

10 VSTO Dokumentlösung mit Textmarken-Steuerelemente

VSTO 2005-Dokumentlösungen

Manchmal ist eine dokumentunabhängige Lösung wie eine *.exe*-Datei oder ein COM-Add-In genau das, was gesucht wird. Es gibt jedoch Aufgaben, für die eine dateiorientierte Methode besser geeignet ist: Die Art des Dokuments (Brief, Memo, Bericht) und die Werkzeuge dafür bilden eine Einheit, und die spezialisierte Funktionalität ist nur in diesem Zusammenhang verfügbar. Für die *.NET*-Umgebung hat Microsoft deshalb VSTO – Visual Studio Tools for Office – entwickelt und 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 im *.NET* Framework festgelegt und unterliegen den üblichen Bestimmungen (das Assembly kann beispielsweise mit einem »Strong name« und digitalen Zertifikat signiert werden).
- Da der Code vom Dokument getrennt ist, müssen nicht beide zusammen am gleichen Ort gespeichert werden. Das Dokument kann beispielsweise lokal auf dem Rechner gespeichert werden, während der Code im Netzwerk oder sogar im Internet bereit liegt.
- Weil Code und Dokument getrennt sind, ist es relativ einfach, die Lösung zu aktualisieren, auch wenn mehrere Benutzer mit ihren Dokumentkopien arbeiten.

Als Nachteile sind die lange Ladezeit beim Öffnen eines Lösungs-Dokuments sowie die Komplexität der Verteilung und Installation einer Lösung zu nennen. Zudem enthalten nur die Stand-alone- und Office-Professional-Versionen von Word 2003 die notwendigen Zusätze, um eine VSTO-Lösung zu laden. Benutzern mit der Standard-Version von Office 2003 steht diese Funktionalität nicht zur Verfügung.

Hinweis Beginn

Visual Studio 2005 Tools for Office (läuft auf *.NET* Framework 2.0 sowie 3.0) wurde für die Zusammenarbeit mit Office 2003 Professional entwickelt. VSTO 2005-Dokumentlösungen laufen auch unter Office 2007 (alle Versionen, nicht ausschließlich Professional). Sie unterstützen jedoch weder die neuen RibbonX- noch die Custom Task Pane-Technologien. Diese werden erst Teil der nächsten

Version von Visual Studio, die seit Ende 2007 erhältlich ist.

Hinweis Ende

Die gegenwärtige Version 2005 hat mit ihrem Vorgänger (VSTO 2003) sehr wenig gemeinsam. Sie ist leistungsfähiger und entwicklerfreundlicher. Dokumentlösungen für Word enthalten einige wichtige Neuerungen gegenüber VSTO 2003:

- **Dokument-Entwurfsmodus.** Das mit der VSTO-Lösung verbundene Dokument wird in Visual Studio angezeigt und kann dort bearbeitet werden (siehe Abbildung 10.4).
- **Windows Forms-Steuerelemente.** Werden von VSTO in ein Office-gerechtes ActiveX-Format »gepackt« und bereitgestellt, so dass sie in ein Word-Dokument eingefügt werden können.
- **Dokumentaktionen-Aufgabenbereich.** Das VSTO-Team hat eine entwicklerfreundliche Schnittstelle für die Smart Document-Technologie bereitgestellt. Somit steht ein eigener, dokumentspezifischer Aufgabenbereich zur Verfügung, der auch Windows Form Controls enthalten kann.
- **Data Binding.** Daten mit einer externen Datenquelle dynamisch verbinden und im Dokument (über Textmarken-Steuerelemente oder XML-Nodes) anzeigen.
- **Data Caching.** Daten können in einem Zwischenspeicher im Dokument in XML-Format gelesen und geschrieben werden, ohne das Dokument in Word öffnen zu müssen.
- **Dokumentspezifische Smarttags.** Begriffe werden nur im VSTO-Dokument erkannt und entsprechende Handlungen im Smarttag-Kontextmenü zur Verfügung gestellt.

Eine eingehende Diskussion der VSTO-Technologie würde ein ganzes Buch füllen. Deshalb bietet der folgende Abschnitt lediglich einen Überblick einiger Facetten der Technologie, mit Beschreibung der ersten Schritte, um den Einstieg zu erleichtern.

Hinweis Beginn

Visual Studio 2005 Tools for Office ist im Lieferumfang von Visual Studio 2005 Team Suite enthalten, kann aber auch als separates Produkt erworben werden.

VSTO 2005 Second Edition, die Werkzeuge für die Entwicklung von Office Add-Ins bereitstellt, kann kostenlos von der Microsoft-Website unter <http://www.microsoft.com/downloads/details.aspx?familyid=F5539A90-DC41-4792-8EF8-F4DE62FF1E81&displaylang=de> heruntergeladen werden. Voraussetzung ist Visual Studio 2005 Professional oder höher. VSTO 2005 SE wird im Abschnitt »**Fehler! Verweisquelle konnte nicht gefunden werden.**« vorgestellt.

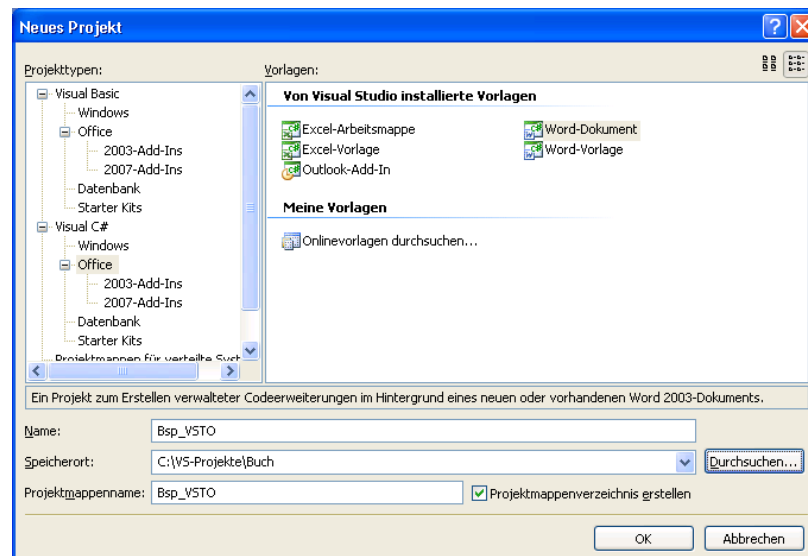
Bitte beachten Sie, dass Visual Studio 2005-Produkte sprachspezifisch sind. Es ist beispielsweise nicht möglich, die englische Version von VSTO 2005 SE zusammen mit der deutschen Version von Visual Studio 2005 Professional zu installieren. Achten Sie also darauf, dass immer die korrekte Sprachversion heruntergeladen wird.

Hinweis Ende

Eine VSTO-Lösung anlegen

Nach erfolgreicher Installation von Visual Studio 2005 Tools for Office erscheinen unter den Visual Basic- und C#-Ordern des Visual Studio-Dialogfelds *Neues Projekt* Ordner für *Office*, wie in Abbildung 10.1 ersichtlich.

Abbildung 10.1: Visual Studio 2005-Projektvorlagen mit installiertem VSTO 2005 sowie VSTO 2005 SE



Nach Auswahl eines Word-Eintrags (*Word-Dokument* bzw. *Word-Vorlage*) wird das Dialogfeld aus Abbildung 10.2 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 10.3).

