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

Microsoft Word Programmierung – Das Handbuch, © 2010 O'Reilly Verlag GmbH & Co. KG, ISBN 978-3-86645-458-3

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.

Microsoft Word Programmierung – Das Handbuch, © 2010 O'Reilly Verlag GmbH & Co. KG, ISBN 978-3-86645-458-3

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
```


Microsoft Word Programmierung – Das Handbuch, © 2010 O'Reilly Verlag GmbH & Co. KG, ISBN 978-3-86645-458-3

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 Arbeitblatts 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 = CDbl(InputBox("Anzahl Elemente eingeben", "Kombinationen berechnen", "64"))
  dblBasis = CDbl(InputBox("Anzahl Elemente in jeder Kombination eingeben", _
                           "Kombinationen berechnen", "4"))

'Neue Excel-Instanz erzeugen
  Set xlsApp = New Excel.Application
  With xlsApp
    dblResultat = .WorksheetFunction.Combin(dblZahl, dblBasis)
    .Quit
  End With

  MsgBox "Kombination mit Excel COMBIN-Funktion berechnet" & vbCr & _
    "Anzahl Elemente" & vbTab & dblZahl & vbCr & _
    "Anzahl Elemente in einer Kombination" & vbTab & dblBasis & vbCr & _
    "Anzahl mögliche Kombinationen" & vbTab & dblResultat & vbCr

  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.

Microsoft Word Programmierung – Das Handbuch, © 2010 O'Reilly Verlag GmbH & Co. KG, ISBN 978-3-86645-458-3

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 nebst den Texten aus Platzhaltern auch Texte aus Textfeldern, Kommentaren oder Notizenseiten übertragen werden.

Listing 11.4 Einlesen der Texte aus den Platzhaltern und übertragen in ein Dokument

```
Sub Demo_PowerpointStruktur_Übertragen()
  Const strDATEI_NAME As String = "C:\WordBuch\Beispiele\Kap11\Bsp_Demo.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 & vbCr
            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 & vbCr
```

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 olKalenderEintrag = olOrdner.Items(1)
```

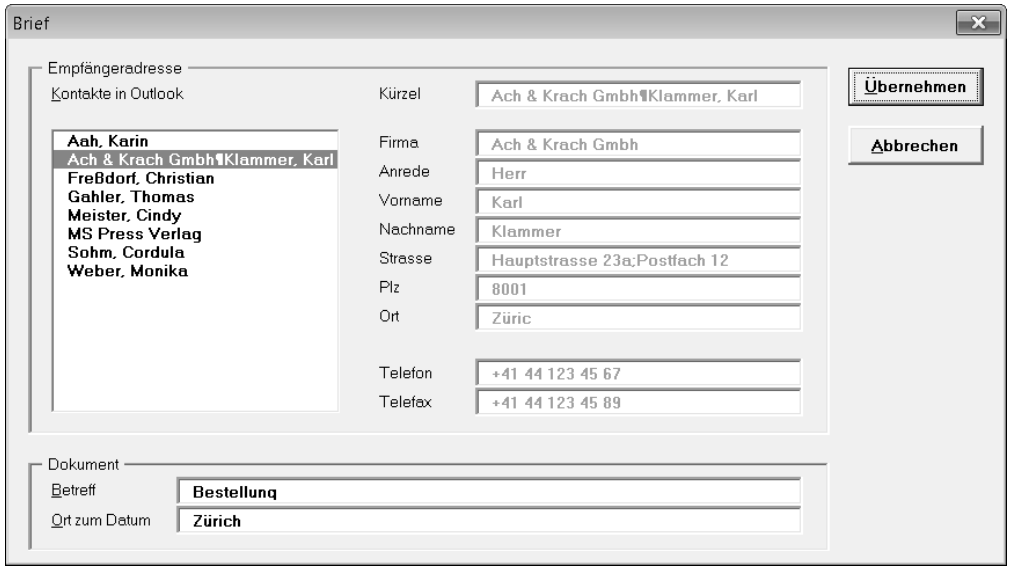
CD-ROM Das obige Beispiel finden Sie in der Beispieldatei *Bsp11_01.docm* auf der CD-ROM zum Buch im Ordner *\Beispiele\Kap11*.

