der Eigenschaften, die darin gespeichert werden können. Wenn vorhanden, werden sie in `/docProps/custom.xml` gespeichert. Dieser Teil wird erst bei Bedarf von Word erstellt.

Es ist nicht möglich, benutzerdefinierte Eigenschaften mit einem Inhaltssteuerelement zu verknüpfen; es gibt dafür auch keinen Eintrag im Menü *Einfügen/Text/Schnellbausteine/Dokumenteigenschaften*. Der Inhalt kann jedoch mit einer *DocProperty*-Feldfunktion angezeigt werden.

Deckblatteigenschaften

Word 2007 führte das Konzept und die Schaltfläche *Deckblatt* ein. Damit wird am Dokumentanfang ein Deckblatt eingefügt, bestückt mit Feldern für die Eingabe von Informationen wie »Titel«, »Datum der Veröffentlichung« und »Kurzbeschreibung«. Im Ganzen sind es sechs deckblattspezifische Eigenschaften: Kurzfassung, Firmenadresse, Firmen-E-Mail-Adresse, Firmenfaxnummer, Firmentelefonnummer und Veröffentlichungsdatum. Verknüpfte Inhaltssteuerelemente stehen für alle diese Eigenschaften im Menü *Einfügen/Text/Schnellbausteine/Dokumenteigenschaften* zur Verfügung.

Word speichert deren Inhalt in einem gewöhnlichen Custom XML Part.

Ein solcher Teil dient als Beispiel für einen »wohl geformten« Custom XML Part. Word benutzt dafür bestimmte Schemas und arbeitet bewusst mit dem Teil.

»Server«-Eigenschaften

Diese Eigenschaften dienen der Dokumentverwaltung in einer Server-Umgebung wie SharePoint. Sie sind im Open XML-Standard *nicht* definiert.

Beim Öffnen eines auf SharePoint gespeicherten Dokuments erstellt oder aktualisiert SharePoint drei sich im Dokument befindende Custom XML Parts. Einer dieser Teile enthält den aktuellen Wert der Eigenschaften, ein anderer die Schema-Informationen. Die Eigenschaften erscheinen meistens in der Liste *Einfügen/Text/Schnellbausteine/Dokumenteigenschaften*. Es ist auch möglich, diese Eigenschaften in einem Sonderteil »Document-Eigenschaften – Server« des Dokumentbereichs zu integrieren. Sie werden in Word 2010 auch im Eigenschaftenbereich der Backstage-Ansicht angezeigt, außer sie sind Teil einer Dropdownliste. In diesem Fall erscheint ein Link, welcher sie im Dokumentbereich einblendet.

»Server«- wie Deckblatt-Eigenschaften werden in einem wohl geformten Custom XML Part gespeichert.

Erwähnenswert ist, dass

- neben den Kern-, erweiterten und Deckblatt-Eigenschaften diese Eigenschaften automatisch durch Word in den Eigenschaftenlisten, dem Dokumentbereich und dem Eigenschaftenbereich der Backstage-Ansicht eingebunden werden

- mit InfoPath der Dokumentbericht angepasst werden kann
- für diese Eigenschaftsart erweiterte Funktionalität zur Verfügung steht. Die Eigenschaft kann beispielsweise als Dropdownliste bezeichnet werden. In diesem Fall übernimmt die Liste ihren Inhalt aus einem mit der Eigenschaft verknüpften Schema. Sie wird im Dokumentbereich und als Inhaltssteuerelement als Dropdownliste dargestellt.

Für eine eingehendere Diskussion dieser Eigenschaften müssen Sie in der SharePoint-Dokumentation nachschlagen.

Entwicklerdefinierte Eigenschaften

Es handelt sich hier um keine Eigenschaften im herkömmlichen Sinne. Da sie aber ohne Einsatz des Objektmodells gelesen und geschrieben und zudem in einem mit einem Inhaltssteuerelement verknüpften Custom XML Part gespeichert werden, betrachten wir sie als Eigenschaften. Word führt sie in keiner Eigenschaftenliste und keinem -bereich auf und verknüpft sie nicht automatisch mit einem Inhaltssteuerelement. Diese Handlungen müssen alle durch den Entwickler vollzogen werden.

Kern-, erweiterte und Deckblatt-Eigenschaften vs. Entwicklerdefinierte Eigenschaften

Word besitzt einen Mechanismus, um Inhaltssteuerelemente mit allen diesen Eigenschaftstypen zu verknüpfen. Da dieser mit den Word-Objekten CustomXMLParts und CustomXMLPart arbeitet, muss Word die Speicherorte von Kern- und erweiterten Eigenschaften als Custom XML Parts behandeln.

Wenn Sie durch die im Word-Dokument vorhandenen CustomXMLParts schleifen, werden auch die Kern-, erweiterten und Deckblattteile aufgelistet. Dies wird durch Ausführung der Prozedur in Listing 22.12 veranschaulicht, wenn ActiveDocument ein leeres Dokument ist.

Listing 22.12 Den Inhalt jedes CustomXMLParts des aktuellen Dokuments ausgeben

```
Sub enumCustomXML()
    Dim objCXPM As Office.CustomXMLPrefixMappings
    Dim objCXPs As Office.CustomXMLParts
    Dim objCXP As Office.CustomXMLPart
    For Each objCXP In ActiveDocument.CustomXMLParts
        Debug.Print objCXP.XML
    Next
End Sub
```

Im Direktfenster (**Strg** + **G**) werden Zeilen aufgelistet, die wie folgt anfangen, wobei »Properties« die erweiterten Eigenschaften bedeuten:

```
<cp:coreProperties
<Properties
<CoverPageProperties
```

Beim Betrachten des XML für die Kern- und erweiterten Eigenschaften fällt auf, dass das von Listing 22.12 ausgegebene XML nicht genau übereinstimmt mit dem Inhalt der Paketeile *core.xml* bzw. *app.xml*. Das über das Objektmodell zur Verfügung gestellte XML enthält lediglich die Elemente, die mit einem Inhaltssteuerelement verknüpft werden können.

Im Vergleich wird das XML, das aus einem »echten« Custom XML Part stammt, vollumfänglich ausgegeben.

Sicherlich ein Grund dafür ist, dass ein Inhaltssteuerelement nicht mit einem Wert verknüpft werden soll, den Word selber verwaltet (beispielsweise das Erstellungsdatum oder die Anzahl Seiten). Dieser Mechanismus ist jedoch nicht hundertprozentig konsequent.

CD-ROM

Auf der CD-ROM zum Buch finden Sie im Ordner `\Beilagen\XMLPackages` die Excel-Arbeitsmappe *wordeigenschaften.xlsx*. Diese listet verschiedene Informationen für die Kern-, erweiterten und Deckblatteigenschaften auf, mit den für die Verknüpfung benötigten XPath-Angaben.

Beispiel: Custom XML Parts ersetzen ohne das Word-Objektmodell

Im Kapitel 7 haben Sie erfahren, wie Standardbriefe mit Daten aus einer Datenbank erstellt werden. Ein Beispiel veranschaulicht die programmatische Verknüpfung von Inhaltssteuerelementen mit einem CustomXMLPart und die Übertragung der Daten in diesen Teil – alles mit dem Word-Objektmodell und MSXML.

Abbildg. 22.7 Das Dokument mit Inhaltssteuerelementen, verknüpft mit einem Custom XML Part

The image shows a Word document template for a letter. It contains several placeholders for content that would be supplied by a Custom XML Part:

- Header:** A box containing "Northwind GmbH" and "Abteilung Verkauf" on the left, and "Tacomus, den 22. Juni 2010" on the right.
- Address:** A block of text: "Around the Horn", "Thomas Hardy", "120 Hanover Sq.", "WAI 1DP London", "Großbritannien".
- Subject:** A line starting with "Betreff: Betreff".
- Salutation:** A line starting with "Sehr geehrtes Fräulein Thomas Hardy".
- Text:** A line starting with "Den Briefinhalt hier einfügen".
- Signature:** A box containing the name "Andrew Fuller" and a blue rectangular placeholder for a signature or stamp.

In diesem Abschnitt zeigen wir eine alternative Methode auf, die sich *nicht* auf das Objektmodell von Word stützt. Stattdessen wird das Open XML-Dokument ausgepackt, die Daten aus der Nordwind2007.accdb-Datenbank direkt in den CustomXMLPart geschrieben und das Dokument wieder eingepackt. Obwohl der Code sich in einem VBA-Modul befindet und in Word läuft, könnte dieser Vorgang genauso gut in Access oder einer Server-Anwendung, ohne Vorhandensein der Word-Anwendung, ausgeführt werden.

Der Code in *Bsp22_07.docm* führt folgende Handlungen aus:

- Sammelt die Daten für alle Absender und Kunden aus der Nordwind2007.accdb-Datenbank
- Blendet ein Dialogfeld ein, in welchem der Benutzer den Absender und Empfänger sowie die bevorzugte Sprache für die Grußzeilen auswählt
- Packt den Standardbrief (*Bsp22_08.docm*) aus
- Schreibt die Daten für Absender und Kunden in den passenden Custom XML-Teil des Briefs
- Packt den Ordnerinhalt in ein neues Dokument (*Bsp22_09.docm*) ein
- Öffnet das erstellte Dokument, um die Demo fortzuführen

Beim Öffnen von *Bsp22_09.docm* wird ein Makro ausgeführt, das die Listen der Inhaltssteuer-elemente für Begrüßungen mit Auswahloptionen in der gewählten Sprache füllt. (Es wäre durchaus möglich gewesen, diese Listen bei der Bearbeitung des XMLs zu füllen, wir wollen aber auch die andere Methode aufzeigen.)

Das Beispiel ausführen

CD-ROM

Für dieses Beispiel benötigen Sie die Beispieldokumente *Bsp22_07.docm* und *Bsp22_08.docm*. Beide befinden sich auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap22*. Es setzt zudem die Datenbank *Nordwind2007.accdb* voraus.

Um das Beispiel vorzubereiten müssen Sie

- *Bsp22_07.docm* öffnen und den VB-Editor starten (**Alt** + **F11**)
- Das Modul *Bsp22_07_Datenbank* einblenden und die Pfadangaben in den Konstanten *sBriefDatei*, *sTargetBriefDatei* und *sArbeitsPfad* Ihrem System entsprechend anpassen
- Die Prozedur *Brief_Vorbereiten* im Modul *bsp22_07* ausführen
- Im eingeblendeten Dialogfeld einen Absender, einen Kunden und eine Sprache auswählen, dann auf **OK** klicken

Der Code führt die beschriebenen Handlungen aus und öffnet am Schluss das Dokument *Bsp22_09.docm*.

Das Hauptmodul: Bsp22_07.docm

Dieses Modul enthält einige Prozeduren, die in Listing 22.13 ersichtlich sind:

- *Brief_Vorbereiten* blendet das Dialogfeld ein und ruft beim Klicken der Schaltfläche **OK** die Prozedur *BriefInfoErstellen* auf

- Diese ist ähnlich wie in den bisherigen Beispielen aufgebaut: Sie stellt sicher, dass die benötigten Beispieldateien existieren. In diesem Fall wissen wir, dass der Custom XML Part bereits im Dokument vorhanden ist.
- Weiter ersetzt die Prozedur die Daten in den Elementen des XML (Alternativ könnte das XML im Speicher zusammengestellt und den vorhandenen Teil ersetzen.)

Die unterstützenden Prozeduren sind denen im Beispiel *Bsp07_03_CC.dotm* ähnlich und sind hier nicht aufgeführt.

Listing 22.13 Hauptmodul (Auszug) von *Bsp22_07.docm*

```
Option Explicit

Public Const sBriefDatei As String = "c:\Beispiele\Kap22\Bsp22_08.docm"
Public Const sTargetBriefDatei As String = "c:\Beispiele\Kap22\Bsp22_09.docm"
Public Const sArbeitPfad As String = "c:\Beispiele\Kap22\Bsp22_07"
Public Const sBriefXMLDatei As String = "/customXml/item2.xml"

Public Sub Brief_Vorbereiten()
    Dim frm As frmBsp22_07

    Set frm = New frmBsp22_07
    frm.Show
    If frm.Tag = "OK" Then
        BriefInfoErstellen frm
    End If
    Unload frm
    Set frm = Nothing
    Word.Documents.Open sTargetBriefDatei
End Sub

Private Sub BriefInfoErstellen( _
    frm As frmBsp22_07 _
)

    Dim bContinue As Boolean
    Dim objFSO As Scripting.FileSystemObject
    Dim objXMLDoc As MSXML2.DOMDocument60
    Dim objRS As ADODB.Recordset
    Dim strSQL As String
    Dim strBriefXmlDatei As String

    bContinue = True
    Set objFSO = New FileSystemObject
    If bContinue Then
        If Not objFSO.FileExists(sBriefDatei) Then
            bContinue = logError("Das Quelldokument konnte nicht gefunden " & _
                                "oder nicht geöffnet werden.")
        End If
    End If

    If bContinue Then
        If Not objFSO.FolderExists(sArbeitPfad) Then
            objFSO.CreateFolder sArbeitPfad
        End If
    End If
End Sub
```

Listing 22.13 Hauptmodul (Auszug) von *Bsp22_07.docm* (Fortsetzung)

```

If Not objFSO.FolderExists(sArbeitPfad) Then
    bContinue = logError("Der Pfad zum Arbeitsordner konnte nicht gefunden " & _
        "oder nicht erstellen werden.")
End If
End If

If bContinue Then
    bContinue = logPack(unpack(sBriefDatei, sArbeitPfad))
End If

If bContinue Then
    strBriefXmlDatei =
        objFSO.BuildPath(sArbeitPfad, sBriefXMLDatei)
    If objFSO.FileExists(strBriefXmlDatei) Then
        Set objXMLDoc = New MSXML2.DOMDocument60
        With objXMLDoc
            .validateOnParse = True
            If .Load(strBriefXmlDatei) Then
                'Die Absender-Informationen ermitteln und zuweisen
                With .SelectSingleNode("//absender").Attributes
                    .getNamedItem("name").Text = frm.cboAbsender.Text
                    .getNamedItem("id").Text = frm.cboAbsender.Value
                strSQL =
                    " SELECT Ort" &
                    " FROM   Persona" &
                    " WHERE  [Personal-Nr] = " & frm.cboAbsender.Value
                Set objRS = ADODatenHolen(strSQL)
                .getNamedItem("ort").Text = RSvalue(objRS, "Ort")
                Set objRS = Nothing
            End With

            'Die Kunden-Informationen ermitteln und zuweisen
            strSQL =
                " SELECT *" &
                " FROM   Kunden" &
                " WHERE  [Kunden-Code] = '" & _
                    frm.cboKunde.Value & "'"
            Set objRS = ADODatenHolen(strSQL)
            .SelectSingleNode("//kunde/kunden-nr").Text = _
                frm.cboKunde.Value
            .SelectSingleNode("//kunde/firma").Text = _
                RSvalue(objRS, "Firma")
            .SelectSingleNode("//kunde/kontakt").Text = _
                RSvalue(objRS, "Kontaktperson")
            .SelectSingleNode("//kunde/strasse").Text = _
                RSvalue(objRS, "Straße")
            .SelectSingleNode("//kunde/ort").Text = _
                RSvalue(objRS, "Ort")
            .SelectSingleNode("//kunde/plz").Text = _
                RSvalue(objRS, "PLZ")
            .SelectSingleNode("//kunde/land").Text = _
                RSvalue(objRS, "Land")

            .SelectSingleNode("//bestellung/sprache").Text = _
                AusgewählteSprache(frm)
        End With
    End If
End If

```

Listing 22.13 Hauptmodul (Auszug) von *Bsp22_07.docm* (Fortsetzung)

```
'Das vom Benutzer angegebene Datum in das XML-Dokument schreiben
.SelectSingleNode("//datum").Text = Format(Date, "d. MMMM yyyy")

.Save strBriefXmlDatei
Else
    bContinue = logError("Could not open the expected Custom XML Part")
End If
End With
Set objXMLDoc = Nothing
Else
    bContinue = logError("Der erwartete Custom XML Part konnte nicht geöffnet werden.")
End If
bContinue = logPack(pack(sArbeitPfad, sTargetBriefDatei))
End If
End Sub
```

Das Hauptmodul Bsp22_08/09.docm

Beim Öffnen des neuen Dokuments *Bsp22_09.docm* wird das Ereignis `Document_Open` angestoßen (Listing 22.14).