Abbildung 10.2: Festlegen, ob das Projekt auf einem neuen oder vorhandenen Dokument basieren soll

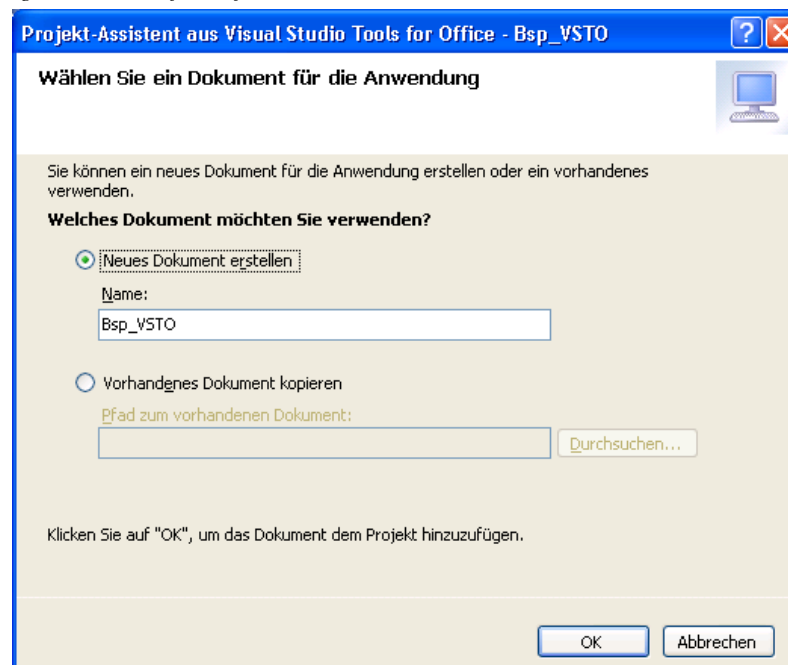
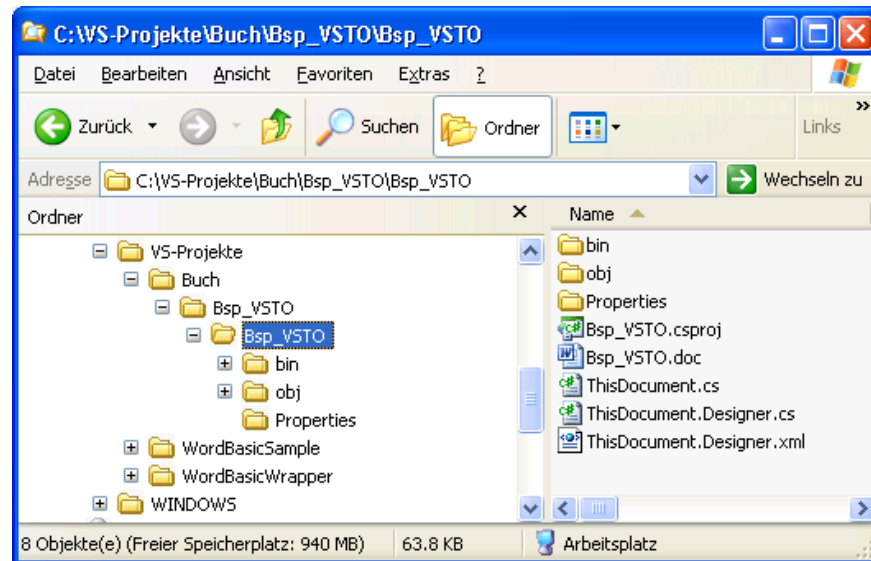
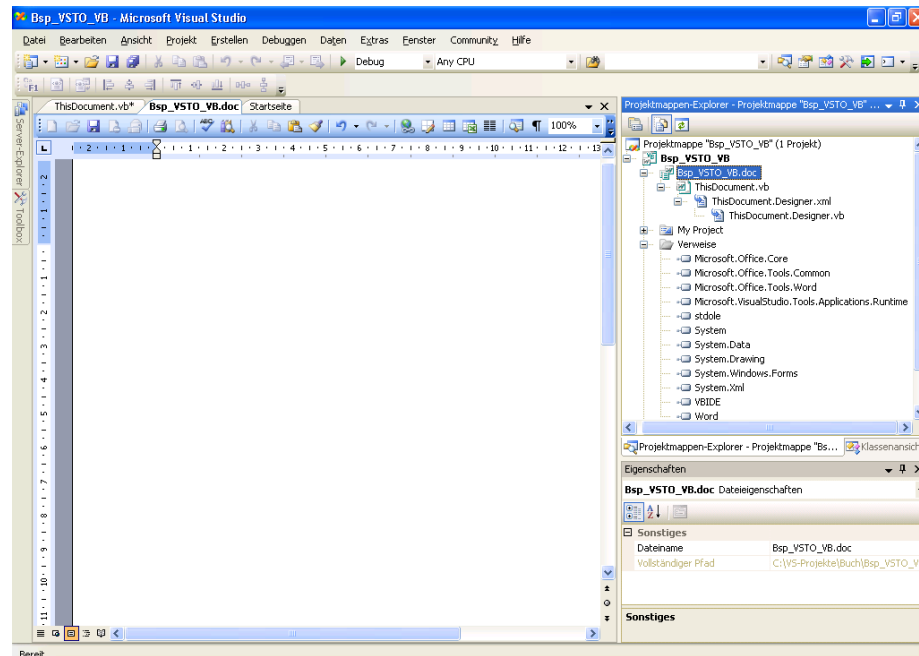


Abbildung 10.3: Von VSTO angelegte Ordner-Struktur und Projekt-Dateien, inklusive des Word-Dokuments



Nach einigen Sekunden erscheint im Visual Studio-Anwendungsfenster der VSTO Designer (der Entwurfsmodus) samt Word-Dokument, dargestellt in Abbildung 10.4. Die Menüleiste enthält zusätzliche Menüpunkte: Darüber stehen alle Word-Befehle zur Verfügung, die in einer VSTO-Lösung einsetzbar sind. Gemeinsame Menünamen werden zusammengefasst. Beispielsweise befindet sich unter *Extras* ein Menüeintrag *Microsoft Office Word Extras*, worunter das *Extras*-Menü von Word zu finden ist.

Abbildung 10.4: Neu in VSTO 2005: Das Word-Dokument kann in Visual Studio bearbeitet werden.



Hinter den Kulissen hat VSTO ein unsichtbares ActiveX-Steuerelement der OLE-Klasse `VSTO.StorageRuntime.1` als Mitglied der Shapes-Auflistung ins Dokument eingefügt. Es ist für die Verwaltung der VSTO-Funktionalität im Word-Dokument verantwortlich.

Hinweis Beginn

Mehr Informationen zum »Runtime Storage Control« finden Sie auf der Seite [http://msdn2.microsoft.com/en-us/library/7ydwehbf\(VS.80\).aspx](http://msdn2.microsoft.com/en-us/library/7ydwehbf(VS.80).aspx).

Hinweis Ende

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 Dateieindung `.cs` für C# bzw. `.vb` für Visual Basic) mit der Teilklasse *ThisDocument*. Sie enthält nur den Code, der für die Zusammenarbeit mit dem Word-Dokument benötigt wird. Wie in Visual Studio 2005 üblich, werden die vom Visual Studio Designer erstellten Support-Strukturen durch weitere (in Abbildung 10.4 ersichtliche) Dateien aufrechterhalten. Im Alltag muss sich der VSTO-Entwickler selten damit befassen, für einen vertiefenden Einblick in die Wirkungsweise von VSTO bieten diese Ergänzungsdateien jedoch eine interessante Lektüre.