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:

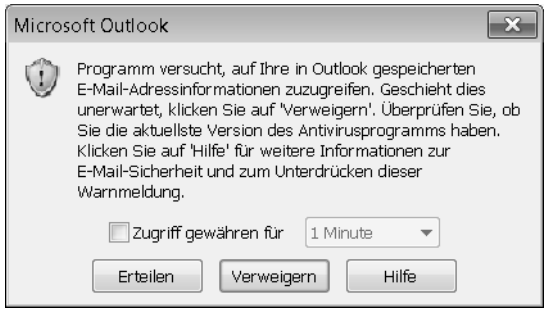
```
strItemFilter = "[FileAs] = " & Chr$(34) & frmBrief.lstKontakteKürzel.Text & Chr$(34)
Set olItemFilter = olItem.Restrict(strItemFilter)
```

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 erhaltet 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 end_Sub
  End If
End With
```

Da nur eine gespeicherte Datei als Anhang hinzugefügt werden kann, wird ausführlich geprüft, ob die Datei überhaupt schon gespeichert wurde (was bei neu angelegten Dateien nicht der Fall ist). In diesem Fall wird das Dialogfeld `Dialogs(wdDialogFileSaveAs)` zum Speichern der Datei angezeigt. Nur wenn der Dialog über *Speichern* verlassen wird und das Dokument anschließend gespeichert ist

(`ActiveDocument.Saved = True`), wird die Datei als Anhang an die Mail angehängt. Andernfalls wird der Dateiversand abgebrochen.

Auch wenn die Datei nach dem letzten Speichern noch geändert und anschließend nicht wieder gespeichert wurde, erfolgt eine entsprechende Abfrage.

Listing 11.12 Prüfen, ob das aktuelle Dokument gespeichert ist

```
'Prüfen, ob aktuelles Dokument gespeichert ist und ggf. speichern
  Dim bSaved As Boolean: bSaved = True
  Do While ActiveDocument.Saved = False Or ActiveDocument.Path = ""
    If ActiveDocument.Path = "" Then ' Neues Dokument, noch nicht gespeichert
      bSaved = False ' Nicht gespeichert
      MsgBox "Das Dokument wurde noch nicht gespeichert." & vbCrLf & _
       "Bitte erst speichern!", vbInformation, c_Title
      With Dialogs(wdDialogFileSaveAs) ' Speichern-Dialog aufrufen
        intRet = .Show
      End With
      If intRet = -1 And ActiveDocument.Saved = True Then ' Über 'Speichern' gespeichert
        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:

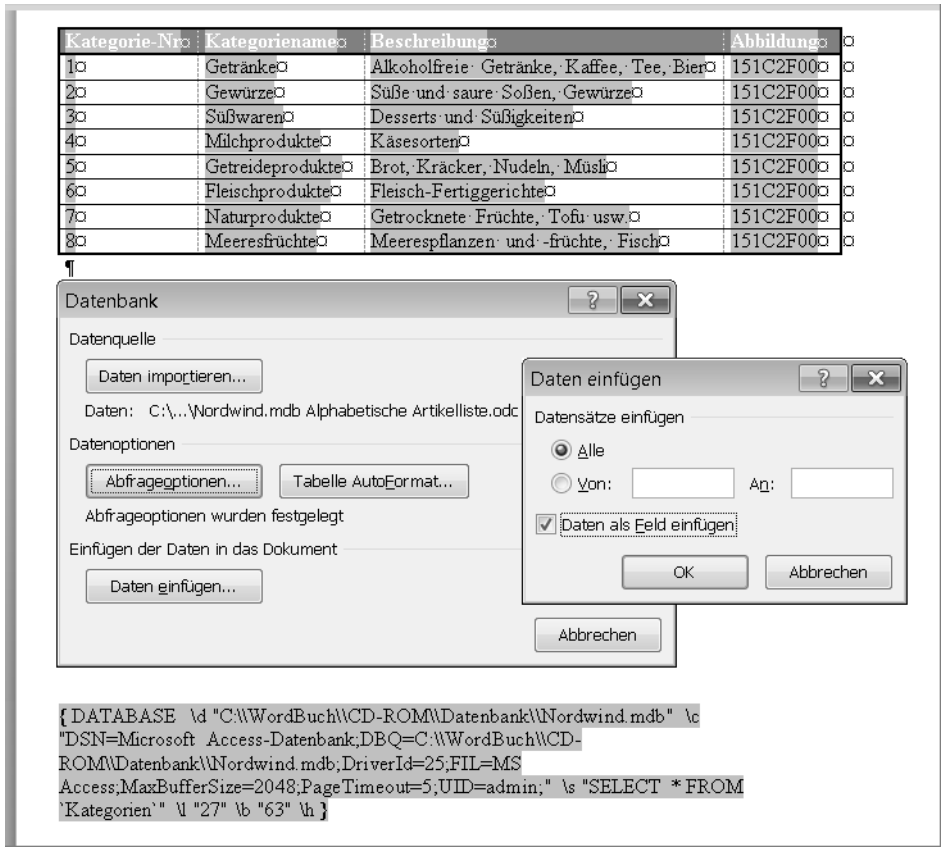
- Die Schnittstelle wird von allen Word-Versionen unterstützt
- Wird eine dynamische Verknüpfung aufgebaut, funktioniert diese Schnittstelle am zuverlässigsten