Listing 22.14 Das Ereignis *Document_Open*

```
Private Sub Document_Open()
    Brief_Vorbereiten
End Sub
```

Dieses ruft die Prozedur `BriefVorbereiten` auf, die folgende Aufgaben durchführt:

- Code für die Sprache aus dem Custom XML Part einlesen
- Anhand des Codes Zusammenstellung einer Liste von Begrüßungsformeln in der gewählten Sprache. Diese Information ist, im Gegensatz zum Beispiel des Kapitel 7, innerhalb des Dokuments, in weiteren Custom XML Parts gespeichert.
- Die Inhaltssteuerelemente, wie im Beispiel des Kapitels 7, gruppieren und sperren

Als der Besitzer (Entwickler und Ersteller des Dokuments und der Custom XML Parts) kennen wir die ID-Werte der Custom XML Parts. Wir können diese anstelle des Namensraumes einsetzen, um die Custom XML Parts direkt anzusprechen. Die Konstante mit diesen ID-Werten sind am Anfang der Prozedur in Listing 22.15 ersichtlich.

Listing 22.15 Bsp22_08/09 – Main Module

```
Option Explicit

Const sBriefXmlId As String = "{AF41273F-F24B-4C4A-BA90-186D44611319}"
Const sAnredeXmlId As String = "{163C49DF-ED09-4F91-A5B6-D4A4158F07C4}"
Const sGruessXmlId As String = "{63B0593E-64F5-43F8-96D8-0A49080A59CA}"

Public Sub Brief_Vorbereiten()
    Dim cc As Word.ContentControl
    Dim ccGruppe As Word.ContentControl
    Dim doc As Word.Document
```

Listing 22.15 Bsp22_08/09 – Main Module (Fortsetzung)

```

Dim objCXP As office.CustomXMLPart
Dim rng As Word.Range
Dim strSprache As String

Set doc = ActiveDocument
Set objCXP = doc.CustomXMLParts.SelectByID(sBriefXmlId)
strSprache = objCXP.SelectSingleNode("/bestellung/sprache").Text
Set objCXP = Nothing
'Inhaltssteuerelemente fertig vorbereiten
ListeFüllen doc, "Anrede", "anrede", sAnredeXmlId, strSprache
ListeFüllen doc, "Grußformel", "begruessung", sGruessXmlId, strSprache

'Die Inhaltssteuerelemente gegen das Löschen schützen
For Each cc In doc.ContentControls
    If cc.Title <> "Unterschriftsgrafik" Then
        cc.LockContentControl = True
    End If
    If cc.Title = "Beschreibung" Or cc.Title = "Artikel-Nr" Then
        cc.LockContents = True
    End If
Next

'Die Inhaltssteuerelemente gruppieren, um das Dokument zu schützen
Set rng = doc.Content
rng.MoveEnd Unit:=wdCharacter, Count:=-1
rng.Select
Set ccGruppe = doc.ContentControls.Add( _
    Type:=wdContentControlGroup, _
    Range:=Selection.Range)
ccGruppe.LockContentControl = False
doc.Bookmarks("BriefInhalt").Range.ContentControls(1).Range.Select
Set ccGruppe = Nothing
Set rng = Nothing
Set doc = Nothing
End Sub

Private Sub ListeFüllen( _
    doc As Word.Document, _
    strCCTitle As String, _
    strElement As String, _
    strXmlID As String, _
    strSprache As String)

Dim cc As Word.ContentControl
Dim objCXP As office.CustomXMLPart
Dim objNodes As office.CustomXMLNodes
Dim objNode As office.CustomXMLNode
Set objCXP = doc.CustomXMLParts.SelectByID(strXmlID)
Set objNodes = objCXP.SelectNodes("//" & strSprache & "/" & strElement)
For Each cc In doc.SelectContentControlsByTitle(strCCTitle)
    cc.DropDownListEntries.Clear
    For Each objNode In objNodes
        cc.DropDownListEntries.Add objNode.Text
    Next
Next
Next

```


Listing 22.15 Bsp22_08/09 – Main Module (Fortsetzung)

```
Set objNode = Nothing
Set objNodes = Nothing
Set objCXP = Nothing
End Sub
```

Open XML-Dokumente & XSLT

Die XSLT (Transformationen) wurde im Kapitel 21 kurz vorgestellt. Es handelt sich um eine kräftige, nicht prozedurale Sprache, die den Inhalt eines XML-Dokuments in eine Vielzahl andersartige Dokumentformate umwandeln kann. Oft wird XSLT dazu benutzt, um ein XML-Quelldokument in eine HTML-Webseite umzuwandeln. Enthält das XML-Dokument mehrere Einträge einer bestimmten Art (anders ausgedruckt, einer Liste gleichartiger Elemente), können diese beispielsweise in eine HTML-Tabelle ausgegeben werden. Vereinfacht gesehen besteht das XSLT aus zwei Umwandlungsmustern:

- Muster 1 erstellt das HTML für die gesamte Webseite, inklusiv die Struktur-Elemente für die Tabelle und ihre Kopfzeile
- Muster 2 generiert je eine Tabellenzeile pro Eintrag einer Liste von gleichartigen Elementen

Eine Anpassung dieser Muster ermöglicht die Erstellung eines Word Open XML-Dokument. In diesem Fall:

- Muster 1 erstellt das WordProcessingML für das Word-Dokument, unter anderem mit den Element-Strukturen für eine Tabelle und ihre Kopfzeile
- Muster 2 generiert in WordProcessingML eine Tabellenreihe pro Eintrag einer Liste von gleichartigen Elementen

Wenn Sie jemals mit der Seriendruckfunktionalität gearbeitet haben, wird Ihnen diese Idee bekannt vorkommen. XSLT ist in der Tat eine Methode, ein 1:n Resultat zu erzielen, wie beim Seriendruck. Sie ist vor allem nützlich auf einem Server-System, wo die XML-Daten aus einer SQL-Datenquelle stammen.

Überlegungen zur Arbeit mit XSLT

Es müssen einige Punkte beachtet werden, wenn Sie Open XML-Dokument mit XSLT erstellen möchten:

- Sie müssen zuerst die XSLT-Sprache beherrschen. Falls Sie ausschließlich mit XML-Werkzeugen von Microsoft arbeiten, bedeutet dies XSLT 1.0 sowie Xpath 1.0. Diese haben gegenüber XSLT 2.0 und Xpath 2.0 eine begrenzte Funktionalität.
- Sie müssen eingehende Kenntnisse des WordProcessingML haben
- Die auszuführende Handlung soll genau analysiert werden, um die optimale Vorgehensweise zu ermitteln. Falls die Aufgabe daraus besteht, Text in das Hauptdokument einzufügen, wäre es sinnvoll, diesen Teil aus dem Paket zu lösen, dessen XML zu transformieren und den Teil im Paket anschließend zu ersetzen.

Muss zusätzlich der Inhalt weiterer Teile geändert werden (beispielsweise Kopf- und Fußzeilen), muss jeder Teil für sich transformiert werden. Oder das Paket könnte in ein flaches OPC-Format umwandelt werden, dieses transformiert und das Resultat wieder in ein standardisiertes Paket zurückgewandelt werden.

- Das Erstellen und die Formatierung jedes kleinen Teils des neuen Inhalts muss überlegt werden. Nehmen wir als Beispiel die Einbeziehung eines Datums in der Fußzeile. Wird es durch XSLT eingefügt? Kann das XSLT 1.0 überhaupt? Oder wäre es besser, das Datum durch eine Feldfunktion anzugeben? Wenn ja, kann garantiert werden, dass die Feldfunktion aktualisiert wird, sodass das Ergebnis sichtbar ist und ausgedruckt wird? Wie steht es mit der Formatierung des Datums? Wird die Transformation auf das Format TT.MM.JJJJ bestehen oder kann man es den Einstellungen des Benutzersystems überlassen? Falls man das Letztere machen möchte, wie ist vorzugehen? Vielleicht wäre eine Feldfunktion doch besser geeignet?
- Sie müssen wissen, was das Resultat enthalten soll. Falls Sie beispielsweise 1.000 Datensätze in ein Dokument ausgeben möchten, mit einem Abschnitt pro Datensatz, braucht es auch 1.000 verschiedene Kopf- und Fußzeilen? Vielleicht schon, wenn jede andere Informationen enthalten soll.
- Es muß sichergestellt werden, dass die zur Verfügung stehenden Werkzeuge fähig sind, mit der Datenmenge und der Komplexität der Aufgabe umzugehen. Sie können damit vielleicht ein Dokument mit 1.000 Abschnitten generieren, aber was ist, wenn es 1.000.000 Datensätze sind?
- Die benötigten Ressourcen für den Unterhalt einer auf XSLT basierenden Lösung müssen in Betracht gezogen werden

Eine letzte Bemerkung

Das Erstellen eines Word-Dokuments ist nur eine der möglichen Aufgaben, wofür sich XSLT eignet. Jedes Mal, wenn Sie XML generieren oder ändern müssen, stellt sich die Frage, wie am besten vorzugehen ist:

- Ist die Aufgabe am besten über die Programmierschnittstelle eines DOMDocument zu lösen?
- Oder wäre es besser, ein XSLT zu schneiden, um das benötigte XML zu generieren?

Die einfachere oder robustere Methode ergibt sich vom Fall zu Fall. Ebenfalls müssen die Kenntnisse, Erfahrung und Fähigkeiten des Entwicklers in Betracht gezogen werden. Jemand, der sich mit XSLT sehr gut auskennt, aber von der anzuwendenden Programmiersprache wenig Kenntnisse hat, wird beispielsweise die Aufgabe mittels XSLT schneller und zuverlässiger lösen können.

Beispiel

CD-ROM

Die Dateien für das Beispiel befinden sich im *Kap22_XSLT.zip* auf der CD-ROM im Ordner *\Beispiele\Kap22*. Alle Einzelheiten zum Beispiel können Sie dem englischen Text des Dokuments *Kap22_XSLT.doc* entnehmen.

Aus Platzgründen können wir kein Codebeispiel für den Einsatz von XSLT mit Open XML ins Buch miteinbeziehen. Es steht jedoch eines auf der CD-ROM zur Verfügung. Damit wird mit XSLT und Daten aus der Nordwind2007.accdb-Datenbank ein Bericht mit 1:n-Tabellen erstellt. Der Ausgangspunkt ist in Abbildung 22.8 ersichtlich und das Endresultat in Abbildung 22.9.

Abbildg. 22.9 Das durch XSLT erstellte 1:n-Ergebnis

NORDWIND
GmbH

Bericht von Beteiligungen durch Kunden

Firma: Alfrida Fintenkette
Funden-Code: ALFRKI

<u>Beleg-Nr.</u>	<u>Belegdatum</u>	<u>Belegart</u>	<u>Umsatzbezug</u>	<u>Umsatzmonat</u>	<u>Umsatzperiode</u>	<u>Umsatzart</u>
12082	12.08.1987	12082	Umsatz-Bezug	12.08.1987	12.08.1987	Umsatz-Einnahme
12083	22.10.1987	12083	Umsatz-Bezug	21.10.1987	12.08.1987	Umsatz-Einnahme
12084	12.10.1987	12084	Umsatz-Bezug	12.10.1987	12.08.1987	Umsatz-Einnahme
12085	12.10.1987	12085	Umsatz-Bezug	12.10.1987	12.08.1987	Umsatz-Einnahme
12086	12.10.1987	12086	Umsatz-Bezug	12.10.1987	12.08.1987	Umsatz-Einnahme
12087	12.10.1987	12087	Umsatz-Bezug	12.10.1987	12.08.1987	Umsatz-Einnahme
12088	12.10.1987	12088	Umsatz-Bezug	12.10.1987	12.08.1987	Umsatz-Einnahme
12089	12.10.1987	12089	Umsatz-Bezug	12.10.1987	12.08.1987	Umsatz-Einnahme
12090	12.10.1987	12090	Umsatz-Bezug	12.10.1987	12.08.1987	Umsatz-Einnahme
12091	12.10.1987	12091	Umsatz-Bezug	12.10.1987	12.08.1987	Umsatz-Einnahme
12092	12.10.1987	12092	Umsatz-Bezug	12.10.1987	12.08.1987	Umsatz-Einnahme
12093	12.10.1987	12093	Umsatz-Bezug	12.10.1987	12.08.1987	Umsatz-Einnahme
12094	12.10.1987	12094	Umsatz-Bezug	12.10.1987	12.08.1987	Umsatz-Einnahme
12095	12.10.1987	12095	Umsatz-Bezug	12.10.1987	12.08.1987	Umsatz-Einnahme
12096	12.10.1987	12096	Umsatz-Bezug	12.10.1987	12.08.1987	Umsatz-Einnahme
12097	12.10.1987	12097	Umsatz-Bezug	12.10.1987	12.08.1987	Umsatz-Einnahme
12098	12.10.1987	12098	Umsatz-Bezug	12.10.1987	12.08.1987	Umsatz-Einnahme
12099	12.10.1987	12099	Umsatz-Bezug	12.10.1987	12.08.1987	Umsatz-Einnahme
12100	12.10.1987	12100	Umsatz-Bezug	12.10.1987	12.08.1987	Umsatz-Einnahme
12101	12.10.1987	12101	Umsatz-Bezug	12.10.1987	12.08.1987	Umsatz-Einnahme
12102	12.10.1987	12102	Umsatz-Bezug	12.10.1987	12.08.1987	Umsatz-Einnahme
12103	12.10.1987	12103	Umsatz-Bezug	12.10.1987	12.08.1987	Umsatz-Einnahme
12104	12.10.1987	12104	Umsatz-Bezug	12.10.1987	12.08.1987	Umsatz-Einnahme
12105	12.10.1987	12105	Umsatz-Bezug	12.10.1987	12.08.1987	Umsatz-Einnahme
12106	12.10.1987	12106	Umsatz-Bezug	12.10.1987	12.08.1987	Umsatz-Einnahme
12107	12.10.1987	12107	Umsatz-Bezug	12.10.1987	12.08.1987	Umsatz-Einnahme
12108	12.10.1987	12108	Umsatz-Bezug	12.10.1987	12.08.1987	Umsatz-Einnahme
12109	12.10.1987	12109	Umsatz-Bezug	12.10.1987	12.08.1987	Umsatz-Einnahme
12110	12.10.1987	12110	Umsatz-Bezug	12.10.1987	12.08.1987	Umsatz-Einnahme
12111	12.10.1987	12111	Umsatz-Bezug	12.10.1987	12.08.1987	Umsatz-Einnahme
12112	12.10.1987	12112	Umsatz-Bezug	12.10.1987	12.08.1987	Umsatz-Einnahme
12113	12.10.1987	12113	Umsatz-Bezug	12.10.1987	12.08.1987	Umsatz-Einnahme
12114	12.10.1987	12114	Umsatz-Bezug	12.10.1987	12.08.1987	Umsatz-Einnahme
12115	12.10.1987	12115	Umsatz-Bezug	12.10.1987	12.08.1987	Umsatz-Einnahme
12116	12.10.1987	12116	Umsatz-Bezug	12.10.1987	12.08.1987	Umsatz-Einnahme
12117	12.10.1987	12117	Umsatz-Bezug	12.10.1987	12.08.1987	Umsatz-Einnahme
12118	12.10.1987	12118	Umsatz-Bezug	12.10.1987	12.08.1987	Umsatz-Einnahme
12119	12.10.1987	12119	Umsatz-Bezug	12.10.1987	12.08.1987	Umsatz-Einnahme
12120	12.10.1987	12120	Umsatz-Bezug	12.10.1987	12.08.1987	Umsatz-Einnahme