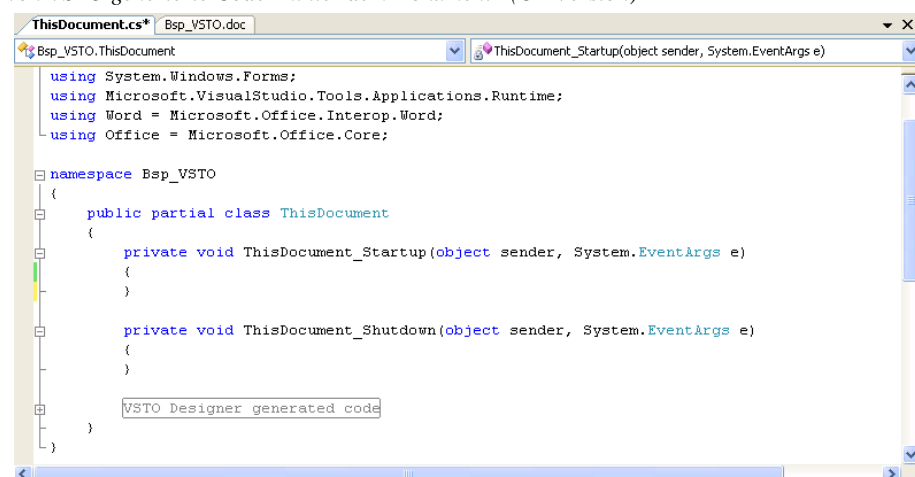
Wie die Abbildung 10.4 zudem veranschaulicht, erledigt VSTO einige Aufgaben für den Entwickler. Verweise zu den benötigten .NET-DLL und COM-Bibliotheken (PIAs) werden erstellt und die C#-Schnittstelle fügt benötigte *using*-Befehlszeilen ein (der VB-Entwickler muss gewünschte Imports-Zeilen selbst eingeben). Außerdem definiert VSTO die zwei Prozeduren *ThisDocument_Startup* sowie *ThisDocument_Shutdown* (Abbildung 10.5). Diese werden beim Starten bzw. beim Schließen des Dokuments ausgeführt: hier ist der Ansatzpunkt für jede VSTO-Lösung.

Tipps Beginn

Die Code-Ansicht kann über den entsprechenden Befehl im Kontextmenü des *ThisDocument*-Eintrags im Projektmappen-Explorer eingeblendet werden.

Tipps Ende

Abbildung 10.5: Der von VSTO generierte Code »hinter dem Dokument« (C#-Version)



Das VSTO-Dokument vorbereiten

Falls das VSTO-Dokument oder die VSTO-Vorlage mit Standardtext oder -inhalt bestückt werden soll, kann die Bearbeitung wahlweise in Word oder im VSTO-Entwurfsmodus erfolgen. Allfällige WinForms- oder VSTO-Steuerelemente (einzig das Textmarken-Steuerelement steht für Word-Lösungen bereit) müssen jedoch im VSTO-Entwurfsmodus hinzugefügt werden.

Als Beispiel für unseren Überblick dient ein Firmenbrief. Die Eingabestellen für Standardangaben wie Empfängeradresse, Betreffzeile und Anrede werden mit VSTO-Textmarken-Steuerelementen gekennzeichnet. Der Benutzer kann diese anklicken und den Text direkt in das Dokument eingeben. Er kann sich auch des Aufgabenbereichs bedienen, der aus der Firmendatenbank eine Liste mit Kundenadressen bereitstellt, woraus er nach Belieben einen Eintrag wählen kann. Der Aufgabenbereich wird mit einer eigens für diese Lösung erstellten Symbolleiste ein- und ausgeblendet.

Die in den Textmarken enthaltenen Daten werden in den Daten-Cache (Dokumentzwischenspeicher) geschrieben, wo sie für die VSTO-Technologie auch bei geschlossenem Dokument lesbar sind. Dazu dient das `ServerDocument`-Objekt der VSTO Runtime.

Ferner zeigt das Beispiel, wie der Benutzer eine VSTO-Dokumenten Anpassung entfernen kann, falls das Dokument an jemanden weitergeleitet wird, der keinen Zugang zur VSTO-Lösung hat.

Hinweis Beginn

Im Gegensatz zu den Code-Beispielen für die Word- und Office-Objektmodelle wird in den folgenden Teilen VB.NET verwendet. Da die von VSTO zur Verfügung gestellten Eigenschaften und Methoden den Regeln von .NET-Framework entsprechen, erfolgt die Umwandlung in C# problemlos. Hingegen wird die Leserschaft, die aus der VBA-Ecke kommt, den VB-Code-Beispielen besser folgen können, was den Einstieg in VSTO etwas erleichtern wird.

Hinweis Ende

VSTO-Textmarken-Steuerelemente

VSTO-Textmarken-Steuerelemente bieten gegenüber gewöhnlichen Textmarken zwei wichtige Vorteile: sie können mit einer Datenquelle und einem Daten-Cache verbunden werden, und sie stellen fünf Ereignisse bereit. Diese Ereignisse basieren auf Word-Anwendungsereignissen:

- `BeforeDoubleClick` (vor einem Doppelklick)
- `BeforeRightClick` (vor einem Rechtsklick)
- `Deselected` (der Fokus verlässt den Textmarkenbereich)
- `Selected` (der Fokus tritt in den Textmarkenbereich ein)
- `SelectionChange` (der Fokus im Textmarkenbereich wird durch bedienen der Maus oder der Tastatur geändert)

Textmarken-Steuerelemente können entweder über die Befehlsfolge *Einfügen/Textmarke* (Abbildung 10.6) oder über die Visual Studio-Toolbox (

Abbildung 10.7) eingefügt werden. Egal, welche Methode gewählt wurde, können Namen sowie Textmarkenbereich nachträglich ganz einfach über das Eigenschaftenfenster geändert werden. Um den Namen zu ändern, genügt es, irgendwo im Textmarkenbereich zu klicken. Um den Bereich zu ändern, muss dieser zuerst markiert werden (mindestens ein Zeichen des gegenwärtigen Bereichs muss sich innerhalb der Markierung befinden).

Achtung Beginn

Wenn Sie auf ein VSTO-Textmarke-Steuerelement doppelklicken, wird eine Klick-Ereignis-Prozedur in der Code-Ansicht erstellt. Eine Markierung muss also mit der Tastatur oder durch klicken und ziehen erfolgen.

Achtung Ende

Abbildung 10.6: Ein Textmarken-Steuerelement über Words internen Einfügen/Textmarke-Befehl einfügen

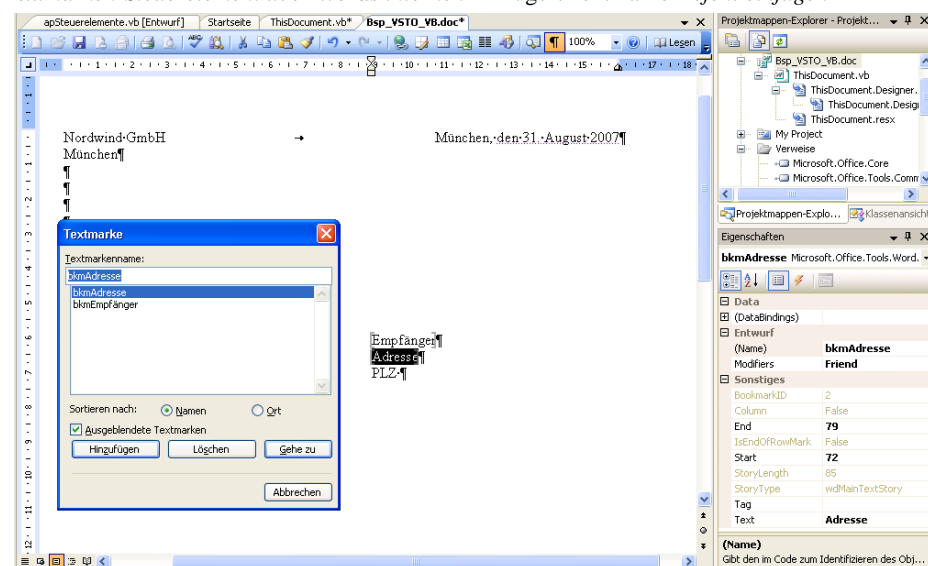
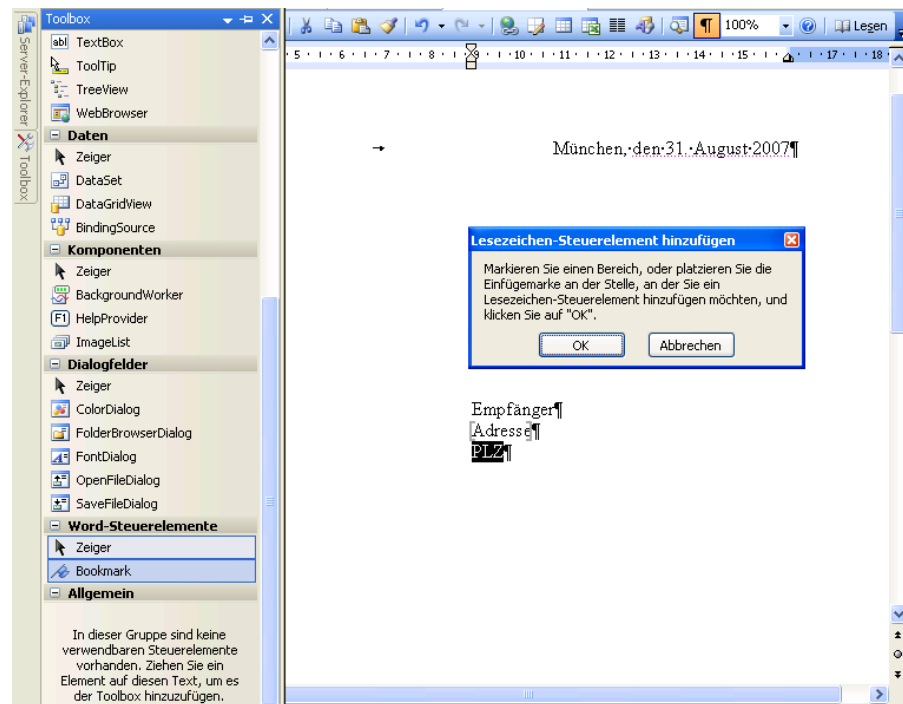