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.

Microsoft Word Programmierung – Das Handbuch, © 2010 O'Reilly Verlag GmbH & Co. KG, ISBN 978-3-86645-458-3

Abbildg. 11.5 Daten aus einer Access-Datenbank mit der Feldfunktion *Database* in Word als Tabelle verknüpfen



Kategorie-Nr	Kategoriename	Beschreibung	Abbildung
1	Getränke	Alkoholfreie Getränke, Kaffee, Tee, Bier	151C2F00
2	Gewürze	Süße und saure Soßen, Gewürze	151C2F00
3	Süßwaren	Desserts und Süßigkeiten	151C2F00
4	Milchprodukte	Käsesorten	151C2F00
5	Getreideprodukte	Brot, Kräcker, Nudeln, Müsli	151C2F00
6	Fleischprodukte	Fleisch-Fertiggerichte	151C2F00
7	Naturprodukte	Getrocknete Früchte, Tofu usw.	151C2F00
8	Meeresfrüchte	Meerespflanzen und -früchte, Fisch	151C2F00


```
{ DATABASE  \d "C:\\WordBuch\\CD-ROM\\Datenbank\\Nordwind.mdb"  \c
"DSN=Microsoft  Access-Datenbank;DBQ=C:\\WordBuch\\CD-
ROM\\Datenbank\\Nordwind.mdb;DriverId=25;FIL=MS
Access;MaxBufferSize=2048;PageTimeout=5;UID=admin;"  \s "SELECT  * FROM
`Kategorien`"  \l "27"  \b "63"  \h }
```

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 anwendungspezifischen ODBC-Treiber umgesetzt. Dies besorgt bei ADO ein OLE DB-Provider für die Datenschnittstelle. Das Verbindungsprotokoll (»Connection String«), das die Verbindung herstellt, ist OLE DB-Provider-spezifisch. Eine umfassende Liste der »Connection Strings« für mehrere Datenbanktypen finden Sie unter *http://www.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