Firma: Ana Trullio Dependayord y hielados
Funden-Code: ANATRE

<u>Beleg-Nr.</u>	<u>Belegdatum</u>	<u>Belegart</u>	<u>Umsatzbezug</u>	<u>Umsatzmonat</u>	<u>Umsatzperiode</u>	<u>Umsatzart</u>
12094	26.06.1986	12094	Umsatz-Bezug	12.06.1986	12.06.1986	Umsatz-Einnahme
12095	26.06.1986	12095	Umsatz-Bezug	12.06.1986	12.06.1986	Umsatz-Einnahme
12096	26.06.1986	12096	Umsatz-Bezug	12.06.1986	12.06.1986	Umsatz-Einnahme
12097	26.06.1986	12097	Umsatz-Bezug	12.06.1986	12.06.1986	Umsatz-Einnahme
12098	26.06.1986	12098	Umsatz-Bezug	12.06.1986	12.06.1986	Umsatz-Einnahme
12099	26.06.1986	12099	Umsatz-Bezug	12.06.1986	12.06.1986	Umsatz-Einnahme
12100	26.06.1986	12100	Umsatz-Bezug	12.06.1986	12.06.1986	Umsatz-Einnahme
12101	26.06.1986	12101	Umsatz-Bezug	12.06.1986	12.06.1986	Umsatz-Einnahme
12102	26.06.1986	12102	Umsatz-Bezug	12.06.1986	12.06.1986	Umsatz-Einnahme
12103	26.06.1986	12103	Umsatz-Bezug	12.06.1986	12.06.1986	Umsatz-Einnahme
12104	26.06.1986	12104	Umsatz-Bezug	12.06.1986	12.06.1986	Umsatz-Einnahme
12105	26.06.1986	12105	Umsatz-Bezug	12.06.1986	12.06.1986	Umsatz-Einnahme
12106	26.06.1986	12106	Umsatz-Bezug	12.06.1986	12.06.1986	Umsatz-Einnahme
12107	26.06.1986	12107	Umsatz-Bezug	12.06.1986	12.06.1986	Umsatz-Einnahme
12108	26.06.1986	12108	Umsatz-Bezug	12.06.1986	12.06.1986	Umsatz-Einnahme
12109	26.06.1986	12109	Umsatz-Bezug	12.06.1986	12.06.1986	Umsatz-Einnahme
12110	26.06.1986	12110	Umsatz-Bezug	12.06.1986	12.06.1986	Umsatz-Einnahme
12111	26.06.1986	12111	Umsatz-Bezug	12.06.1986	12.06.1986	Umsatz-Einnahme
12112	26.06.1986	12112	Umsatz-Bezug	12.06.1986	12.06.1986	Umsatz-Einnahme
12113	26.06.1986	12113	Umsatz-Bezug	12.06.1986	12.06.1986	Umsatz-Einnahme
12114	26.06.1986	12114	Umsatz-Bezug	12.06.1986	12.06.1986	Umsatz-Einnahme
12115	26.06.1986	12115	Umsatz-Bezug	12.06.1986	12.06.1986	Umsatz-Einnahme
12116	26.06.1986	12116	Umsatz-Bezug	12.06.1986	12.06.1986	Umsatz-Einnahme
12117	26.06.1986	12117	Umsatz-Bezug	12.06.1986	12.06.1986	Umsatz-Einnahme
12118	26.06.1986	12118	Umsatz-Bezug	12.06.1986	12.06.1986	Umsatz-Einnahme
12119	26.06.1986	12119	Umsatz-Bezug	12.06.1986	12.06.1986	Umsatz-Einnahme
12120	26.06.1986	12120	Umsatz-Bezug	12.06.1986	12.06.1986	Umsatz-Einnahme

Firma: Antonio Moreno Zapatera
Funden-Code: ANTON

<u>Beleg-Nr.</u>	<u>Belegdatum</u>	<u>Belegart</u>	<u>Umsatzbezug</u>	<u>Umsatzmonat</u>	<u>Umsatzperiode</u>	<u>Umsatzart</u>
12088	27.11.1986	12088	Umsatz-Bezug	12.11.1986	12.11.1986	Umsatz-Einnahme
12089	27.11.1986	12089	Umsatz-Bezug	12.11.1986	12.11.1986	Umsatz-Einnahme
12090	27.11.1986	12090	Umsatz-Bezug	12.11.1986	12.11.1986	Umsatz-Einnahme
12091	27.11.1986	12091	Umsatz-Bezug	12.11.1986	12.11.1986	Umsatz-Einnahme
12092	27.11.1986	12092	Umsatz-Bezug	12.11.1986	12.11.1986	Umsatz-Einnahme
12093	27.11.1986	12093	Umsatz-Bezug	12.11.1986	12.11.1986	Umsatz-Einnahme
12094	27.11.1986	12094	Umsatz-Bezug	12.11.1986	12.11.1986	Umsatz-Einnahme
12095	27.11.1986	12095	Umsatz-Bezug	12.11.1986	12.11.1986	Umsatz-Einnahme
12096	27.11.1986	12096	Umsatz-Bezug	12.11.1986	12.11.1986	Umsatz-Einnahme
12097	27.11.1986	12097	Umsatz-Bezug	12.11.1986	12.11.1986	Umsatz-Einnahme
12098	27.11.1986	12098	Umsatz-Bezug	12.11.1986	12.11.1986	Umsatz-Einnahme
12099	27.11.1986	12099	Umsatz-Bezug	12.11.1986	12.11.1986	Umsatz-Einnahme
12100	27.11.1986	12100	Umsatz-Bezug	12.11.1986	12.11.1986	Umsatz-Einnahme
12101	27.11.1986	12101	Umsatz-Bezug	12.11.1986	12.11.1986	Umsatz-Einnahme
12102	27.11.1986	12102	Umsatz-Bezug	12.11.1986	12.11.1986	Umsatz-Einnahme
12103	27.11.1986	12103	Umsatz-Bezug	12.11.1986	12.11.1986	Umsatz-Einnahme
12104	27.11.1986	12104	Umsatz-Bezug	12.11.1986	12.11.1986	Umsatz-Einnahme
12105	27.11.1986	12105	Umsatz-Bezug	12.11.1986	12.11.1986	Umsatz-Einnahme
12106	27.11.1986	12106	Umsatz-Bezug	12.11.1986	12.11.1986	Umsatz-Einnahme
12107	27.11.1986	12107	Umsatz-Bezug	12.11.1986	12.11.1986	Umsatz-Einnahme
12108	27.11.1986	12108	Umsatz-Bezug	12.11.1986	12.11.1986	Umsatz-Einnahme
12109	27.11.1986	12109	Umsatz-Bezug	12.11.1986	12.11.1986	Umsatz-Einnahme
12110	27.11.1986	12110	Umsatz-Bezug	12.11.1986	12.11.1986	Umsatz-Einnahme
12111	27.11.1986	12111	Umsatz-Bezug	12.11.1986	12.11.1986	Umsatz-Einnahme
12112	27.11.1986	12112	Umsatz-Bezug	12.11.1986	12.11.1986	Umsatz-Einnahme
12113	27.11.1986	12113	Umsatz-Bezug	12.11.1986	12.11.1986	Umsatz-Einnahme
12114	27.11.1986	12114	Umsatz-Bezug	12.11.1986	12.11.1986	Umsatz-Einnahme
12115	27.11.1986	12115	Umsatz-Bezug	12.11.1986	12.11.1986	Umsatz-Einnahme
12116	27.11.1986	12116	Umsatz-Bezug	12.11.1986	12.11.1986	Umsatz-Einnahme
12117	27.11.1986	12117	Umsatz-Bezug	12.11.1986	12.11.1986	Umsatz-Einnahme
12118	27.11.1986	12118	Umsatz-Bezug	12.11.1986	12.11.1986	Umsatz-Einnahme
12119	27.11.1986	12119	Umsatz-Bezug	12.11.1986	12.11.1986	Umsatz-Einnahme
12120	27.11.1986	12120	Umsatz-Bezug	12.11.1986	12.11.1986	Umsatz-Einnahme

Firma: Around the Horn
Funden-Code: AROUT

<u>Beleg-Nr.</u>	<u>Belegdatum</u>	<u>Belegart</u>	<u>Umsatzbezug</u>	<u>Umsatzmonat</u>	<u>Umsatzperiode</u>	<u>Umsatzart</u>
12088	18.11.1986	12088	Umsatz-Bezug	12.11.1986	12.11.1986	Umsatz-Einnahme
12089	18.11.1986	12089	Umsatz-Bezug	12.11.1986	12.11.1986	Umsatz-Einnahme
12090	18.11.1986	12090	Umsatz-Bezug	12.11.1986	12.11.1986	Umsatz-Einnahme
12091	18.11.1986	12091	Umsatz-Bezug	12.11.1986	12.11.1986	Umsatz-Einnahme
12092	18.11.1986	12092	Umsatz-Bezug	12.11.1986	12.11.1986	Umsatz-Einnahme
12093	18.11.1986	12093	Umsatz-Bezug	12.11.1986	12.11.1986	Umsatz-Einnahme
12094	18.11.1986	12094	Umsatz-Bezug	12.11.1986	12.11.1986	Umsatz-Einnahme
12095	18.11.1986	12095	Umsatz-Bezug	12.11.1986	12.11.1986	Umsatz-Einnahme
12096	18.11.1986	12096	Umsatz-Bezug	12.11.1986	12.11.1986	Umsatz-Einnahme
12097	18.11.1986	12097	Umsatz-Bezug	12.11.1986	12.11.1986	Umsatz-Einnahme
12098	18.11.1986	12098	Umsatz-Bezug	12.11.1986	12.11.1986	Umsatz-Einnahme
12099	18.11.1986	12099	Umsatz-Bezug	12.11.1986	12.11.1986	Umsatz-Einnahme
12100	18.11.1986	12100	Umsatz-Bezug	12.11.1986	12.11.1986	Umsatz-Einnahme
12101	18.11.1986	12101	Umsatz-Bezug	12.11.1986	12.11.1986	Umsatz-Einnahme
12102	18.11.1986	12102	Umsatz-Bezug	12.11.1986	12.11.1986	Umsatz-Einnahme
12103	18.11.1986	12103	Umsatz-Bezug	12.11.1986	12.11.1986	Umsatz-Einnahme
12104	18.11.1986	12104	Umsatz-Bezug	12.11.1986	12.11.1986	Umsatz-Einnahme
12105	18.11.1986	12105	Umsatz-Bezug	12.11.1986	12.11.1986	Umsatz-Einnahme
12106	18.11.1986	12106	Umsatz-Bezug	12.11.1986	12.11.1986	Umsatz-Einnahme
12107	18.11.1986	12107	Umsatz-Bezug	12.11.1986	12.11.1986	Umsatz-Einnahme
12108	18.11.1986	12108	Umsatz-Bezug	12.11.1986	12.11.1986	Umsatz-Einnahme
12109	18.11.1986	12109	Umsatz-Bezug	12.11.1986	12.11.1986	Umsatz-Einnahme
12110	18.11.1986	12110	Umsatz-Bezug	12.11.1986	12.11.1986	Umsatz-Einnahme
12111	18.11.1986	12111	Umsatz-Bezug	12.11.1986	12.11.1986	Umsatz-Einnahme
12112	18.11.1986	12112	Umsatz-Bezug	12.11.1986	12.11.1986	Umsatz-Einnahme
12113	18.11.1986	12113	Umsatz-Bezug	12.11.1986	12.11.1986	Umsatz-Einnahme
12114	18.11.1986	12114	Umsatz-Bezug	12.11.1986	12.11.1986	Umsatz-Einnahme
12115	18.11.1986	12115	Umsatz			

Zusammenfassung

Im vorliegenden Kapitel haben wir einen kurzen Überblick des Office Open XML-Dateiformats und der Arbeit damit geboten:

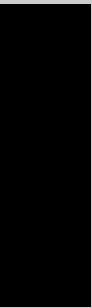
- Begonnen wurde das Kapitel mit einer Diskussion über die Ursprünge und Ziele des Office Open XML-Dateiformats (Seite 916 ff.).
- Als Nächstes haben wir gezeigt, wie mit einem Texteditor ein einfaches Open XML Word-Dokument im flachen OPC-Format erstellt wird. Die Grundkonzepte Pakete, Teile und Beziehungen wurden vorgestellt (Seite 917 ff.).
- Es folgte ein einfaches Beispiel, wie mit VBA (ohne Bezug des Word-Objektmodells) ein Teil in ein Paket eingefügt wird (Seite 926 ff.).
- Anschließend haben wir vertiefte Informationen über die zur Verfügung stehenden Typen von Teilen und Beziehungen präsentiert. Die Bestimmungen für die Benennungen von Teilen wurden vorgestellt sowie eine kurze Einführung in die Standarddokumentation (Seite 934 ff.).
- Im nächsten Abschnitt wurde über die programmtechnische Bearbeitung von Open XML sowie geeignete Methoden und Werkzeugen diskutiert. Die Vor- und Nachteile der Arbeit mit dem Objektmodell gegenüber der direkten Bearbeitung von Paketen und Teilen wurden erwägt (Seite 944 ff.).
- Ein Beispiel zum Navigieren durch die Teile und Beziehungen eines Pakets mit VBA folgte. Es zeigte auf, wie XML und Beziehungen entfernt werden. Das alles ohne Hinzuziehung des Objektmodells (Seite 948 ff.).
- Dokumenteigenschaften und ihr Zusammenhang mit Custom XML Parts war das Thema im nächsten Abschnitt. Diese Diskussion wurde abgerundet durch ein VBA-Beispiel (basierend auf *Bsp_07_03_CC.dotm*) für das Einfügen eines Custom XML Parts in ein Word-Dokument, ohne das Objektmodell zu bemühen (Seite 954 ff.).
- Das Kapitel wurde abschlossen mit einer kurzen Vorstellung der Nützlichkeit von XSLT bei der Arbeit mit Open XML-Dokumenten (Seite 965 ff.).