Abbildung 10.7: Ein Textmarken-Steuerelement aus der Visual Studio-Toolbox einfügen



Textmarken sind bekanntlich äußerst »fragil«: sie sind schnell gelöscht, ohne dass der Benutzer es merkt. VSTO-Textmarken-Steuerelemente sind leider genauso anfällig. Um diesem Problem entgegenzuwirken, bedient sich die Beispiel-Lösung der Select- sowie der Deselect-Ereignisse, um dem Benutzer den Umgang mit den Textmarken zu erleichtern. Der Code dafür befindet sich in Listing 10.1.

Bewegt der Benutzer die Einfügemarke in ein Textmarken-Steuerelement, wird dessen Select-Ereignis ausgelöst, das über das Argument *e* die neue Markierung (Selection) bereitstellt. So wird die Textmarke ermittelt und in einer globalen Variablen (*selectedBkm*) festgehalten. Damit der Benutzer mit der Texteingabe sofort beginnen kann, wird der Textmarkenbereich markiert. Anschließend wird der Bereich um ein Zeichen erweitert (die Markierung ändert sich dadurch nicht), so dass die Benutzereingabe das letzte Zeichen der Textmarke nicht erfasst und diese folglich nicht entfernt.

Hinweis Beginn

Die Codezeile `selectedBkm.End = selectedBkm.End + 1` in Listing 10.1 zeigt noch einen Vorteil auf, den VSTO bietet. Eine Word-Textmarke hat keine Start- und End-Eigenschaften, diese gehören dem Range-Objekt. VSTO überträgt diese auf das Textmarken-Steuerelement und spart damit dem Entwickler einige Tastenschläge. Ähnlich steht es um die Select-Methode (`selectedBkm.Select()`).

Hinweis Ende

Wichtig ist natürlich, dass nach der Bearbeitung der Textmarkenbereich zurückgesetzt wird, sonst würde er bei jedem Eintritt in das Textmarken-Steuerelement um ein weiteres Zeichen wachsen. Hierfür verantwortlich zeichnet

das Deselect-Ereignis.

Problematisch bei diesem Vorgang ist die Reihenfolge, in der die beiden Ereignisse ausgelöst werden. Wenn der Benutzer zwischen Textmarken wechselt, müssen zwei Textmarken verwaltet und deren Bereiche korrekt erweitert bzw. gekürzt werden. Es stellt sich heraus, dass diese Reihenfolge nicht vorhersehbar ist: Oft wird zuerst Deselect ausgelöst, aber nicht immer. Daher verwendet der Code drei zusätzliche, globale Variablen: `deselectedBkm` hält die Textmarke, die es zurückzusetzen gilt, fest; `selectedAusgeführt`, ob das Ereignis Select schon ausgelöst wurde; während `deselectedAusgeführt` den Status des Ereignisses Deselect festhält. Die Ereignisse fragen den Stand dieser vier Variablen ab, um den richtigen Zeitpunkt der Ausführung von `deselectedBkm = selectedBkm` zu berechnen.

Listing 10.1: Den Umgang mit Textmarken erleichtern mit den Select- sowie Deselect-Ereignissen

```
Dim selectedBkm As Word.Bookmark = Nothing
Dim deselectedBkm As Word.Bookmark = Nothing
Dim selectedAusgeführt As Boolean = False
Dim deselectedAusgeführt As Boolean = False

Private Sub bkmSelect(ByVal sender As System.Object, _
    ByVal e As Microsoft.Office.Tools.Word.SelectionEventArgs) _
    Handles bkmAdresse.Selected, bkmEmpfänger.Selected, bkmPLZ.Selected, _
    bkmLand.Selected, bkmOrt.Selected, bkmRegion.Selected, bkmBetreff.Selected, _
    bkmAnrede.Selected

    'Die Textmarke in einer globalen Variablen festhalten.
    selectedBkm = e.Selection.Bookmarks(1)
    'Den Inhalt der gewählten Textmarke markieren
    selectedBkm.Select()
    'Die Textmarke bis zum Endpunkt des Bereichs erweitern.
    selectedBkm.End = selectedBkm.End + 1

    If Not selectedAusgeführt And Not deselectedAusgeführt Then
        If deselectedBkm Is Nothing Then
            'Eine Textmarke wird erstmals markiert
            deselectedBkm = selectedBkm
        Else
            'Sonst wurde Select vor Deselect ausgeführt
            deselectedAusgeführt = False
            selectedAusgeführt = True
        End If
    ElseIf selectedAusgeführt = False And deselectedAusgeführt = True Then
        'Deselect wurde zuerst ausgeführt
        selectedAusgeführt = False
        deselectedAusgeführt = False
        deselectedBkm = selectedBkm
    Else
        selectedAusgeführt = True
    End If
End Sub

Private Sub bkmDeselect(ByVal sender As System.Object, _
    ByVal e As Microsoft.Office.Tools.Word.SelectionEventArgs) _
```

```

Handles bkmEmpfänger.Deselected, bkmAdresse.Deselected, bkmPLZ.Deselected, _
bkmOrt.Deselected, bkmRegion.Deselected, bkmLand.Deselected, _
bkmBetreff.Deselected, bkmAnrede.Deselected

'Den Textmarkenbereich um ein Zeichen kürzen, so dass sie
'den Text enthält, den der Benutzer eingegeben hat.
deselectedBkm.End = deselectedBkm.End - 1

If selectedAusgeführt = True Then
    'Select wurde zuerst aufgeführt
    Me.deselectedBkm = selectedBkm
    selectedAusgeführt = False
    deselectedAusgeführt = False
ElseIf selectedAusgeführt = False Then
    'Deselect wird zuerst ausgeführt
    deselectedAusgeführt = True
End If
End Sub

```

Die Benutzerschnittstellen einer VSTO-Lösung

Ob VBA-Makros oder VSTO-Code: wir erwarten, dass der Benutzer interaktiv mit der programmierten Lösung zusammenarbeitet. Neben den Dialogfeldern und Office-Symboleisten (CommandBars) steht dem VSTO-Entwickler für dokumentspezifische Lösungen der Aufgabenbereich *Dokumentaktionen* als zusätzliche Schnittstelle zur Verfügung.