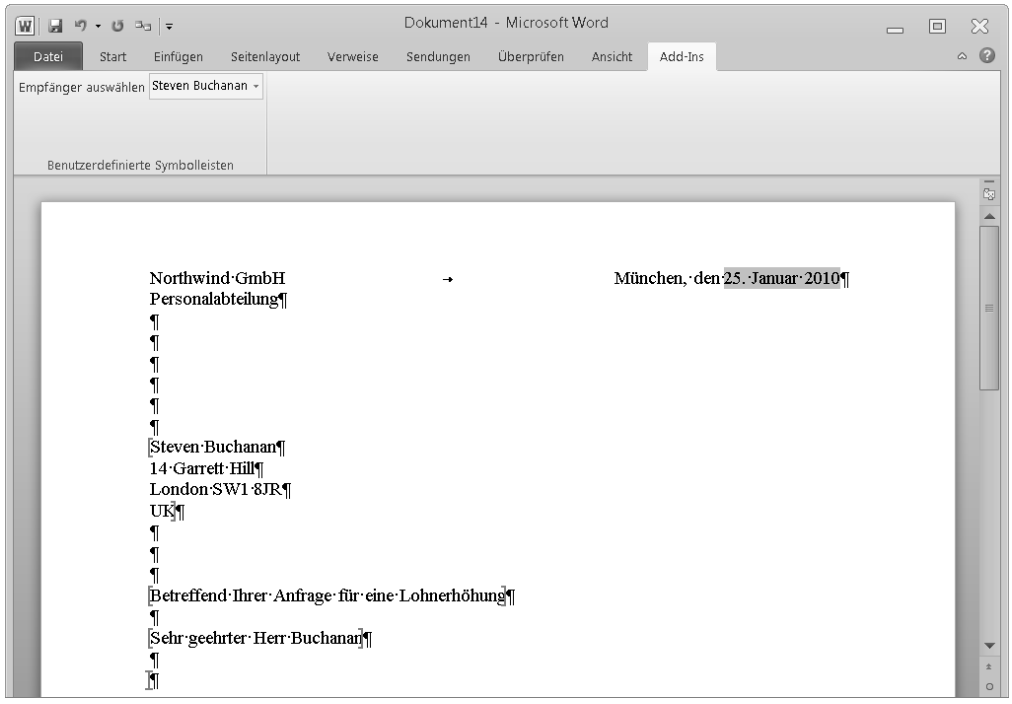
Personal-N	Nachnam	Vorname	Position	Anrede	Geburtsdatun	Einstellung	Straße
1	Davolio	Nancy	Vertriebsmitarbeiterin	Frau	08.Dez.1968	01.Mai.1992	507 - 20th Ave. E.
2	Fuller	Andrew	Geschäftsführer	Herr	19.Feb.1952	14.Aug.1992	908 W. Capital Way
3	Leverling	Janet	Vertriebsmitarbeiterin	Frau	30.Aug.1963	01.Apr.1992	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)							

Datensatz: 1 von 9 | Kein Filter | Suchen

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

Microsoft Word Programmierung – Das Handbuch, © 2010 O'Reilly Verlag GmbH & Co. KG, ISBN 978-3-86645-458-3

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 & vbCr & RS.Fields("Straße").Value & _
    vbCr & RS.Fields("Ort").Value & " " & RS.Fields("PLZ").Value & _
    vbCr & RS.Fields("Land").Value, doc, False
  'Den Benutzer zur Eingabe der Betreffzeile auffordern.
  DatenSchreiben "Betreffszeile", 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.

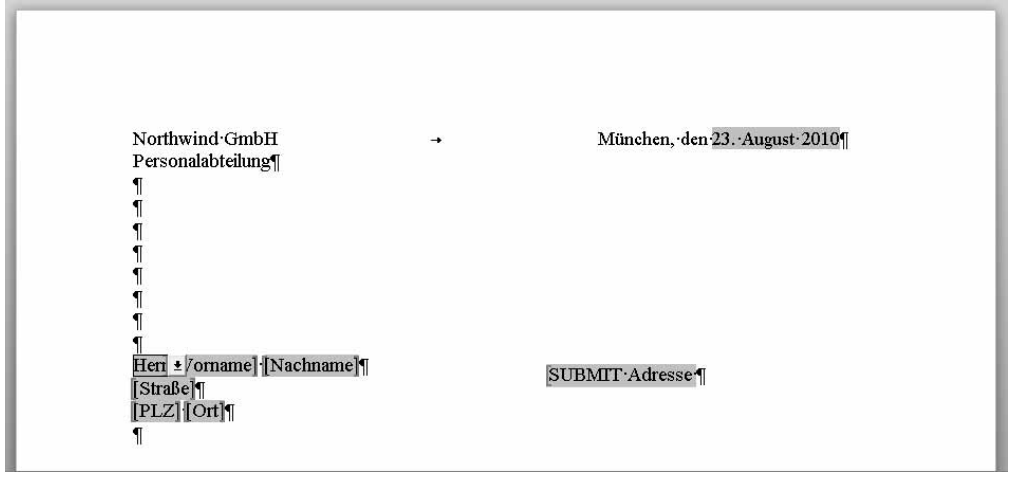
Diese Aufgabe kann auf mehreren Wegen erledigt werden. So ist es beispielsweise möglich, eine Verbindung zur Datenbank herzustellen, auf Basis einer Tabelle einen leeren `Recordset` zu erstellen, ihm neue Datensätze hinzuzufügen und diese mit Daten aus einem Word-Dokument zu füllen. Anschließend werden die Änderungen über die Verbindung zurück in die Datenbank geschrieben.

Microsoft Word Programmierung – Das Handbuch, © 2010 O'Reilly Verlag GmbH & Co. KG, ISBN 978-3-86645-458-3

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



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