Teil F

Anhang

In diesem Teil:

Anhang A	Namenskonventionen für Word-VBA	971
Anhang B	Allgemeines zum Thema Feldfunktionen	975
Anhang C	Startoptionen von Word	983
Anhang D	Bei Problemen – Word zurücksetzen	987
Anhang E	Nützliche Links ins Internet	991
Anhang F	Begleitdateien	993



Anhang A

Namenskonventionen für Word-VBA

In diesem Anhang:

Variablen	972
Benutzerdefiniertes Dialogfeld	972
Entwicklungsumgebung	973
Wordobjekte	973

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 rng 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
Dokument-eigenschaft	prop	Dim prop As Office.DocumentProperty	Set prop = doc.BuiltInDocumentProperties(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)
Kopfzeile	hdr	Dim hdr As Word.HeaderFooter	Set hdr = doc.Sections(1).Headers(1)

Objekt	Präfix	Deklaration	Beispiel
Shape	shp	Dim shp As Word.Shape	Set shp = doc.Shapes(1)
Spalte	col	Dim col As Word.Column	Set col = tbl.Columns(1)
Tabelle	tbl	Dim tbl As Word.Table	Set tbl = doc.Tables(1)
Textmarke	bkm	Dim bkm As Word.Bookmark	Set bkm = doc.Bookmarks(1)
Zeile	row	Dim row As Word.Row	Set row = tbl.Rows(1)
Zelle	cel	Dim cel As Word.Cell	set cel = tbl.Cell(1,1)

Anhang B

Allgemeines zum Thema Feldfunktionen

Über die letzten Jahren gingen immer mehr Informationen in der Hilfe-Dokumentation zu den Feldfunktionen verloren. Ab Version 2007 ist, was übrig bleibt nur online verfügbar, entweder in Form einer Hilfe-Datei zu Word 2003 (<http://office.microsoft.com/de-de/results.aspx?qu=Feldfunktion>) oder in englischer Sprache (<http://office.microsoft.com/en-us/word/CH061047231033.aspx>).

Einige Feldfunktionen wurden im Buch näher erläutert. Hier werden nur einige Grundlagen für Word-Neulinge erläutert.



Feldfunktionen können via Menüband mit dem Befehl *Einfügen/Schnellbausteine/Feld* oder direkt ins Dokument eingefügt werden. Bei eingeblendetem Feldcode ist erkennbar, dass sich die Feldfunktion innerhalb eines geschweiften Klammernpaares 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 eingeblendet werden.

Tabelle B.1 Die wichtigsten Tastaturkombinationen für die Arbeit mit Feldfunktionen

Kürzel	Wirkung	VBA Äquivalent
[F9]	Markierte Feldcodes aktualisieren	<code>.Fields.Update</code>
[Strg] + [F9]	Feldklammern einfügen { }	<code>.Fields.Add</code>
[Alt] + [F9]	Alle Feldcodes ein- oder ausblenden	<code>.View.ShowFieldCodes</code>
[⇧] + [F9]	Feldcode der markierten Feldfunktion einblenden	<code>.Field.ShowCodes</code>
[⇧] + [Strg] + [F9]	Markierte Feldfunktionen in Text umwandeln	<code>.Fields.Unlink</code>
[Alt] + [⇧] + [F9]	GOTOBUTTON oder MACROBUTTON ausführen	
[F11]	Springt zur nächsten Feldfunktion im Dokument	<code>.NextField</code>
[⇧] + [F11]	Springt zur vorherigen Feldfunktion im Dokument	<code>.PreviousField</code>
[Strg] + [F11]	Markierte Feldfunktionen für die Aktualisierung sperren	<code>.Fields.Locked = True</code>
[⇧] + [Strg] + [F11]	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.docx" }.

Tabelle B.2 Allgemeine Formatschalter

Schalter	Wirkung	Beispiel
\#	Legt die numerische Abbildung eines Feldfunktionsergebnisses fest	{ = SUM(A1;A2) \# "0,00" } 10,00
\@	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, beispielsweise des Felds { PAGE } (Seitennummer), sodass die korrekte Seitennummer für die verknüpfte Datei statt der des aktuellen Dokuments erscheint	{ IncludeText "C:\\UnterKapitel\\UK_1.docx" bkmSeiten_Ref ! }
*	Legt das allgemeine Textformat fest	{ REF Textmarke * Upper } DER INHALT

Seit der Version 2000 verwendet Word die englischen Namen für Feldfunktionen und deren Schalter. In Tabelle B.3 finden Sie alle Feldfunktionen aufgelistet: links nach dem älteren, deutschen Ausdruck sortiert; rechts nach dem englischen.

Tabelle B.3 Gegenüberstellung der englischen und deutschen Feldfunktionsnamen

WdFieldType	Englischer Ausdruck	Alter deutscher Ausdruck
wdFieldExpression	=(Formula)	=(Ausdruck)
wdFieldAddressBlock	AddressBlock	(neu in Word 2002)
wdFieldAdvance	Advance	Versetzen
wdFieldAsk	Ask	Frage
wdFieldAutor	Author	Autor
wdFiledAutoNum	AutoNum	AutoNr
wdFiledAutoNumLegal	AutoNumLgl	AutoNrDez
wdFiledAutoNumOutline	AutoNumOut	AutoNumGli
wdFiledAutoText	AutoText	AutoText
wdFieldAutoTextList	AutoTextList	AutoTextListe
wdFieldComments	Comments	Kommentar
wdFieldCompare	Compare	Vergleich
wdFieldCreateDate	CreateDate	ErstellDat
wdFieldDatabase	Database	Datenbank
wdFieldDate	Date	AktualDat
wdFieldDocProperty	DocProperty	DokEigenschaft

Tabelle B.3 Gegenüberstellung der englischen und deutschen Feldfunktionsnamen (Fortsetzung)

wdFieldType	Englischer Ausdruck	Alter deutscher Ausdruck
wdFieldDocVariable	DocVariable	DokVariable
wdFieldEditTime	EditTime	(neu in Word 2002)
wdFieldFormula	EQ	Formel
wdFieldFileName	FileName	Dateiname
wdFieldFileSize	FileSize	Dateigröße
wdFieldFillin	Fillin	Eingeben
wdFieldGoToButton	GoToButton	Gehezu
wdFieldGreetingLine	GreetingLine	(neu in Word 2002)
wdFieldHyperlink	Hyperlink	Hyperlink
wdFieldIf	If	Wenn
wdFieldIncludePicture	IncludePicture	EinfügenGrafik
wdFieldIncludeText	IncludeText	EinfügenText
wdFieldIndex	Index	Index
wdFieldInfo	Info	Info
wdFieldKeyWord	Keywords	Stichwörter
wdFieldLastSavedBy	LastSavedBy	GespeichertVon
wdFieldLink	Link	Verknüpfung
wdFieldListNum	ListNum	ListenNr
wdFieldMacroButton	Macrobutton	MakroSchaltfläche
wdFieldMergeField	Mergefield	Seriendruckfeld
wdFieldMergeRec	MergeRec	Datensatz
wdFieldMergeSeq	MergeSeq	SeriendruckSeq
wdFieldNext	Next	Nächster
wdFieldNextIf	NextIf	Nwenn
wdFieldFootnoteRef	NoteRef	FußEndnoteRef
wdFieldNumChars	NumChars	AnzZeichen
wdFieldNumPages	NumPages	AnzSeiten
wdFieldNumWords	NumWords	AnzWörter
wdFieldPage	Page	Seite
wdFieldPageRef	PageRef	SeitenRef
wdFieldPrint	Print	Druck

Tabelle B.3 Gegenüberstellung der englischen und deutschen Feldfunktionsnamen (Fortsetzung)

wdFieldType	Englischer Ausdruck	Alter deutscher Ausdruck
wdFieldPrintDate	PrintDate	DruckDat
wdFieldQuote	Quote	Angeben
wdFieldRefDoc	RD	RD
wdFieldRef	Ref	Ref
wdFieldRevisionNum	RevNum	Überarbeitungsnummer
wdFieldSaveDate	SaveDate	SpeicherDat
wdFieldSection	Section	Abschnitt
wdFieldSequence	Seq	Seq
wdFieldSet	Set	Bestimmen
wdFieldSkipIf	SkipIf	Überspringen
wdFieldStyleRef	StyleRef	FVRef
wdFieldSubject	Subject	Thema
wdFieldSymbol	Symbol	SondZeichen
wdFieldTOCEntry	TC	Inhalt
wdFieldTemplate	Template	DokVorlage
wdFieldTime	Time	Zeit
wdFieldTitle	Title	Titel
wdFieldTOC	TOC	Verzeichnis
wdFieldUserAddress	UserAddress	BenutzerAdr
wdFieldUserInitials	UserInitials	Benutzerinitialen
wdFieldUserName	UserName	Benutzername
wdFieldIndexEntry	XE	XE
Veraltete Feldtypen, die aus Gründen der Rückwärtskompatibilität noch aufgeführt sind		
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

Tabelle B.3 Gegenüberstellung der englischen und deutschen Feldfunktionsnamen (Fortsetzung)

wdFieldType	Englischer Ausdruck	Alter deutscher Ausdruck
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)
wdFieldSubscriber		Diese Feldfunktion kommt nur in Word für Macintosh vor. Sie ist ähnlich der <i>IncludePicture</i> -Feldfunktion aus Word für Windows.
Feldfunktionen für die interne Verwaltung (Typ nur lesbar)		
wdFieldBidiOutline	Legt die Gliederungsrichtung auf rechts nach links	
wdFieldEmbed	Verwaltet ein eingebettetes Objekt wie ein Excel-Tabellenblatt	
wdFieldHTMLActiveX	Ein Steuerelement aus der Webtools-Symbolleiste	
wdFieldOCX	Ein ActiveX-Steuerelement, das über die Steuerelement-Toolbox eingefügt wurde.	
wdFieldFormCheckbox	Ein Formular-Kontrollkästchen	
wdFieldFormDropdown	Ein Formular-Dropdown	
wdFieldFormTextInput	Eine Formular-Textbox	
wdFieldPrivate	Word speichert Informationen für Dokumente, die aus einem anderen Format konvertiert wurden, um das Dokument wieder in das ursprüngliche Format zurückspeichern zu können, möglichst ohne Informationsverluste	
wdFieldShape	Eine AutoForm, die »mit Text in Zeile« formatiert ist	

Noch kritischer ist die Liste der Formatschalter in Tabelle B.4, weil Sie nur die deutschen Ausdrücke in den Hilfedateien von Word finden werden. Wenn Sie probieren, diese einzusetzen, liefern die Feldfunktionen Fehlermeldungen statt 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
Kleinbuchstaben	Lower

Tabelle B.4 Deutsche und englische Feldschalternamen (Fortsetzung)

Deutscher Name	Englischer Name
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
CardText	Grundtext
CharFormat	Zeichenformat
DollarText	Währungstext
FirstCap	SatzanfangGroß
Hex	Hex
Lower	Kleinbuchstaben
MergeFormat	Formatverbinden
Ordinal	OrdnungsZahl
OrdText	OrdText
roman	Römisch
Upper	Großbuchstaben

Anhang C

Startoptionen von Word

In diesem Anhang:

Die Befehlszeilenargumente von Word

984

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 C.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 AutoNew abgearbeitet noch das Ereignis DocumentNew 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 AutoNew abgearbeitet und das Ereignis DocumentNew 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 AutoExec -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.dotm</i> befinden. Es werden keine AutoExec -Makros ausgeführt. Das definierte Makro kann sich jedoch auch auf einen internen Word-Befehl beziehen.
<i>winword.exe /a</i>	Startet eine neue Instanz von Word, ohne die hinterlegten Add-In zu aktivieren
<i>winword.exe /safe</i>	Startet eine neue Instanz von Word im abgesicherten Modus. Dies entspricht dem Argument /a , 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 C.1 Zusammenfassung der Startoptionen von Word (Fortsetzung)

Syntax	Bedeutung
<code>winword.exe /l"Addinname"</code>	Startet eine neue Instanz von Word, zusätzlich wird das entsprechende Add-In geladen und aktiviert
<code>winword.exe /</code>	Startet eine neue Instanz von Word, wenn nur ein "/" oder eine unbekannte Befehlszeilenoption angegeben wird
<code>/c</code>	Startet eine neue Instanz von Word und lädt zusätzlich NetMeeting
<code>/x</code>	Startet Word aus der Umgebung des Betriebssystems heraus (beispielsweise, um ein Dokument aus dem Kontextmenü einer .dotx-Datei direkt auszudrucken). Kann nur über DDE gesteuert werden und akzeptiert lediglich einen einzigen DDE-Befehl.

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 Schalten /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 D

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, klicken Sie im Menüband den Befehl *Datei/Optionen/Speichern* an 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.dotm neu erzeugen



Normal.dotm

Bei einer unstabilen Word-Umgebung oder wenn einzelne Dokumente regelmäßig beschädigt werden, sollte die Standardvorlage *Normal.dotm* 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.dotm* führen. Dies wiederum kann Word regelmäßig zum Absturz bringen.

Wie ein neues Original der Standardvorlage *Normal.dotm* erzeugt wird, wurde bereits in Kapitel 13 ausführlich beschrieben.

HINWEIS

Bevor eine neue *Normal.dotm* 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.dotm*). Somit können zu einem späteren Zeitpunkt die in der *Normal.dotm* abgespeicherten Daten (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

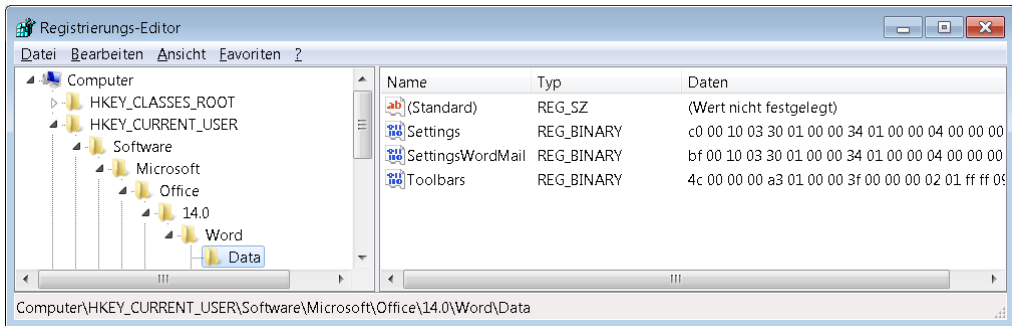
Word hinterlegt viele Einstellungen zu den gewählten Optionen und zur aktuellen Programmumgebung in der Windows-Registrierung. Diese Daten werden im Schlüssel *Data* in ein paar wenigen Einträgen mit entsprechend großen binären Datenfeldern abgespeichert.

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\14.0\Word\Data*. Markieren Sie diesen Ordner und wählen Sie den Menübefehl *Datei/Exportieren*, um eine Sicherheitskopie zu erstellen.

len. In einem zweiten Schritt wird der Schlüssel entfernt, dazu muss der Menübefehl *Bearbeiten/Löschen* angewählt werden.

Abbildg. D.1 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 E

Nützliche Links ins Internet

Auf dieser Seite sind die Adressen einiger interessanten Internet-Seiten zusammengefasst, die wir Autoren weiterempfehlen können. Diese Liste hat kein Anspruch auf Vollständigkeit.

<http://answers.microsoft.com/de-ch/default.aspx>

Die neue Community-basierte Support-Website von Microsoft für Windows und alle Office-Produkte

<http://www.microsoft.com/germany/msdn/library/office/default.mspix>

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 reichliche Informationen zur Verfügung gestellt, die zur Entwicklung von Word-Lösungen, auch basierend auf .NET, dienen.

[http://msdn.microsoft.com/en-us/library/bb726434\(v=office.12\).aspx](http://msdn.microsoft.com/en-us/library/bb726434(v=office.12).aspx)

Zusätzliche englische MSDN-Seiten für Microsoft Office Development zu allen Office-Versionen

<http://homepage.swissonline.ch/cindymeister>

Die englischsprachige Homepage von Cindy Meister, Autorin dieses Buchs 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

Die englischsprachige FAQ-Seite von MVP Bill Coan. Sie enthält Beiträge mit vertieftem Einblick zu verschiedenen Themen rund um Word.