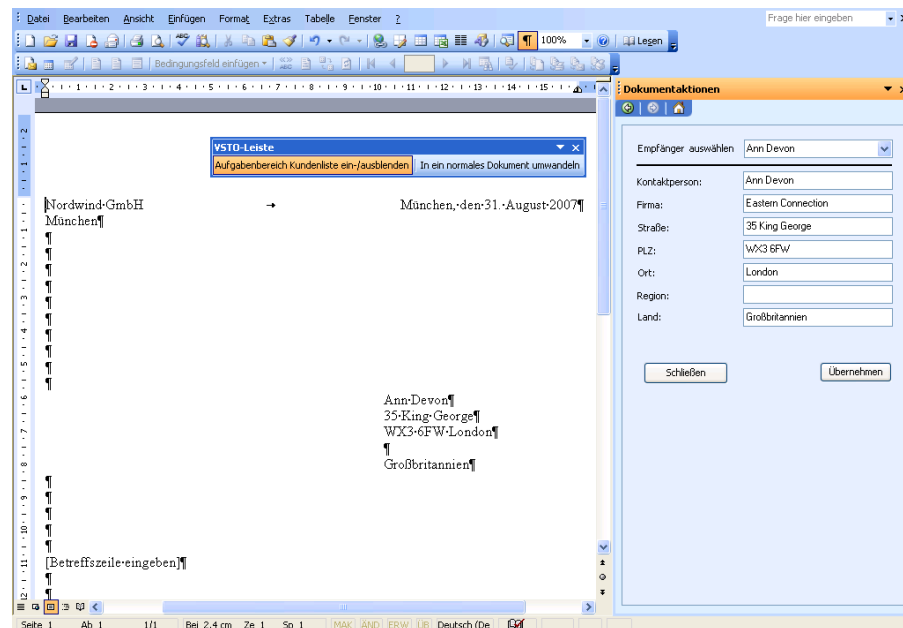
Eine weitere Methode, auf Benutzerhandlungen zu reagieren, bilden die Word- und VSTO-Ereignisse, wie jene für Textmarken, die im vorherigen Abschnitt vorgestellt wurden.

Egal, welche Schnittstellen der Entwickler wählt, die Grundlagen dafür werden in der Prozedur *ThisDocument_Startup* gelegt. Und wo erforderlich, werden sie in der Prozedur *ThisDocument_Shutdown* außer Kraft gesetzt.

Die Beispiellösung zeigt auf, wie eine Symbolleiste mit zwei Symbolschaltflächen sowie ein *Dokumentaktionen*-Aufgabenbereich angelegt werden, wie in Abbildung 10.8 zu sehen. Die erste Symbolschaltfläche schaltet den Aufgabenbereich ein und aus. Die zweite entfernt die VSTO-Lösung aus dem Dokument, so dass Word-Benutzer ohne Zugang zur Lösung störungsfrei damit arbeiten können.

Dieser Aufgabenbereich ermöglicht die Auswahl einer Kundenadresse aus der Firmendatenbank und deren nachträgliche Bearbeitung, bevor die Informationen in die Textmarken eingefügt werden.

Abbildung 10.8: VSTO-Dokumentlösung mit Symbolleiste und Dokumentaktionen-Aufgabenbereich



Symbolleisten und Symbolschaltflächen anlegen

Die Handhabung der CommandBar-Objekte und -Eigenschaften von Office in VSTO unterscheidet sich nicht wesentlich von jener unter VBA (siehe Kapitel 16). Das Listing 10.2 zeigt die Prozedur *ThisDocument_Startup*, die beim Laden des Dokuments ausgeführt wird. Standardmäßig gilt das VSTO-Dokument (im Beispiel die Dokumentvorlage) als Speicherort (CustomizationContext) für alle im Code vorgenommenen Änderungen; es ist nicht notwendig, diesen ausdrücklich anzugeben.

Alle Symbolleistenobjekte werden auf Klassenebene als globale Variablen deklariert. Somit haben sie Gültigkeit, bis das Projekt aus dem Speicher entfernt wird, und werden nicht vorzeitig durch Garbage Collection entfernt. Falls weitere Klassen in der VSTO-Lösung darauf zugreifen, sollen sie als »Friend« oder »Public« deklariert werden.

Die OnAction-Eigenschaft, die eine Symbolschaltfläche mit ihrem VBA-Makro verbindet, ist außerhalb einer VBA-Lösung nicht wirksam. Stattdessen wird der Symbolschaltfläche ein Click-Ereignis zugewiesen. Dieses wird ausdrücklich über AddHandler-Codezeilen mit der auszuführenden Prozedur verbunden:

```
AddHandler cbBtnAufgabenbereich.Click, AddressOf AufgabenBereichEinAus
AddHandler cbBtnInNormalesDokWandeln.Click, AddressOf InNormalesDokWandeln
```

Da das Anlegen von Symbolleisten Änderungen in die VSTO-Vorlage schreibt, wird Word den Benutzer irgendwann fragen, ob er diese speichern möchte. Um solche Meldungen zu unterbinden, wird am Schluss der Prozedur die Saved-Eigenschaft der Vorlage auf »Wahr« gesetzt: *ThisTemplate.Saved = True*. Auch Speicheraufforderungen wegen Änderungen im Dokument selbst werden ebenso wie für die Vorlage *Normal.dot* im Beispielcode unterbunden. Möglicherweise müssen Sie diese Zeilen wiederholt in der Lösung verwenden, um die lästigen Meldungen zu

vermeiden.

Listing 10.2: Eine Symbolleiste in ThisDocument_Startup erstellen

```
Imports System.Windows.Forms
Imports System.Data
Imports Word = Microsoft.Office.Interop.Word
Imports Tools = Microsoft.Office.Tools
Imports Office = Microsoft.Office.Core

'Globale Variablen
Dim cb As Office.CommandBar          'Symbolleiste
Friend cbBtnAufgabenbereich As Office.CommandBarButton 'Symbolschaltfläche
Dim cbBtnInNormalesDokWandeln As Office.CommandBarButton

Private Sub ThisDocument_Startup(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles Me.Startup

    Try
        'Die VSTO-Symbolleiste erstellen...
        Me.cb = ThisApplication.CommandBars.Add(Name:="VSTO-Leiste", _
            Position:=Office.MsoBarPosition.msoBarFloating, MenuBar:=False)
        '...und mit Schaltflächen bestücken
        Me.cbBtnAufgabenbereich = CType(cb.Controls.Add( _
            Type:=Office.MsoControlType.msoControlButton), Office.CommandBarButton)
        With Me.cbBtnAufgabenbereich
            .Style = Microsoft.Office.Core.MsoButtonStyle.msoButtonCaption
            .Caption = "Aufgabenbereich Kundenliste ein-/ausblenden"
            .Tag = "AufgabenbereichEinAus"
            .State = Microsoft.Office.Core.MsoButtonState.msoButtonUp
            .Visible = True
        End With
        Me.cbBtnInNormalesDokWandeln = CType(cb.Controls.Add( _
            Type:=Office.MsoControlType.msoControlButton), Office.CommandBarButton)
        With Me.cbBtnInNormalesDokWandeln
            .Style = Microsoft.Office.Core.MsoButtonStyle.msoButtonCaption
            .Caption = "In ein normales Dokument umwandeln"
            .Tag = "InNormalesDokumentWandeln"
            .Visible = True
            .BeginGroup = True
        End With
        Me.cb.Visible = True

        AddHandler cbBtnAufgabenbereich.Click, AddressOf AufgabenBereichEinAus
        AddHandler cbBtnInNormalesDokWandeln.Click, AddressOf InNormalesDokWandeln

        'Sicherstellen, dass das Dokument in der korrekten Ansicht erscheint
        Me.ActiveWindow.View.Type = Word.WdViewType.wdPrintView
        'dass diese Symbolleiste nicht sichtbar ist,
        ThisApplication.CommandBars("Control Toolbox").Visible = False
        'dass die Textmarken nicht sichtbar sind und,
        Me.ActiveWindow.View.ShowBookmarks = False
        'dass keine Speicheraufforderung erscheint, falls
        'der Benutzer das Dokument sofort schließt
```

```
'Die Daten einlesen
kundenDaten = New KundenListe
kundenDaten.ReadXml(Me.AttachedTemplate.Path & "\KundenDaten.xml")

'Speicheraufforderungen unterbinden
ThisTemplate = Me.AttachedTemplate
ThisTemplate.Saved = True
Me.Saved = True
Catch ex As System.Exception
    MessageBox.Show(ex.Message)
End Try

InitializeDataCache()
ThisApplication.NormalTemplate.Saved = True
End Sub
```