<http://www.shaunakelly.com/word>

Die englischsprachige FAQ-Seite von MVP Shauna Kelly. Auch diese Seite enthält Beiträge mit vertieftem Einblick zu verschiedenen Themen rund um Word. Wer mit der automatischen Nummerierung von Word arbeiten muss oder möchte, soll unbedingt Shaunas Artikel lesen.

<http://www.aboutvb.de/vba/vba.htm>

Eine deutschsprachige Internet-Seite mit qualitativ hochwertigen Beiträgen, die sich an den professionellen Entwickler richten. Neben Beiträgen zu VBA stehen hier die Artikel zum klassischen Visual Basic sowie .NET und anderen Programmierthemen im Vordergrund.

<http://architag.com/xray/>

Architag stellt Werkzeuge zum Erstellen von *xml*-, *xsl*- sowie *xsd*-Dateien zur Verfügung

CD-ROM Sie finden zu jeder aufgeführten Internet-Adresse eine entsprechende *url*-Datei auf der CD-ROM zum Buch im Ordner *\Beispiele\AnhangF*.

Anhang F

Begleitdateien

In diesem Anhang:

Die Beispieldateien	994
Zusatz-Software: Die Dateien im Anhang	994

Die Beispieldateien

Alle im Buch dargestellten Beispiele finden Sie auf der CD-ROM zum Buch im Ordner *\Beispiele*.

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 Schreibschutzattribut 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.

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.

Zu folgenden Thema sind noch weitere Information auf der CD-ROM vorhanden:

- Die Datei *Dsofile.dll* bietet eine Schnittstelle, über die Dateieigenschaften aller Office-Dateien von außerhalb bearbeitet werden können. Die entsprechenden Informationen und die zugehörige *.dll*-Datei sind im Ordner *\Beilagen\Dsofile* abgelegt.
- Im Ordner *\Beilagen\Interne Word-Befehle* ist eine Zusammenstellung aller internen Word-Befehle (deutsch und englisch) abgespeichert
- Um die Eigenschaften von TrueType-Schriften im Windows-Explorer darzustellen, wird von Microsoft ein Add-In angeboten. Dieses Add-In steht im Ordner *\Beilagen\TrueTypeFont Eigenschaften* zur Verfügung.
- Die Hilfe zum WordBasic-Objekt sowie zu den internen Dialogfeldern von Word ist in der aktuellen Version der Onlinehilfe 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*.
- 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.
- Ein Dokument mit vertiefenden Informationen zum Thema »Suchen und Ersetzen« ist im Ordner *\Beilagen\SuchenErsetzen* gespeichert. Hier wird unter anderem die Funktionsweise von »Suchen mit Mustervergleich« genauer beschrieben.
- Die Anbindung von Word an Datenbanken kann auf verschiedene Arten erfolgen. Alle diese Möglichkeiten zu beschreiben, würde ein ganzes Buch füllen. Zusätzliche Informationen sind im Ordner *\Beilagen\Zusatzmaterial Seriendruck* abgelegt.

Verzeichnis zum Objektmodell

A

ActiveDocument 188
ActiveVBProject
 FileName-Eigenschaft 825
 Name-Eigenschaft 825
Add-In-Objekt
 Add-Methode 204, 518
 Delete-Methode 204, 518
Application-Objekt 176
 ActivePrinter 184, 201
 AutomationSecurity 179, 456, 594
 BackgroundPrintingStatus 197
 ButtonClicks 387
 CentimetersToPoints 326
 CustomizationContext 680
 DisplayAlerts 179
 Exists 614
 KeysBoundTo 788
 Language 180, 291
 MacroContainer 841
 MailingLabel 444, 445
 OnTime-Methode 854
 Options *siehe* Options-Objekt 180
 PathSeparator 108, 180, 191
 Quit 614
 RestrictLinkedStyles 293
 Run-Methode 204, 514, 838
 ScreenUpdating 178, 218
 UndoRecord 177
 Visible 178
 WordBasic *siehe* WordBasic 183
AttachedTemplate *siehe* Document-Objekt 191
AutomationSecurity *siehe* Application-Objekt 179
AutoTextEntry-Objekt
 Insert-Methode 229

B

BackgroundPrintingStatus *siehe* Application-Objekt
Bookmark-Objekt 342
 Add 343
 Exists 346
 Range.Text 346
BuildingBlockEntries-Objekt 273
 Add-Methode 273

BuildingBlocks-Objekt 274, 277
 BuildingBlockType-Objekt 274
 BuildingBlockTypes 277
 Category 273, 277
 Description 273
 Eigenschaften 273
 Insert-Methode 277
 LoadBuildingBlocks-Methode 269
 Name 273
 Type 273, 274
 Value 273
 wdBuildingBlockTypes-Konstanten 267
BuildingBlockTypes-Objekt 277
 wdBuildingBlockTypes-Konstanten 278

C

Collapse *siehe* Range-Objekt 209
CommandBar-Objekt 598, 799
 ExecuteMso 542
 Height 799
 Left 799
 Position 799
 RowIndex 799
 Top 799
 Visible 800
 Width 799
CommandBars-Objekt
 ExecuteMso-Methode 731, 754
 GetEnabledMso-Methode 731
 GetImageMso-Methode 731
 GetLabelMso-Methode 731
 GetPressedMso-Methode 731
 GetScreenTipMso-Methode 731
 GetSuperTipMso-Methode 731
 GetVisibleMso-Methode 731
ComputeStatistics *siehe* Document-Objekt 190
ContentControl-Objekt 402
 Add 423
 ContentControlAfterAdd-Ereignis 423
 ContentControlBeforeContentUpdate-Ereignis 423
 DateCalendarType 406
 DateDisplayFormat 406
 DateDisplayLocale 406
 DateStorageFormat 406
 DefaultTextStyle 405

ContentControl-Objekt (*Fortsetzung*)

- DropDownListEntries 406
- ID 419, 431
- LockContentControl 423
- LockcontentControl 405
- Lockcontents 405
- Multiline 406
- PlaceholderText 404
- SelectContentControlsByTag-Methode 405, 420
- SelectContentControlsByTitle-Methode 405, 420
- SetPlaceholderText-Methode 404
- ShowingPlaceholderText 404
- Temporary 406
- WdContentControlType 403

CustomDocumentProperties-Auflistung *siehe* Document-Objekt

CustomXMLPart-Objekt

- Add 418
- CustomXMLNode.AppendChildSubtree-Methode 431
- LoadXML 418
- NamespaceManager.LookupPrefix-Methode 430
- NodeAfterDelete-Ereignis 426
- NodeAfterInsert-Ereignis 426
- NodeAfterReplace-Ereignis 426
- SelectSingleNode 419
- SetMapping 419
- SetMappingByNode 419

CustomXMLParts-Auflistung

- PartAfterAdd 426
- PartAfterLoad 426
- PartBeforeDelete 426

D

DefaultFilePath *siehe* Options-Objekt 181

Designer-Objekt

- Controls 848
- Controls.Add-Methode 848

Dialogs-Objekt

- Display-Methode 709
- Execute-Methode 710
- Show-Methode 709
- Timeout-Parameter 709
- Update-Methode 710
- wdWordDialog-Konstanten 713

DisplayAlerts *siehe* Application-Objekt 179

Document-Objekt 188

- Add-Methode 189, 191, 202
- AttachedTemplate 191
- AutoFormatOverride 286
- ComputeStatistics 190
- DocumentProperty 670
- ExportAsFixedFormat-Methode 195
- FormattingShowClear-Methode 806

- FullName 191
- LockQuickStyleSet 287
- LockTheme 287
- MailMerge 434
- MailMerge.MailMergeType 434
- Name 191
- Open-Methode 189, 521
- Path 191
- PrintOut-Methode 196, 521
- SaveAs-Methode 193
- Saved 193
- Save-Methode 193
- StyleSort 808
- Type 190, 191
- Unprotect-Methode 398
- Variable 292, 673

E

Events 454

- ContentControlAfterAdd 423
- ContentControlBeforeContentUpdate 423
- ContentControlBeforeDelete 423
- ContentControlBeforeStoreUpdate 424
- ContentControlOnEnter 424
- ContentControlOnExit 424
- Document_Close 456
- Document_New 456
- Document_Open 456
- DocumentBeforeClose 473
- DocumentBeforePrint 475
- DocumentBeforeSave 474
- DocumentBeforeSync 477
- DocumentChange 472
- DocumentOpen 472
- NewDocument 471
- NodeAfterDelete 426
- NodeAfterInsert 426
- NodeAfterReplace 426
- PartAfterAdd 426
- PartAfterLoad 426
- PartBeforeDelete 426
- ProtectedViewWindowActivate 486
- ProtectedViewWindowBeforeClose 490
- ProtectedViewWindowBeforeEdit 488
- ProtectedViewWindowDeactivate 489
- ProtectedViewWindowOpen 487
- ProtectedViewWindowSize 491
- WindowActivate 464
- WindowBeforeDoubleClick 466
- WindowBeforeRightClick 467
- WindowDeactivate 465
- WindowSelectionChange 469
- WindowSize 465
- WithEvents 461

F

Field-Objekt 381
 Add-Methode 382
 Code 384
 DoClick 387
 Kind 381, 385
 Locked 381
 Result 384
 Unlink-Methode 381, 389
 Update-Method 381
 UpdateSource 266
FileDialog-Objekt 715
 DialogType-Eigenschaft 715
 Execute-Methode 717
 FileDialogFilters.Add-Methode 719
 FileDialogFilters.Clear-Methode 718
 FileDialogFilters-Auflistung 718
 Filters-Eigenschaft 718
 msoFileDialogFilePicker-Dialogfeld 723
 msoFileDialogFolderPicker-Dialogfeld 724
 msoFileDialogOpen-Dialogfeld 720
 msoFileDialogOpen-Konstanten 715
 msoFileDialogSaveAs-Dialogfeld 721
 Show-Methode 717
Find-Objekt
 ClearFormatting 222
 Execute-Methode 229
 Format 222, 288
 Forward 222, 223
 Found 227
 HitHighlight-Methode 223
 IgnorePunct 223
 IgnoreSpace 223
 MatchAllWordForms 222
 MatchCase 222
 MatchPrefix 222
 MatchSoundsLike 222
 MatchSuffix 223
 MatchWholeword 222
 MatchWildcards 222
 Replacement.Text 222
 Style 288, 350
 Text 222
 Wrap 222, 224
Formfield-Objekt 395
 EntryMacro 400
 ExitMacro 400
 Result 396
 Valid 396
Frames-Objekt
 Add 321

H

HeaderFooter-Objekt 244, 256
 Exists 259
 IsHeader, IsFooter 261
 LinkToPrevious 258
 PageNumbers 259

I

InlineShape-Objekt 308
 AddChart-Methode 617, 618
 AddOLEObject 607
 AddPicture-Method 309
 AddSmartArt 635
 HasSmartArt 637
 OLEFormat 607
 OLEFormat.DoVerb-Methode 607

K

KeyBinding-Objekt 781
 Add-Methode 790
 BuildKeyCode 783
 ClearAll-Methode 789
 Clear-Methode 790
 Count 781
 Disbale-Methode 790
 FindKey 783
 Key 783
 KeyString 784
 Rebind-Methode 791

L

Language *siehe* Application-Objekt 180
ListFormat-Objekt
 ListString 298
 ListValue 298
List-Objekt 298
 ConvertNumbersToText 298
 Count 298
 ListLevel 298
 RevmoveNumbers 298
ListTemplate-Objekt 300
 ListLevel 301
 Name 301
 OutlineNumbered 301

M

MailMerge-Objekt
 DataSource 435
 DataSource.ActiveRecord 435
 DataSource.Included 436
 DataSource.RecordCount 436
 DataSource.SetAllIncludedFlags-Methode 437
 Destination 441
 Execute-Methode 441
 FindRecord 438
 MailMergeWizardStateChange-Ereignis 440
 OpenDataSource-Methode 446
 ShowSendToCustom-Methode 440
 ShowWizard-Methode 440
 Mso-Konstantwert
 MsoBarPosition 799
 MsoLanguageID 180
 MsoOrgChartLayout 642
 MsoPresetTextEffect 629
 MsoPresetTextEffectShape 629
 MsoShapeType 637

N

NormalTemplate-Objekt 191, 203, 650

O

OperatingSystem *siehe* System-Objekt 174
 Options-Objekt 180
 DefaultFilePath 181
 PrintBackground 197

P

Page-Objekt 214
 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
 Paragraph-Objekt
 ListFormat 298
 PasteSpecial *siehe* Range-Objekt 313
 PathSeparator *siehe* Application-Objekt 191

PrintBackground *siehe* Options-Objekt 197
 PrintOut *siehe* Dokument-Objekt 196
 PrivateProfileString *siehe* System-Objekt
 ProtectedViewWindow-Objekt 484

R

Range-Objekt 207
 Calculate-Methode 218
 Collapse-Methode 209
 ConvertToTable-Methode 353
 Duplicate 216, 227
 ExportFragment-Methode 946
 Find 225
 Find *siehe* Find 222
 Font.Color 358
 Font.Name 358
 FormattedText 217, 350
 GoTo-Methode 214
 ImportFragment-Methode 946
 Information 214
 InRange-Methode 215, 385
 InsertAfter-Methode 210
 InsertBefore-Methode 210
 InsertBreak-Methode 210
 InsertParagraphAfter-Methode 210
 InsertParagraphBefore-Methode 210
 InsertXML-Methode 946
 InStory-Methode 215
 IsEqual 215
 MoveEnd-Methode 211
 MoveEndUntil-Methode 211
 MoveEndWhile-Methode 211
 Move-Methode 211
 MoveStart-Methode 211
 MoveStartUntil-Methode 211
 MoveStartWhile-Methode 211
 MoveUntil-Methode 211
 MoveWhile-Methode 211
 Paragraph.Border 359
 ParagraphFormat.FirstlineIndent 358
 PasteAndFormat-Methode 315
 PasteSpecial 313
 Select-Methode 207
 Shading 359
 Text 209
 TextRetrievalMode 216, 233
 WordOpenXML 946
 XML 946
 Reference-Objekt
 AddFromFile-Methode 857
 AddFromGUID-Methode 857
 Count 856
 IsBroken 860
 Remove-Methode 862

S

- Save *siehe* Document-Objekt 193
- SaveAs *siehe* Document-Objekt 193
- Saved *siehe* Document-Objekt 193
- SaveSetting 662
- Section-Objekt 242
 - ProtectedForForms 399
- SelectCurrent *siehe* Selection-Objekt 187
- Selection-Objekt 186
 - Find *siehe* Find 222
 - Move 187
 - SelectCurrent 187
 - Type 187
 - WdSelectionType 187
- Shape-Objekt 308
 - AddOLEObject 607
 - AddPicture-Method 309
 - AddSmartArt 635
 - AddTextEffect-Methode 628
 - HasSmartArt 637
 - Left 324
 - LockAnchor 324
 - Nodes.Points 336
 - OLEFormat 607
 - OLEFormat.DoVerb-Methode 607
 - RelativeHorizontalPosition 324
 - RelativeVerticalPosition 324
 - TextboxTightWrap 331
 - TextEffect 632
 - TextFrame 338
 - Top 324
 - WrapFormat 330
 - ZOrder-Methode 332
 - ZOrderPosition 333
- SmartArtColor-Objekt 638
- SmartArtLayout-Objekt 634
 - Category 635
 - Description 635
 - ID 635
 - Name 635
- SmartArtNode-Objekt 640
 - AddNode 641
 - Delete 641
 - Demote 641
 - Larger 641
 - Level 640, 641
 - OrgChartLayout 640, 641
 - ParentNode 641
 - Promote 641
 - ReorderDown 641
 - Reorder-Method 645
 - ReorderUp 641
 - Shape 641
 - Smaller 641
 - TextFrame2 640, 641
 - Type 641
- SmartArt-Objekt
 - AllNodes 637
 - Color 637
 - Layout 637
 - Nodes 637
 - QuickStyle 637
 - Reset 638
 - Reverse 638
- SmartArtQuickStyle-Objekt 639
- StoryRanges-Auflistung 233
 - NextStoryRange 245
- StoryRanges-Objekt 244
- Style-Objekt
 - Add-Methode 293
 - BuiltIn 290
 - Delete-Methode 288
 - Hidden 808
 - InUse 287
 - Locked 286
 - NameLocal 290
 - Priority 808
 - Type 293
 - UnhideWhenUsed 808
 - Visibility 807
- System-Objekt 173
 - CountryRegion 173
 - Cursor 173
 - LanguageDesignation 174
 - OperatingSystem 174
 - PrivateProfileString 174, 663
 - ProfileString 174
 - Version 174

T

- Table-Objekt 351
 - BottomPadding 363
 - Cell 353
 - Cell.Border 359
 - Cell.Split-Methode 372
 - Cell.VerticalAlignment 362
 - Cells.Merge-Methode 372
 - Column.Width 369
 - ConvertToText-Methode 374
 - FitText 363
 - ID 373
 - LeftPadding 363
 - NestingLevel 378
 - RightPadding 363
 - Row.AllowBreakAcrossPages 361
 - Row.Cells 353
 - Row.HeadingFormat 362

Table-Objekt (*Fortsetzung*)

- Row.Height 370
- Row.Index 353
- Row.LeftIndent 361
- Rows.Add-Methode 353
- Rows.Alignment 367
- Rows.HorizontalPosition 368
- Rows.RelativeHorizontalPosition 368
- Rows.RelativeVerticalPosition 368
- Rows.VerticalPosition 368
- Rows.WrapAroundText 368
- Shading 359
- TopPadding 363
- WordWrap 363

Template-Objekt 202

- BuildingBlockEntries 273
- OpenAsDocument-Methode 206
- Saved 204
- Type 205

Templates-Auflistung 204

TextEffectFormat-Objekt
632

ThisDocument-Modul 457

ThisDocument-Objekt 840

U

UndoRecord *siehe* Application-Objekt 177

UserForm-Objekt 685

- Activate 691
- Auflistung 693
- Caption 687, 700
- Deactivate 691
- Font 700
- Height 687, 700
- Hide 689
- Initialize 691
- Layout 691
- Left 687, 700
- Load 689
- Move 690
- Name 687
- QueryClose 692
- Repaint 691
- Show 689
- StartPosition 687
- Tag 688
- Terminate 691
- Top 687, 700
- Unload 689
- Width 687, 700

V

Variables-Auflistung *siehe* Document-Objekt

VB-Anweisung

- #Const 91, 507
- #If...Then 91
- Debug.Print 95
- Do-Schleife 88, 106, 227
- Enum 75, 116, 523
- For Each...Next 166
- For...Next 89, 166
- GetObjekt 102
- GetSetting 663
- If...Then...Else 85
- Is 215
- Kill 112
- On Error Goto 100
- On Error Resume Next 103
- Open 111
- Option Base 67
- Redim 67
- Redim Preserve 67
- Resume 101
- SaveSetting 662
- Select Case 77, 87, 102
- Set 162
- Type 71

VBComponent-Objekt

- Add-Methode 842
- CodeModule 829, 843
 - InsertLines-Methode 846
- CodeModule.AddFromFile-Methode 843
- CodeModule.AddFromString-Methode 844
- CodeModule.CountOfDeclarationLiens 831
- CodeModule.CountOfLines 830, 845
- CodeModule.CreateEventProc-Methode 845
- CodeModule.DeleteLines 839
- CodeModule.InsertLines-Methode 844
- CodeModule.Lines 830
- CodeModule.ProcBodyLine 833
- CodeModule.ProcCountLines 833
- CodeModule.ProcOfLine 832
- CodeModule.ProcStartLine 833
- CodeModule.ReplaceLine 837
- Count 827
- Designer *siehe* Designer
- Name 841
- Remove-Methode 850
- Type 827

VB-Datentyp

- Boolean 57
- Integer 58
- Long 58
- Object 59, 161
- Single 58

VB-Datentyp (*Fortsetzung*)

String 57
Variant 58

VBE-Objekt 822

ActiveVbProject 825
FileName 825
Name 825
Protection 824
VbProject.Import-Methode 840
VbProject.References *siehe* Reference-Objekt
VbProject.VbComponents *siehe* VbComponent-Objekt
VbProjects 824

VB-Ereignis

AutoMakros 454

VB-Funktion

CallByName 853
CBool 62
CInt 62
CLng 62
CSng 62
CStr 62
CVar 62
Date 84
DateDiff 524
Dir 92, 106, 123, 724
Environ 144
FileDateTime 113
FileLen 115
Format 84, 85
GetAttr 109, 110
GetPoint 215
InputBox 79
InStr 83
InstrRev 83
IsDate 86
IsNumeric 85
LBound 68
Left 82
Mid 82
MsgBox 80
OnTime 839
Replace 83, 520
Right 82, 108
Timer 147
UBound 68
UCase 86

VB-Funktionen

CreateObject 504

VB-Konstantwert

vbAlias 110
vbArchive 110
vbCR 83
vbDirectory 110, 111
vbHidden 110

vbNormal 110
vbObjectError 104
vbReadOnly 110
vbSystem 110
vbVolume 110

VB-Objekt

Err 103
UndoRecord 93

VB-Schlüsselwort

ByRef 64, 65
ByVal 63, 64
Nothing 61, 164
Optional 114
Private 59, 71, 75
Public 60
Step 168
WithEvents 461

Version *siehe* System-Objekt 174

View-Objekt

ShowTextBoundaries 250

Visible *siehe* Application-Objekt 178

W

Wd-Konstantwert

WdBreakType 211
WdBuildingBlockTypes 267, 274
WdBuiltinStyle 290
WdCollapseDirection 209
WdCursor 173
WdDefaultFilePath 182
WdDocPartInsertOptions 276
WdDocumentType 190
WdDontBreakWrappedTable 369
WdFieldKind 381
WdFieldType 977
WdFormatPDF 194
WdHeaderFooterIndex 257
WdInlineShapeType 637
WdKey 781
WdKeyCategory 781
WdMailMergeActiveRecord 435
WdMailMergeDestination 441
WdMailMergeMainDocType 434
WdMergeSubType 448
WdPaperTray 255
WdPasteDataType 313
WdPrintOutItem 200
WdPrintOutPages 200
WdPrintOutRange 198
WdRowHeightRule 370
WdSelectionType 187
WdShowFilter 807
WdStatistic 190

Wd-Konstantwert (*Fortsetzung*)

WdStoryType 245

WdTablePosition 368

WdTaskPane 796

WdTemplateType 206

WdTextboxTightWrap 331

wdTypeQuickParts-Konstante 274

WdUnits 211

wdWithinTable 214

wdLine *siehe* Selection-Objekt, Move 187

WdProtectedViewCloseReason-Enumeration 490

Word.Chart-Objekt

AddChart-Methode 617, 618

WordBasic-Objekt 183

CreateCommonFieldBlockFromSel 184

DisableAutoMacros 180, 183, 456

FileNameInfo 184

FilePrintSetup 184

MailMergePropagateLabel 445

SelectSimilarFormatting 183

SortArray 184

X

XMLNamespace-Objekt 912

XSLTransform-Objekt 912

Stichwortverzeichnis

- .NET
 - aufgezeichneten Code bearbeiten 31
 - Interop Assemblies 502
 - Konstantwerte qualifizieren 179
 - Konvertierung
 - von Hilfe-Beispielen 56
 - von VBA ByVal / ByRef 64
 - von VBA User Defined Types 72
 - von VBA Variant 59
 - von VBA-Ganzzahlen 58
 - von VBA-Zeichenketten 57
 - Late Binding 510
 - Option Strict 510
 - optionale Argumente 189
 - Umwandlung von VBA-Datentypen 63
 - Verweis
 - auf COM-Objekte 502
 - zur COM-Anwendung hinzufügen 40
- l:n (programmtechnisch) 421
- A**
 - Absatz
 - Einzug der ersten Zeile 358
 - mit Rahmen versehen 359
 - Abschnitt 242
 - Eigenschaften 242
 - Kopf- und Fußzeile 244
 - Abschnittsumbruch einfügen 210
 - Access *siehe* Fernsteuern
 - Add-In (COM) 533
 - Add-In (Vorlage)
 - darauf verweisen 523
 - Funktion aufrufen 523
 - Prozedur aufrufen 523
 - Prozeduren aufrufen
 - aus anderer Anwendung 526
 - UserForm aufrufen 524
 - Vorlage (*.dotm) 522
 - Add-In (VSTO) 533
 - anlegen 535
 - ausschalten 550
 - ComVisible 548
 - Custom Task Pane 537, 545
 - mit Benutzersteuerelement 545
 - Dokumente zur Laufzeit bestücken 550
 - GetCustomUI 539
 - im Debug-Modus starten 550
 - Menüband-Erweiterung zufügen 537
 - Menüband-XML 537
 - mit Tastenkombination verbinden 548
 - Verteilung 534
 - vom Rechner entfernen 550
 - ADO 596, 601, 618
 - Aktualisierung
 - von Feldfunktionen 381
 - Anforderungen an ein Dialogfeld 700
 - Ansicht festlegen 653
 - API 118
 - 64-Bit-Unterstützung 118
 - Declare-Anweisung 118
 - Deklaration und Aufbau 118
 - FindWindow 131
 - GetKeyState 150
 - GetPrivateProfileSection 139
 - GetPrivateProfileSectionNames 142
 - GetUserNameEx 144
 - PathAddBackslash 123
 - PathAddExtension 125
 - PathAppend 125
 - PathCombine 124
 - PtrSafe-Attribut 118
 - RegCloseKey 133
 - RegEnumKeyEx 132
 - RegOpenKey 132
 - SHBrowseForFolder 132
 - ShellExecute 127
 - Sleep 146
 - sndPlaySound 147
 - Array *siehe* Datenfeld
 - Aufgabenbereich
 - XML-Struktur 906
 - Aufgabenbereich *siehe* Task Pane 794
 - Auflistung
 - bearbeiten 166
 - Element
 - ansprechen 165
 - entfernen 167
 - wahlweise löschen 90
 - Index eines Objekts ermitteln 165
 - Aufzählungen 297
 - Ausgabe in Datei 201
 - Autoform
 - Standardwerte festlegen 656
 - AutoKorrektur
 - definieren 654

AutoMakros 180, 454

AutoClose 455

AutoExec 455

AutoExit 455

AutoNew 455, 587, 598

AutoOpen 455

und Makro-Sicherheit 456

unterbinden 456

Vergleich mit Ereignissen 457

AutoText

entfernen 660

B

Backstage-Ansicht 728

Bausteine 266

Building Blocks.dotx 267

laden erzwingen 269

BuildingBlockTypes 267

Built-In Building Blocks.dotx 267

laden erzwingen 269

neu erstellen 270

Dokumenteigenschaften, *siehe* Dokument,

Eigenschaften 266

Eigenschaften bearbeiten 272

Felder 272

Felder, *siehe* Feldfunktion, Bausteine 266

im Dokument einfügen 277

Katalog 267, 275

Katalog, benutzerdefiniert 278

Makrorekorder 274

mit Text und Grafik erstellen 274

Organizer 266, 279–280

Schnellbaustein erstellen 272

Schnellbaustein-Katalog 266

Auswahl speichern im 272

Schnellbaustein-Menü

Dokumenteigenschaften 270

Felder 272

Organizer 280

Befehlszeilenargumente 984

Benutzerdefinierte Dialogfelder

alle geladenen Objekte bearbeiten 693

als Argument an Prozedur übergeben 694

Anforderungen des Anwenders 700

anzeigen 689

benennen 687

Daten in Xml-Datei 707

effizient erstellen 702

Eigenschaften

in Dateien auslagern 705

in Ini-Datei 705

in Txt-Datei 706

zwischenspeichern 702

Benutzerdefinierte Dialogfelder (*Fortsetzung*)

entladen 689

Ereignisse 691

erstellen 685

Fensterhandle hWnd 131

Fenstertitel eintragen 687

Flackern am Bildschirm verhindern 691

hWnd 131

Klassenbezeichnung ThunderDFrame 131

laden 689

neu zeichnen 691

positionieren 687

schließen verhindern 692

Startposition festlegen 687

Steuerelement

Anzeige 696

Befehlsschaltfläche 696

Bezeichnungsfeld 695

Bildlaufleiste 696

Drehfeld 696

Kombinationsfeld 695

Kontrollkästchen 695

Listenfeld 695, 698

Multiseiten 696

Optionsfeld 695, 697

Rahmen 696

Register 696

Textfeld 695

Umschaltfeld 695

Steuerelement einbinden 695

verbergen 689

verschieben 690

wann tritt ein Ereignis ein 692

zur Laufzeit beeinflussen 702

Benutzernamen ermitteln (API) 144

Benutzerschnittstelle anpassen

Möglichkeiten 676

Speicherort 676

Speicherort (COM-Add-In) 681

Zielgruppe 678

Bereich 207

auf einen Punkt verkleinern 209

Bildschirmkoordinaten 215

erweitern 211

festlegen 207

formatieren 208

formatierter Text 217

Formel berechnen 218

in Tabelle umwandeln 353

lokalisieren 214

schattieren 359

Standort auf der Seite 215

Text hinzufügen 210

Text lesen 216

Text zuweisen 209

Bereich (*Fortsetzung*)
 und Feldcodes 216
 und verborgener Text 216
 vergleichen 215
 verkleinern 211
 verschieben 211
 Bilder einfügen als 309, 313
 Bildschirm
 Flattern vom 178
 unsichtbar machen 178
 Bitweiser Vergleich 153
 Bold 208
 BOM 872
 Browserobjekt 238
 Building Blocks, *siehe* Bausteine 267
 Built-In Building Blocks.dotx *siehe* Bausteine
 Byte Ordering Mark *siehe* BOM

C

C#
 Anwendung in Word 172
 Late Binding 185
 Version 4.0 19, 172
 Word-Eigenschaft mit Parameter 181
 Codezeilen umbrechen, lange 83
 Codierung 872
 Compiler-Anweisung 91
 Compiler-Konstante 91, 121, 507
 Vba6 122
 Vba7 121–122, 177
 Win32 121
 Win64 121
 Content Controls Toolkit 412

D

Datei
 Ausführung über Dateiendung (API) 127
 Dateisystem
 alle Dateien auflisten 106
 Datei löschen 112
 Dateidatum ermitteln 113
 Dateigröße ermitteln 115
 ist Datei gesperrt 111
 ist Datei vorhanden 109
 ist Verzeichnis vorhanden 110
 Ordnername mit Backslash ergänzen 107
 Platzhalter 107
 Datenbanken *siehe* Fernsteuern
 Datenfeld
 dimensionieren 67
 Größe ermitteln 68
 sortieren 69