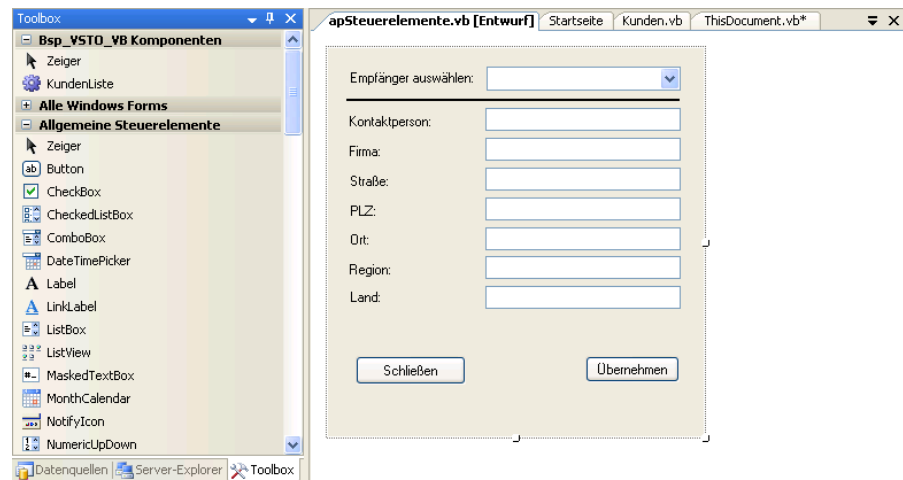
Der Aufgabenbereich *Dokumentaktionen*

Der von VSTO zur Verfügung gestellte Aufgabenbereich *Dokumentaktionen* ist eigentlich Teil der Smart Document-Funktionalität (siehe Kapitel 24). Die Nachfrage seit Erscheinen von Office XP nach einem frei definierbaren Aufgabenbereich war jedoch so groß, dass das VSTO-Team diese Funktionalität integrierte, um VSTO-Entwicklern einen Aufgabenbereich zur Verfügung zu stellen. VSTO übernimmt die mühsame Arbeit, ein XML-Schema und -Manifest zu erstellen, diese mit dem Dokument zu verbinden und sie dynamisch zu verwalten. Der Entwickler kann sich voll auf die Gestaltung und die programmtechnischen Aspekte der Schnittstelle konzentrieren.

Neben HTML-Steuerelementen unterstützt der VSTO-Aufgabenbereich auch WinForms-Steuerelemente. VSTO packt diese in Aufgabenbereich-gerechte COM-ActiveX-Steuerelemente ein. Obwohl es möglich ist, zur Laufzeit einzelne WinForms-Steuerelemente dem Aufgabenbereich hinzuzufügen, stellen die meisten Entwickler bald fest, dass diese Vorgehensweise etwas mühsam ist. Die Positionierungs- sowie Formatierungsmöglichkeiten sind beschränkt und das Ergebnis ist oft wenig zufrieden stellend.

Optimaler ist es, ein WinForms-Benutzersteuerelement im Projekt zu erstellen und dieses in den Aufgabenbereich einzubinden. Damit entfällt die Definition mehrerer einzelner Steuerelemente bei Laufzeit – sie können im Entwurfsmodus erstellt und formatiert werden. Ein Benutzersteuerelement fügen Sie dem Projekt über die Befehlsfolge *Projekt/Benutzersteuerelement hinzufügen* hinzu. Im Dialogfeld *Neues Element hinzufügen* stellen Sie sicher, dass der Eintrag *Benutzersteuerelement* markiert ist, und geben ihm dann einen aussagekräftigen Namen. Nach Betätigung der Schaltfläche *Hinzufügen* erscheint die leere Entwurfsfläche, die wie ein Windows-Formular mit Steuerelementen aus der Toolbox ergänzt wird (Abbildung 10.9). Auch das Eigenschaftfenster steht wie üblich bei der Arbeit mit WinForms zur Verfügung.

Abbildung 10.9: Das Benutzersteuerelement für den VSTO-Aufgabenbereich



Der Code, der den Aufgabenbereich initialisiert und einblendet, ist in Listing 10.3 ersichtlich. Da diese Handlungen ebenfalls Speicheraufforderungen für die *Normal.dot* verursachen, enthält die Prozedur Code, um den erwähnten Meldungen zuvorzukommen. Hat der Benutzer jedoch eigene Anpassungen vorgenommen, sollen diese nicht verworfen werden. Deshalb wird am Anfang der Status der *Saved*-Eigenschaft festgehalten und am Ende die *Saved*-Eigenschaft entsprechend angewendet oder nicht.

Bei der Einblendung des Aufgabenbereichs müssen drei Objekte berücksichtigt werden: das im Word-Aufgabenbereich eingebettete VSTO-Steuerelement, die der Lösung dienenden Steuerelemente (das Benutzersteuerelement) sowie der eigentliche Word-Aufgabenbereich. Diese werden in den globalen Variablen *ap*, *apControls* beziehungsweise *tp* festgehalten. Im Normalfall müssen sie alle nur einmal initialisiert werden. Deshalb testet *AufgabenbereichEinAus* zuerst, ob *ap* sowie *tp* gleich *Nothing* sind. Entsprechend wird die Prozedur *AufgabenbereichInitialisieren* aufgerufen.

Da einer VSTO-Dokumentlösung nur ein möglicher Aufgabenbereich zur Verfügung steht, existiert dieser schon, wir müssen ihn lediglich ansprechen: *Me.ap* = *Me.ActionsPane*. Beim Benutzersteuerelement hingegen handelt es sich um eine Klasse, von der ein neues Objekt erstellt werden muss: *Me.apControls* = *New apSteuerelemente*. Nach Instanziierung des Benutzersteuerelements kann es dem VSTO-Aufgabenbereich hinzugefügt werden: *Me.ap.Controls.Add(apControls)*. Schließlich wird der Word-Aufgabenbereich seiner Variablen zugewiesen: *Me.tp* = *ThisApplication.TaskPanes(Word.WdTaskPanes.wdTaskPaneDocumentActions)*. Anschließend ruft die Prozedur eine weitere auf, um die Steuerelemente mit Daten zu füllen. Darauf kommen wir im nächsten Abschnitt zurück.

Profitipp Beginn

In einer VSTO-Dokumentlösung kann auf die Word- (oder Excel-)Anwendung über das Objekt *ThisApplication* direkt zugegriffen werden. *Me* (oder in C# *this*) stellt die VSTO-Lösung dar, die ihrerseits die meisten Dokumentmethoden und -Eigenschaften zur Verfügung stellt (implementiert). Solche sind jedoch nicht mit

den »echten« Eigenschaften und Methoden zu verwechseln und verhalten sich gelegentlich anders, was zu unerwarteten Ergebnissen führt. Falls Sie ein Word-Objekt direkt ansprechen möchten, geht das über die Eigenschaft `InnerObject`. Um beispielsweise das Word-Dokument (statt die VSTO-Lösung) anzusprechen: `Dim wdDoc As Word.Document = Me.InnerObject`. Als weiteres Beispiel sehen Sie die Prozedur *FillBookmark* in Listing 10.5.

Profitipp Ende

Da die Initialisierung des VSTO-Aufgabenbereichs ihn auch einblendet, testet der Code, ob dies gerade erfolgt ist. Falls nicht, wird er »getoggelt«. Danach wird durch die Prozedur *ToggleButtonState* der Status der Symbolschaltfläche entsprechend festgelegt: Falls der Aufgabenbereich eingeblendet ist, sieht sie »gedrückt« aus, sonst nicht.