Datentyp
 Aufzählung 75
 benutzerdefinierter Typ 71
 umwandeln 62
 Debuggen 94
 Debug.Print 95
 Deklarationsbereich 831
 Designer
 Die graphische Benutzeroberfläche 847
 Designer-Objekt 854
 Diagramm
 als OLE-Objekt 622
 Datentabelle 619
 einfügen 617
 Excel im Word-Dokument 617
 formatieren 619
 im Word-Dokument 616
 Dialogfelder
 anzeigbare Dialogfelder 714
 anzeigen 709
 Argumente für integrierte Dialogfelder 712
 ausführen 709
 benannte Konstanten 709
 DefaultTab-Eigenschaft 713
 Dialogs-Auflistung 709
 direkt ausführen 710
 Einstellungen aktualisieren 710
 Feldfunktionen 272
 Index 714
 integrierte 708
 interne 708
 Konstanten 713
 nur ausführbare Dialogfelder 714
 Organizer für Bausteine 280
 Rückgabewerte 710
 Übersicht der Wordbasic-Anweisungen 712
 Document Object Model *siehe* DOM
 Document Object Model *siehe* XML
 Dokument 188
 als Textdatei speichern 193
 angehängte Vorlage 191
 Ansicht festlegen 653
 ausdrucken 196
 AutoWiederherstellen-Info 988
 Eigenschaft 270
 bearbeiten 670, 672
 Dokumentbereich anzeigen 271
 eintragen 652
 Eigenschaften
 Bausteine 266
 erstellen 189
 im Hintergrund ausdrucken 197
 in Vorlage umwandeln 191
 mit Kennwort speichern 193
 mit Vorlage verbinden 191

Dokument (*Fortsetzung*)
 öffnen 189
 Pfadangaben 191
 schützen 398
 Seitenauswahl drucken 198
 speichern 193
 speichern als PDF 194
 speichern als XPS 194
 statistische Angaben 190
 Typ 190
 Variable bearbeiten 673
 Zoomfaktor festlegen 653
 Dokumentbeschädigung 35, 243
 Dokumenteigenschaft *siehe* Dokument
 Dokumenteinstellungen
 abspeichern 669
 Dokumentvariable *siehe* Dokument
 Dokumentvorlage 202
 globale Add-ins 204
 Druckerschacht 254
 Dsofile.dll 672

E

Early Binding *siehe* Verweis
 Eigenschaftenprozeduren 834
 Eingabeaufforderung
 InputBox 79
 Einheit
 Pixel 688
 Punkt 687–688
 Ereignisse 454
 auf Applikationsebene 459
 anlegen 461
 deklarieren 461
 auf Dokumentebene 456
 ContentControlBeforeDelete 423
 ContentControlBeforeStoreUpdate 424
 ContentControlOnEnter 424
 ContentControlOnExit 424
 DocumentBeforeClose 473
 DocumentBeforePrint 475
 DocumentBeforeSave 474
 DocumentBeforeSync 477
 DocumentChange 472, 542
 DocumentOpen 472
 Gültigkeitsbereich 459
 Klassenmodul einrichten 461
 Klassenmodul initialisieren 463
 MailMergeDataSourceLoad 542
 MailMergeWizardSendToCustom 440
 MailMergeWizardStateChange 440
 NewDocument 471
 NodeAfterInsert 426

NodeAfterReplace 426
 PartAfterAdd 426
 PartAfterLoad 426
 PartBeforeDelete 426
 ProtectedViewWindow 490
 WdProtectedViewCloseReason-
 Enumeration 490
 ProtectedViewWindowActivate 486
 ProtectedViewWindowBeforeClose 490
 ProtectedViewWindowBeforeEdit 488
 ProtectedViewWindowDeactivate 489
 ProtectedViewWindowOpen 487
 ProtectedViewWindowSize 491
 Reihenfolge der Auslösung 457
 Speicherort 459
 WindowActivate 464, 492
 WindowBeforeDoubleClick 466
 WindowBeforeRightClick 467
 WindowDeactivate 465
 WindowSelectionChange 469
 WindowSize 465
 erstellen 316
 Excel
 Tabellenobjekt im Word-Dokument 609
 Tabellenobjekt, Größe festlegen 610
 Tabellenobjekt, schließen 614
 Excel *siehe* Fernsteuern
 Excel-Tabelle in Word verknüpfen 387
 Extensible Markup Language *siehe* XML
 Extensible Stylesheet Language Transformation
siehe XSLT
 Externe Daten einfügen 389
 Externe Daten verknüpfen 387

F

Facets *siehe* XML, Facetten
 Fehlerbehandlung 99, 193
 Fehlermeldungen
 für Task Panes 800
 Feldfunktion
 * MergeFormat-Schalter 316, 383
 aktualisieren 381
 als Baustein 272
 Ausdruck 389
 Bausteine 266
 CreateDate 256, 597
 Database 595
 DocProperty 256
 DocVariable 256
 Feldcode bearbeiten 384
 Feldcode einfügen 217
 Feldcode in Text umwandeln 392
 FileName 256

- Hilfe 380
 - Feldfunktion (*Fortsetzung*)
 - If 256
 - in einem Textfeld 383
 - IncludePicture 316
 - ist Markierung in einer 385
 - Link 388
 - Macrobutton 386, 431, 602
 - NumPages 256
 - Page 256
 - PrintDate 256
 - Ref 415
 - SaveDate 256
 - Section 256
 - SectionPages 256
 - StyleRef 256
 - verschachteln 390
 - Fernsteuern
 - Access 592
 - ADO Verknüpfung 596
 - Bericht ausdrucken 593
 - Datenbank lesen 594
 - Formular anzeigen 593
 - Tabelle in Word verknüpfen 595
 - Datenbanken 594
 - Excel 574, 601
 - ADO-Verbindung 602
 - Arbeitsmappe drucken 575
 - Tabelle als Datenbank ansprechen 601
 - Excel für komplexe Berechnungen verwenden 576
 - Excel-Daten an Word übermitteln 518
 - Outlook 584
 - Dokument versenden 591
 - Kalendereintrag drucken 584
 - Kontakte als Empfängeradresse verwenden 586
 - MailItem-Objekt 591
 - Sicherheitswarnung 590
 - PowerPoint 577
 - Präsentation drucken 578
 - Struktur der Präsentation übernehmen 580
 - Visio 582
 - Zeichnung drucken 582
 - Word 520
 - Dokument drucken 520
 - Fernsteuern (.NET)
 - Late Binding 528
 - Word fernsteuern damit 527
 - Word-Anwendung freigeben 531
 - Word-Instanz starten 528
 - Fett formatieren 208
 - FileDialog-Dialogfeld 715
 - anzeigen und ausführen 717
 - Datei speichern 721
 - Dateien öffnen 720
 - Dateien wählen 723
 - Filter definieren 718
 - Filter löschen und erstellen 719
 - Mehrfachauswahl 720
 - Ordner wählen 724
 - Rückgabewerte 717
 - Formatierung 282
 - Formatvorlage
 - anwenden 294
 - Auswahl in der Benutzerschnittstelle 283
 - definieren 294, 653
 - Einschränkungen 285
 - neue erstellen 293
 - Priorität 806
 - Sichtbarkeit 806
 - Sortierreihenfolge 806
 - statischer Text 306
 - verknüpft 283, 293, 803
 - Vorteile 282
 - Word-interne 290
 - Formel im Bereich berechnen 218
 - Formular 393
 - Enter-Taste unterbinden 791
 - Feldresultate lesen und schreiben 396
 - Formularfeld Ereignisse 400
 - Gültigkeitsprüfung 400
 - Inhaltssteuerelemente 403
 - Optionenfelder 400
 - schützen 398
 - Steuerelement einbinden 500
 - Zugriff auf unsichere ActiveX-Controls (Steuerelemente) 501
- G**
- Geschützte Dokumentansichten *siehe* Ereignisse
 - ProtectedViewWindow 484
 - geschütztes Ansichtsfenster *siehe* Ereignisse
 - ProtectedViewWindow 484
 - GetSetting 663
 - Gitternetz *siehe* Zeichnungselemente
 - Grafiken 308
 - Anker 323
 - AutoFormen
 - mit VBA einfügen und bearbeiten 335
 - Beschriften 321
 - einfügen 309
 - Grafikart 313
 - mit Dialogfeld 315
 - Textfluss beim 309, 313
 - über Zwischenablage 312
 - in Positionsrahmen 321

- ist SmartArt 637
- mit dem Text verschieben 324
- positionieren 310, 322
- Reihenfolge 332
- Grafiken (*Fortsetzung*)
 - Textflussformatierung 329, 331
 - Verknüpfung 316
 - auflösen 318
 - verwalten 319
- GUID für Office 857

H

- Hilfe 38
 - Objektkatalog 36
 - Word 2007 39
- HTML 872

I

- IIF-Funktion 721
- Information *siehe* Range-Objekt 214
- Inhaltssteuerelemente 402
 - benutzerdefinierter XML-Teil 412
 - Content Controls Toolkit 412
 - Daten verbinden 411, 415
 - Eigenschaften 404
 - Ereignisse 423
 - gruppieren 407
 - gruppieren (programmtechnisch) 423
 - Inhalt eingeben (programmtechnisch) 419
 - miteinander verbinden 415
 - Platzhaltertext 404
 - Platzhaltertext formatieren 404
 - programmtechnisch identifizieren 419
 - schützen 407–408
 - strukturiertes Bearbeiten 407
 - XML 410
- INI-Datei 139
 - als Zwischenspeicher 515
 - Werte lesen und schreiben 176
- InRange 215
- InStory 215
- Interoperabilität
 - Excel 609, 622
 - Microsoft Graph 623
- Italic 208

K

- Klick-und-los 511
- Kompatibilitäts-Optionen 659
- Konstante 69
- Kontextmenüs 771

- Konvertierfunktionen
 - Zentimeter nach Points 326
- Kopf- und Fußzeile 253, 256
- Kursiv formatieren 208

L

- Lange Dokumente 262
- Late Binding
 - .NET 528
- Late Binding *siehe* Verweis
- Listvorlage 300
 - definieren 301
 - einfache 302
 - erstellen 301
 - gegliederte 306

M

- MacroContainer 826
- Makro
 - dem Menüband zuweisen 42
 - der QAT zuweisen 41
 - einer Tastenkombination zuweisen 42
 - erscheint nicht in der Liste der
 - Benutzerschnittstelle 44
 - kopieren 44
 - löschen 44
 - organisieren 44
 - Reihenfolge bei Namensgleichheit 812
 - speichern 34
 - Speicherort 27, 676
 - synchron abarbeiten 514
 - verschieben 44
 - Zielgruppe 678
 - zur Laufzeit nachladen 517
- Makroausführung unterbinden 179
- Makrorekorder 27
 - anhalten 28
 - Arbeitsweise 29
 - Ergebnis betrachten 29
 - starten 27
- Makrosicherheit
 - Projekt signieren 51
 - Sicherheitsstufe anpassen 46
 - Sicherheitswarnung Outlook 590
 - Signatur erstellen 50
 - umgehen 179
 - Zertifikat
 - einlesen 53
 - erstellen 52
- Mathematische Funktionen 576
- Menüband 728
 - aktualisieren 751

- Anwendungs-Schaltflächen 740
- Backstage 774
- bei Null anfangen 755
- CommandBar-Methoden für 542, 731
- Control 730
- Control Group 730
- Menüband (*Fortsetzung*)
 - Custom UI Editor 732, 744
 - Entwurfsmodus in VSTO 562
 - erweitern 732
 - Erweiterungen teilen 751
 - Fehler beim Laden anzeigen 735
 - Grafikdateien ins Dokument einbinden 743
 - Group 730
 - idMso-Werte 739
 - in Dokument einbinden 736
 - Intellisense bei der XML-Eingabe 542
 - Kontextmenü 771
 - QAT 730, 755
 - QuickInfo 742
 - Registerkarten teilen 752
 - Ressourcen im Internet 777
 - Tab 730
 - Tab positionieren 738
 - Tab unsichtbar machen 742
 - Tastenkombination für einen Befehl 738
 - Terminologie 730
 - versionspezifisch 735
 - Word-Befehle übersteuern 542, 741
- Menüband-Attribut
 - AutoSize 754
 - boxStyle 762
 - columns 766
 - enabled 741
 - getEnabled 741
 - getImage 746
 - getItemCount 766
 - getItemHeight 766
 - getItemId 766
 - getItemImage 766
 - getItemLabel 767
 - getItemScreenTip 767
 - getItemSupertip 767
 - getItemWidth 766
 - getPressed 761
 - getSelectedItemId 767
 - getSelectedItemIndex 767
 - getText 763
 - getVisible 751
 - idMso 739
 - idQ 751
 - image 745, 760
 - imageMso 743, 760
 - insertAfterMso 738
 - insertAfterQ 738
 - insertBeforeMso 738
 - insertBeforeQ 738
 - itemWidth 766
 - keytip 738
 - label 759
 - loadImage 543, 746
 - maxLength 763
 - onAction 741, 760
 - onChange 764
 - onLoad 749
 - rows 766
 - screenTip 742, 760
 - size 759
 - sizeString 764
 - startFromScratch 755
 - superscreenTip 742
 - supertip 760
 - tag 759, 761
 - visible 742
- Menüband-Element
 - 741, 772, 776
- Menüband-Methode
 - ActivateTab 752
 - ActivateTabMso 752
 - ActivateTabQ 752
 - Invalidate 748, 751
 - InvalidateControl 748, 761
- Menüband-Steuerelement 755
 - Bearbeitungsfeld 763
 - Beschriftung 766
 - box 762
 - button 759
 - buttonGroup 763
 - checkBox 761
 - comboBox 765
 - dialogBoxLauncher 760
 - dropDown 766, 770
 - dynamicMenu 770, 772
 - editBox 763
 - gallery 766
 - item 765
 - Katalog 766
 - Kombinationsfeld 765
 - Kontrollkästchen 761
 - labelControl 766
 - Schaltfläche 759
 - splitButton 760
 - toggleButton 749, 761
 - Trennschaltfläche 760
 - Umschaltfläche 761
- Microsoft Graph im Word-Dokument 623
- Mitteilungen an den Benutzer
 - MsgBox 80

MSForms *siehe* Benutzerdefinierte Dialogfelder
 MSXML-Parser *siehe* XML, Parser
 Multifunktionsleiste 728

N

Namenskonflikt 455
 Namenskonventionen 972
 Namespaces *siehe* XML, Namensräume
 neue Zeile in einer Zeichenkette 83
 Nodes *siehe* Knotenpunkte
 Normal.dotm 203, 679
 defekt 988
 konfigurieren 650
 Nummerierung 297
 Absätze einer Liste ermitteln 298
 automatische in statische 298
 Eigenschaften 298
 entfernen 298
 Nummerierung *siehe* Listenvorlage