Letztlich wird der Fokus in den eingeblendeten Aufgabenbereich gesetzt. Die Methode `SetFocus` aktiviert lediglich seine Titelleiste. Um ein darin enthaltenes Steuerelement anzuspringen, steht nur `SendKeys` zur Verfügung, was nicht gerade »schön« ist, aber es funktioniert (meistens).

Listing 10.3: Die Betätigung der entsprechenden Symbolschaltfläche führt diese Prozedur aus

```
'Globale Variablen auf Klassenebene
Dim ap As Tools.ActionsPane          'Dokumentaktionen-Aufgabenbereich
Dim apControls As apSteuerelemente 'Benutzerdefiniertes Steuerelement
Friend tp As Word.TaskPane           'allgemeiner Word-Aufgabenbereich

Private Sub AufgabenBereichEinAus(ByVal ctrl As Office.CommandBarButton, _
    ByRef Cancel As Boolean)

    'Hält fest, ob die Normal.dot nicht gespeicherte Daten enthält.
    Dim normalTemplateSaved As Boolean = True
    Dim NormalDot As Word.Template = ThisApplication.NormalTemplate
    Dim apInitialisiert As Boolean = False
    Try
        'Nach Ausführung dieser Prozedur gibt es Änderungen in der NormalTemplate.
        'Falls bislang keine vorhanden sind, die gespeichert werden sollen,
        'wird am Schluss die Abfrage nach Speicherung von Änderungen unterdrückt.
        normalTemplateSaved = NormalDot.Saved

        'Falls der Aufgabenbereich Dokumentaktionen noch nicht initialisiert wurde ...
        If Me.ap Is Nothing Or Me.tp Is Nothing Then
            '...dies tun, ihm das Steuerelement zufügen, und anzeigen.
            apInitialisiert = AufgabenbereichInitialisieren()
        End If
        'Wurde der Aufgabenbereich gerade initialisiert ist er sichtbar
        'und soll nicht umgehend wieder unsichtbar gemacht werden.
        If Not apInitialisiert Then tp.Visible = Not tp.Visible

        'Die Symbolschaltfläche der Sichtbarkeit des Aufgabenbereichs anpassen
        ToggleButtonState(cbBtnAufgabenbereich)

        'Dem Aufgabenbereich den Fokus geben
        If Me.tp.Visible Then
            ThisApplication.CommandBars("Task Pane").Controls(1).SetFocus()
            System.Windows.Forms.SendKeys.Send("{Down}")
        End If
    Catch ex As Exception
        'Fehlerbehandlung
    End Try
End Sub
```

```

        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.apControls = New apSteuerelemente
        Me.ap.Controls.Add(apControls)
        Me.tp = ThisApplication.TaskPanes(Word.WdTaskPanes.wdTaskPaneDocumentActions)
        AufgabenbereichDatenEinbinden(apControls)
        success = True
    Catch ex As Exception
        MessageBox.Show(ex.Message)
    End Try
    Return success
End Function

Friend Sub ToggleButtonState(ByVal btn As Office.CommandBarButton)
    Select Case btn.Tag
        Case "AufgabenbereichEinAus"
            If Me.tp.Visible Then
                Me.cbBtnAufgabenbereich.State = Office.MsoButtonState.msoButtonDown
            Else
                Me.cbBtnAufgabenbereich.State = Office.MsoButtonState.msoButtonUp
            End If
    End Select
End Sub

```

Datenquelle in den Aufgabenbereich einbinden

Als Datenquelle für die Beispiellösung dient eine XML-Datei, die die Kundenliste aus der *Nordwind.mdb*-Datenbank enthält. Diese wird mit dem VSTO-Dokument in den gleichen Ordner kopiert. Für diese Daten wurde ein »Typed Dataset« (die Klasse *KundenListe*) erstellt, das während *ThisDocument_Startup* instanziiert wird:

```

kundenDaten = New KundenListe
kundenDaten.ReadXml(Me.AttachedTemplate.Path & "\KundenDaten.xml")

```

Die Prozedur *AufgabenbereichDatenEinbinden* in Listing 10.4 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 10.4: Die Steuerelemente im Aufgabenbereich mit der Datenquelle verbinden

```

'Variable auf Klassenebene
Dim kundenDaten As KundenListe

Private Sub AufgabenbereichDatenEinbinden(ByVal apInnerControl As Windows.Forms.Control)

```



```

Dim tKunden As DataTable = kundenDaten.Kunde
Dim dvKunden As New DataView(tKunden)
Dim cboKundenListe As ComboBox

Try
    dvKunden.Sort = "Kontaktperson"
    'Combobox mit Daten füllen
    cboKundenListe = CType(apInnerControl.Controls("KontaktpersonComboBox"), ComboBox)
    cboKundenListe.DataSource = dvKunden
    cboKundenListe.DisplayMember = "Kontaktperson"

    'Die Textboxen mit der Datenquelle verknüpfen
    apInnerControl.Controls("KontaktpersonTextbox").DataBindings.Add("Text", dvKunden, _
        "Kontaktperson")
    apInnerControl.Controls("FirmaTextbox").DataBindings.Add("Text", dvKunden, "Firma")
    apInnerControl.Controls("StraßeTextbox").DataBindings.Add("Text", dvKunden, "Straße")
    apInnerControl.Controls("PLZTextbox").DataBindings.Add("Text", dvKunden, "PLZ")
    apInnerControl.Controls("OrtTextbox").DataBindings.Add("Text", dvKunden, "Ort")
    apInnerControl.Controls("RegionTextbox").DataBindings.Add("Text", dvKunden, "Region")
    apInnerControl.Controls("LandTextbox").DataBindings.Add("Text", dvKunden, "Land")
Catch ex As Exception
    MessageBox.Show(ex.Message)
End Try
End Sub

```

Im Aufgabenbereich angezeigte Daten ins Dokument schreiben

Der Aufgabenbereich enthält zwei Schaltflächen. Die eine schließt den Aufgabenbereich und passt den Status der Symbolschaltfläche entsprechend an. Die andere schreibt den Inhalt der Textboxen in die Textmarken des Dokuments. Listing 10.5 veranschaulicht den Vorgang. Falls die Textmarke im Dokument noch vorhanden ist, bekommt sie den Text der entsprechenden Textbox, ansonsten wird eine Meldung eingeblendet.

Profitipp Beginn

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.

Profitipp Ende

Listing 10.5: Im Aufgabenbereich enthaltene Daten in die Dokument-Textmarken schreiben

```

Private Sub btnDatenUebernehmen_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles btnDatenUebernehmen.Click

    With Globals.ThisDocument
        If Not FillBookmark(.bkmEmpfänger, Me.KontaktpersonTextBox) Then _
            MissingBookmark(.bkmEmpfänger.Name)
        If Not FillBookmark(.bkmAdresse, Me.StraßeTextBox) Then _
            MissingBookmark(.bkmAdresse.Name)
        If Not FillBookmark(.bkmPLZ, Me.PLZTextBox) Then _
            MissingBookmark(.bkmPLZ.Name)
        If Not FillBookmark(.bkmOrt, Me.OrtTextBox) Then _

```

```

        MissingBookmark(.bkmOrt.Name)
    If Not FillBookmark(.bkmRegion, Me.RegionTextBox) Then _
        MissingBookmark(.bkmRegion.Name)
    If Not FillBookmark(.bkmLand, Me.LandTextBox) Then _
        MissingBookmark(.bkmLand.Name)
    End With
End Sub

Friend Function FillBookmark(ByVal bkm As Tools.Word.Bookmark, _
    ByVal txt As Windows.Forms.TextBox) As Boolean

    Dim doc As Tools.Word.Document = Globals.ThisDocument
    Dim success As Boolean = False
    Try
        If doc.Bookmarks.Exists(bkm.InnerObject.Name) Then
            bkm.Text = txt.Text
            success = True
        End If
    Catch ex As Exception
        MessageBox.Show(ex.Message)
    End Try
    Return success
End Function

Sub MissingBookmark(ByVal bkmName As String)
    MessageBox.Show("Die Textmarke " & bkmName & _
        " ist im Dokument nicht vorhanden. Die Daten konnten nicht eingefügt werden.")
End Sub

```

Der Datenaustausch bei geschlossenem Dokument

Wie eingangs erwähnt, bietet VSTO die Möglichkeit, Daten in ein geschlossenes Dokument zu schreiben beziehungsweise aus einem geschlossenen zu lesen. Das im Dokument eingebettete VSTO-ActiveX-Steuerelement stellt diese Funktionalität über einen Daten-Cache (Zwischenspeicher) zur Verfügung.