O

Objekte
 als Variablen verwenden 161
 eingebettete 606
 freigeben 164
 im Objektmodell aufspüren 160
 Standardeigenschaft 164
 Objektkatalog 36
 Office
 DOM 418
 Office 2010 64 Bit 118
 Office Open XML *siehe* Open XML
 OLE 606
 In-place-Aktivierung 606, 616
 Objekt
 aktivieren 607
 deaktivieren 608
 einfügen 606
 OLE-Client 606
 OLE-Server 606
 OnTime-Methode 854
 OOXML *siehe* Open XML
 Open XML 916
 als Word Dateiformat 916
 Bestandteile eines Pakets (Package) 919
 Beziehung (Relationship) 919, 939
 Custom XML Part 943
 Custom XML Part, Beispiel 959
 Dokument im Texteditor erstellen 917
 Inhalt lesen, Beispiel 948
 Name 919
 OPC 918

flaches Format 919
 historische Entwicklung 925
 volles Format 923
 Open Packaging Conventions *siehe* Open XML, OPC
 Programmierung 944
 SDK 947
 Teil (Part) 919, 935
 Teil zufügen, Beispiel 926
 Teil-Typen 935
 Typ (Type) 919
 und VBA 947
 vs. das binäre Dateiformat 916
 vs. WordProcessingML 918
 Werkzeuge für die Bearbeitung 946
 XSLT 965
 Ziel (Target) 919
 Option Explicit-Anweisung 843
 Optionen
 Kategorie Dokumentprüfung 657
 Kategorie Erweitert 659
 Kategorie Speichern 657
 Outlook *siehe* Fernsteuern

P

PathSeparator *siehe* Application-Objekt 180
 Pfade
 Backslash anhängen (API) 123
 Datei an Pfad anhängen (API) 125
 Kombinieren zweier Pfade (API) 123
 um Dateierweiterung ergänzen (API) 125
 zwei Pfade kombinieren (API) 124
 Pfadtrennzeichen 180, 191
 PI (processing instruction *siehe* Verarbeitungsanweisung)
 Platzhalter *siehe* Dateisystem
 Positionsrahmen
 als Grafikbehälter 321
 Anker 323
 einfügen 321
 in Formatvorlagen 321
 mit dem Text verschieben 324
 positionieren 322
 PowerPoint *siehe* Fernsteuern
 Programmbibliothek *siehe* Add-In

Q

QAT *siehe* Menüband, QAT
 Quick Access Toolbar *siehe* Menüband, QAT

R

- Registry
 - Editor starten 174
 - Sicherung 132
 - Werte lesen und schreiben 174
 - Zugriff auf die Registry (API) 132
- Rückgängig machen steuern 177

S

- Schaltfläche für Office
 - Befehle außer Kraft setzen 742
- Schemabibliothek 877, 903
 - Schema laden 904
- Schnellbausteine, *siehe* Bausteine 266
- Schriftfarbe 358
- Schritt-für-Schritt
 - AutoKorrektur-Eintrag erstellen 655
 - Benutzerdefiniertes Dialogfeld erstellen 702
 - Custom UI Editor für Menüband-
 - Erweiterung 744
 - Digitale Signatur erstellen 50
 - Digitale Signatur zuweisen 51
 - Early und Late Binding 508
 - eine Transformation mit einem Schema
 - verbinden 910
 - Inhaltssteuerelemente gruppieren und
 - schützen 408
 - interne Word-Befehle auflisten 814
 - Makro einer Symbolschaltfläche zuweisen
 - (QAT) 41
 - Makro einer Tastenkombination zuweisen 42
 - Menüband-Erweiterung einem VSTO-Projekt
 - zufügen 537
 - nachträglich ein geöffnetes Dokument
 - transformieren 907
 - Normal.dotm automatisch anlegen 651
 - Schema in die Schemabibliothek laden 904
 - Tabellenformatvorlage erstellen 364
 - Textdrucker installieren 201
 - Userform erstellen 702
 - XML Datei erstellen 872
 - Zertifikat einlesen 53
 - Zertifikat exportieren 52
- ScreenUpdating *siehe* Application-Objekt 178
- Seitenlayout 248
 - Bundsteg 251
 - festlegen 652
 - Ränder 250
- Seitenumbruch einfügen 210
- Seitenzahlen 259
- Selfcert.exe 50
- SendKeys 738
- Seriendruck 433
- Datenquelle einbinden 446
- Datensatz
 - ausschließen 436
 - navigieren 435
 - suchen 438
- Datenverbindungen
 - DDE, Access 450
 - DDE, Excel 451
 - Dokumente 449
 - Konvertierfilter, Textdateien 449
 - ODBC, Access 450
 - ODBC, Excel 451
 - OLE DB, Access 451
 - OLE DB, Excel 452
- Etiketten 444
- Hauptdokument 434
 - in normales Dokument umwandeln 434
 - öffnen 440
- Seriendruck-Assistent 440
- Umschläge 442
 - zusammenführen 441
- Sicherheit *siehe* Makrosicherheit
- Sicherheitscenter 820
- Signatur 49
- Silbentrennung
 - konfigurieren 654
- SmartArt-Grafik 633
 - Ebenen durchlaufen 640
 - einfügen 635
- Elemente
 - beschriften 640
 - verschieben 642
- Farbe festlegen 638
- Indexwert ermitteln 635
- ist vorhanden 637
- Layout der Elemente 642
- Layouts auflisten 634
- Speichern
 - AutoWiederherstellen-Info 988
- Speicherorte (standardmäßige) festlegen 181
- Sprache der Word-Umgebung 180
- Startoptionen 984
- Startup-Ordner 181, 203, 523, 823
- statistische Angaben eines Dokuments 190
- Steuerelemente 847
- Suchen und Ersetzen 220
 - im ganzen Dokument 233
 - in einer Schleife ausführen 225
 - Unterschiede zur Benutzerschnittstelle 223
- Symbolleiste 799
 - ab Word 2007 731
- Symbolleiste für den Schnellzugriff *siehe* Menüband, QAT
- System Informationen
 - Betriebssystem 174

Ländereinstellung 173–174
Mauszeiger 173

T

Tabelle

aus Access einbinden 595
Automatisierung von 351
Daten lesen 372
Einzug 361
Excel-Tabellenobjekt 609
formatieren 358

Tabelle (*Fortsetzung*)

Höhe 371
im Dokument finden 372
Kopfzeile wiederholen 362
letztes Zeichen einer Tabellenzelle 360
mit Textfluss auf einer Seite behalten 369
positionieren 367
schattieren 359
senkrechte Ausrichtung vom Text in einer Zelle 362
Spalte formatieren 360
Spaltenbreite 369
Textanpassen 363
verschachtelte 376
Zeile formatieren 360
Zeilenhöhe 370
Zellen
 teilen 372
 verbinden 372
Zellenbegrenzung 363
Zellenumbruch 363
Zellenwechsel 361

Tabellenformatvorlage 363

diagonale Rahmenlinien 366
Formatierung von Eckzellen 366
Kopfzeile wiederholen 366
Standard für neue Tabellen 367
Streifen 366
Tabelleneigenschaften 366

Task Pane 794

abfangen oder außer Kraft setzen 802
Anlageoptionen 798
Besprechungsarbeitsbereich 798
Clipart 796
den Inhalt aktualisieren 800
die Word-eigenen anpassen 794
Dokument schützen 796
Dokumentaktionen 796, 802
Dokumentaktualisierung 796, 802
Dokumentverwaltung 796
eigene erstellen 794
einblenden 795

Einfache Suchoptionen 797
Faxdienst 796, 802
Fehlermeldungen 800
Formatierungen anzeigen 797
Formatinspektor 797
Formatvorlage übernehmen 797, 803
Formatvorlagen 803
Formatvorlagen und Formatierungen 796, 803
Formatvorlagen, Optionen 806
Freigegebener Arbeitsbereich 797
Hilfe 797
positionieren 795, 798–799
Recherchieren 797–798, 802
Seriendruck 797, 802
Signaturen 798, 803
Typen 796
Work Pane 794
XML-Dokument 798
XML-Struktur 798, 902
Zugriffsprüfung 798
Zwischenablage 796, 802

Tastaturkürzel

Alt+F11 29, 34
Alt+F9 316, 382, 387
F1 40
F2 36
F8 95
F9 98
Str+Umschalt+F7 266
Strg+F9 99, 382
Strg+G 95
Strg+S 34
Strg+Umschalt+F9 99
Umschalt+F9 97

Tastenkombination

(Strg)+(Bild ab) 238
(Strg)+(Bild auf) 238
Alle auflisten 785
aussperren 790
entfernen 789
für bestimmten Befehl ermitteln 788
programmtechnisch anlegen 790
programmtechnisch ermitteln 783
programmtechnische Definition 781

Tastenstatus ermitteln (API) 150

Tastenwerte

ASCII-Werte 150
Virtuelle Tastenkonstanten 150

Textmarke 342

Benennung von 343
Daten schreiben 346
einfügen 343
Inhalt lesen 348
verborgene 342
vordefinierte 349

TrueType-Schrift
 Eigenschaften 658
 einbetten 657

U

Umgebungsvariablen
 Environ-Funktion 144
 Umwandlungsfunktionen *siehe* Datentyp
 Uniform Resource Identifier *siehe* URI
 URI 888
 Userform *siehe* Benutzerdefinierte Dialogfelder
 UTF-8 *siehe* Codierung

V

Variable 56
 deklarieren 57, 61
 Gültigkeit 59
 Sichtbarkeit 59
 Werte nachprüfen 97
 VBA
 Compiler-Konstanten 269
 Global-Objekt 688
 Sicherheitscenter 820
 Systems-Objekt 688
 Versionsnummer 269
 Vertrauensstellungscenter 820
 VBA-Code 820
 VBA-Projekte 823
 VBE
 AddFromString-Methode 844
 Application-Objekt 826
 Designer 854
 Document-Objekt 826
 MacroContainer-Eigenschaft 826
 Makro ausführen 838
 Property Get 834
 Property Let 834
 Property Set 834
 Property-Prozeduren 834
 References-Auflistung 856
 Show-Eigenschaft 851
 Template-Objekt 826
 UserForm-Objekt 851
 UserForms-Auflistung 852
 Verweise neu setzen 862
 VB-Editor 34
 Code speichern 34
 Debuggen 94
 Eigenschaftenfenster 696
 Haltepunkt
 festlegen 98
 Haltepunkte

alle entfernen 99
 Hilfe zum 34
 IntelliSense 35
 lange Codezeilen umbrechen 83
 nächste Anweisung festlegen 99
 programmtechnisch steuern 822
 Sicherheitskopien erstellen 34, 686
 Überwachungsbereich 97
 Werkzeugsammlung 696
 Verarbeitung unterbrechen (API) 146
 Verknüpfungen 387
 relative Pfadangaben 317
 verwalten 319
 Vertrauensstellungscenter 820
 Verweis 496
 auf Add-In setzen 523
 Early Binding 503, 507, 574, 611
 Late Binding 504, 507, 574, 609
 manuell einfügen 497
 unterschiedliche Versionen 506
 Visio *siehe* Fernsteuern
 Visual Basic-Editor 820
 Code-Fenster 821
 Eigenschaftenfenster 822
 Microsoft Visual Basic for Applications Extensibility 5.3 821
 Option 'Zugriff auf das VBA-Projektobjektmodell vertrauen' 821
 Projekt-Explorer 821
 Sicherheitsstufe anpassen 820
 UserForm-Fenster 821
 VSTO 551
 Aufgabenbereich Dokumentaktionen 556–557
 Benutzerschnittstelle bereitstellen 556
 Code hinter dem Dokument 554
 Daten-Cache 564
 Datenquelle mit Steuerelementen verbinden 560
 Daten-Zwischenspeicher 564
 Dokument mit anderer Vorlage verbinden 568
 Dokument vorbereiten 555
 Dokumentlösungen 551
 Inhaltssteuerelemente 556
 InnerObject 559
 integrierte Registerkarte anpassen im Menüband-Entwurfsmodus 563
 Lösung anlegen 552
 Lösung vom Dokument trennen 568
 Menüband-Entwurfsmodus 562
 Menüband-Erweiterung zufügen 537
 Menüband-XML 537
 Nachschlagswerke 571
 Nachteile 551
 RemoveCustomization 568
 ServerDocument-Funktionalität 564, 568

Steuerelemente 555
 ThisApplication-Objekt 559
 Vorteile 551
 Werkzeuge 551

W

Wave-Dateien abspielen (API) 147
 Windows-Registrierung
 Data-Key löschen 988
 Werte einlesen 663
 Werte speichern 662
 Winword.exe
 Startoptionen 984
 WithEvents 461
 Word
 Neu in Word 2010
 Menüband 274
 Startoptionen 984
 Word *siehe* Fernsteuern
 WordArt 628
 formatieren 632
 Word-Befehle
 übersteuern 741, 812
 unabhängig von Programmiersprache 817
 Word-Meldungen unterbinden 179
 WordML *siehe* WordProcessingML
 WordProcessingML 876, 898
 Work pane *siehe* Taskpane 794

X

XHTML 872, 876
 XML 870
 Attribut 879
 benutzerdefiniertes Vokabular in Word 901
 CDATA 881
 Datei erstellen 872
 Deklarationen 880
 Dokument beim Öffnen transformieren 907
 Dokumentinstanz 881
 DOM 667, 878, 893
 Element 878
 Facetten 885
 gültiges Dokument 882
 Inhaltssteuerelemente und 410
 Knotenpunkte 893, 895
 Markup 881
 Namensräume 428, 878, 886
 Namensraumpräfix 430, 890, 892, 903

Namensraumpräfix programmtechnisch
 ermitteln 430
 Nutzen in Word 876
 Parser 878, 912
 Schema 883, 886
 Schema in Schemabibliothek laden 904
 Schema in Word einem XML-Dokument
 zuweisen 905
 Schema programmtechnisch der Schemabiblio-
 thek hinzufügen 912
 Solutions in Word 907
 Tag 873, 878
 Transformation mit dem DOM 912
 Transformation in Word 907
 Transformation mit Schema verbinden 910
 transformieren 874
 Verarbeitungsanweisung 881, 900
 Vokabular 872
 wohlgeformtes Dokument 882
 Wurzelement 881, 890
 XPath 878
 XSLT 878
 XSD *siehe* XML, Schema
 XSLT 874, 894

Z

Zeichencodierung *siehe* Codierung
 Zeichenkette
 verknüpfen 63
 Zeichenkettenbearbeitung
 Format 84
 InStr 83
 InstrRev 83
 Left 82
 Mid 82
 Replace 83
 Right 82
 Zeichnungselemente
 optimieren 655
 Zeichnungsraaster *siehe* Zeichnungselemente
 Zeilenweise verschieben 187
 Zeitungsspalten 252
 Zentral-/Filialdokumente 262
 Zertifikat 49
 Zoomfaktor
 festlegen 653
 Zugriff auf die Registry (API) 132
 Zwischenablage
 Inhalte einfügen 312
 Office 796

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 Betaphase von Office 2003 stellte sie sich den neuen Herausforderungen von XML, Smart Documents und VSTO (was zugleich die ersten Schritte in der 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 2008 wurde sie erstmals als MVP für VSTO erkannt. Sie kann über die E-Mail-Adresse 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 neun Jahren ist Christian Freßdorf aktiv in den deutschen News-groups und Office-Foren und schwerpunktmäßig in 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 schwerpunktmäßig Add-Ins und Prozeduren rund um die VBA- und Menüband-Programmierung zur Verfügung. Sie können ihn über die Kontaktdaten auf seiner Homepage oder über 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 neun Jahren sein Wissen auf microsoft.public.de.word.vba weiter und war in den Jahren 2002 bis 2009 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. Er ist spezialisiert auf technischen Support, Troubleshooting, unabhängige Verifikation und Validierung für große Projekte und Unterstützung bei großen Projekten im Allgemeinen. Word für Windows benutzt er seit der Version 1.0 und war mehrere Jahre Microsoft MVP. Er wohnt in Manchester, England.

Peter Jamieson ist Mitautor eines weiteren Microsoft Press-Buchs: *Microsoft Word – Das Profibuch*. Sie können ihn unter pjj@pjj-net.demon.co.uk kontaktieren.