Das Listing 10.6 veranschaulicht, wie der Daten-Cache eines VSTO-Dokuments initialisiert wird. In der Beispiellösung werden die Werte mehrerer Variablen des Datentyps String zwischengespeichert. (Die Funktionalität unterstützt zusätzliche Datentypen wie Datasets und DataTables; Voraussetzung ist, dass der Datentyp mit System.Xml.Serialization in XML umgewandelt werden kann. Zudem müssen die Variablen (oder Properties) als «Public» deklariert werden.)

Am Ende der Prozedur *ThisDocument_Startup* (Listing 10.2) steht die Kommandozeile *InitializeDataCache()*. Diese Methode stellt sicher, dass die Variablen für den Zwischenspeicher initialisiert sind und Daten enthalten. (Bleibt ein Teil eines Zwischenspeichers leer, wird er als beschädigt betrachtet und die Daten können nicht gelesen werden. Es ist deshalb wichtig, sicherzustellen, dass alle Elemente des Zwischenspeichers Werte enthalten.)

Beim Speichern des VSTO-Dokuments wird das Ereignis *ThisDocument_BeforeSave* ausgeführt. Diese Prozedur sorgt dafür, dass der Inhalt der Textmarken (sofern sie im Dokument noch vorhanden sind) in die

Variablen des Zwischenspeichers geschrieben werden. Fehlt eine Textmarke, wird stattdessen eine entsprechende Nachricht hinterlegt.

Listing 10.6: Den Zwischenspeicher initialisieren und Daten hinein schreiben

```
'''Variablen für das Daten-Cache
<Cached()> Public CachedEmpfänger As String
<Cached()> Public CachedAdresse As String
<Cached()> Public CachedPLZ As String
<Cached()> Public CachedOrt As String
<Cached()> Public CachedLand As String
<Cached()> Public CachedBetreff As String

Private Sub InitializeDataCache()
    If (Not Me.IsCached("CachedEmpfänger")) Or (CachedEmpfänger Is Nothing) Then
        CachedEmpfänger = "Empfänger nicht bekannt"
    End If
    If (Not Me.IsCached("CachedAdresse")) Or (CachedAdresse Is Nothing) Then
        CachedAdresse = "Adresse nicht bekannt"
    End If
    If (Not Me.IsCached("CachedPLZ")) Or (CachedPLZ Is Nothing) Then
        CachedPLZ = "PLZ nicht bekannt"
    End If
    If (Not Me.IsCached("CachedOrt")) Or (CachedOrt Is Nothing) Then
        CachedOrt = "Ort nicht bekannt"
    End If
    If (Not Me.IsCached("CachedLand")) Or (CachedLand Is Nothing) Then
        CachedLand = "Land nicht bekannt"
    End If
    If (Not Me.IsCached("CachedBetreff")) Or (CachedBetreff Is Nothing) Then
        CachedBetreff = "Betreff nicht bekannt"
    End If
End Sub

Private Sub ThisDocument_BeforeSave(ByVal sender As Object, _
    ByVal e As Microsoft.Office.Tools.Word.SaveEventArgs) Handles Me.BeforeSave
    'Daten in die Daten-Cache schreiben
    CacheTextmarkenDaten(CachedEmpfänger, Me.bkmEmpfänger)
    CacheTextmarkenDaten(CachedAdresse, Me.bkmAdresse)
    CacheTextmarkenDaten(CachedPLZ, Me.bkmPLZ)
    CacheTextmarkenDaten(CachedOrt, Me.bkmOrt)
    CacheTextmarkenDaten(CachedLand, Me.bkmLand)
    CacheTextmarkenDaten(CachedBetreff, Me.bkmBetreff)
End Sub

Private Function CacheTextmarkenDaten(ByRef CacheVariable As Object, _
    ByVal bkm As Tools.Word.Bookmark) As String

    Dim success As Boolean = False
    If Not bkm Is Nothing Then
        Try
            CacheVariable = bkm.Text
            success = True
        Catch ex As Exception
```

```

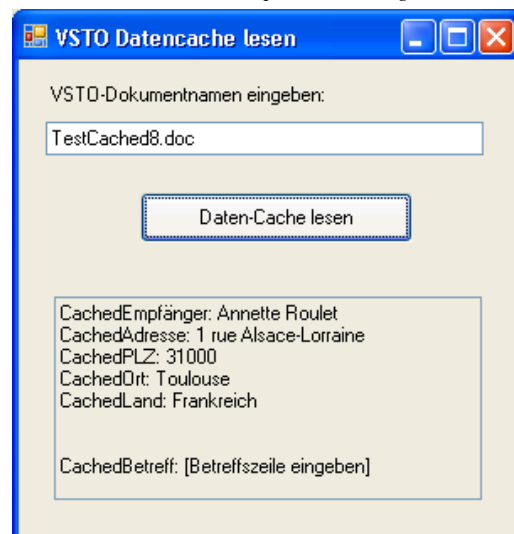
        MessageBox.Show(ex.Message)
        CacheVariable = "Fehler beim Schreiben."
    End Try
Else
    CacheVariable = "Textmarke nicht vorhanden."
End If
Return success
End Function

```

Von außerhalb des Dokuments greift die »ServerDocument«-Funktionalität des Namespaces `Microsoft.VisualStudio.Tools.Applications.Runtime` auf den Dokumentzwischenspeicher zu. Dieser Namespace ist nicht Teil der `Microsoft.Office.Tools-DLL`, die der Entwickler in einer VSTO-Dokumentlösung benutzt. Er befindet sich vielmehr in der Runtime-DLL, die auf dem Benutzerrechner installiert werden muss, um VSTO-Lösungen zu unterstützen. Diese DLL kann auch auf einem Netzlaufwerk oder einem Web-Server installiert werden.

Um die Funktionalität zu veranschaulichen, wurde eine kleine WinForms-Anwendung (siehe Abbildung 10.10) erstellt. Den zugehörigen Code finden Sie in Listing 10.7. Der Dateiname eines VSTO-Dokuments in einem im Code festgelegten Ordner wird in das erste Textfeld eingegeben. Die Betätigung der Schaltfläche *Daten-Cache lesen* schreibt die zwischengespeicherten Informationen in das zweite Textfeld.

Abbildung 10.10: Die Daten aus dem Zwischenspeicher eines geschlossenen VSTO-Dokuments anzeigen



Falls das Dokument im vorgegebenen Pfad (*Eigene Dokumente*) vorhanden ist, wird ein neues `ServerDocument`-Objekt (`serverDok`) angelegt. Sein Zwischenspeicher wird angesprochen (`serverDok.CachedData`), dann die Auflistung aller sich darin befindenden »HostItems«. (Word hat nur ein »HostItem« – das Dokument. Excel hat für die Arbeitsmappe sowie jedes Arbeitsblatt eins.) Jedes »HostItem« in der Auflistung kann mehrere »DataItems« (Datenpunkte oder -elemente) enthalten. Es wird durch alle Datenpunkte 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 Textmarken befindet. Das Ergebnis wird der Zeichenkette `cachedInhalt` hinzugefügt, die am Ende der Prozedur `btnDatenLesen_Click` in das zweite Textfeld geschrieben wird.

Listing 10.7: Daten aus dem Zwischenspeicher eines geschlossenen VSTO-Dokuments lesen

```
Imports Tools = Microsoft.VisualStudio.Tools.Applications.Runtime

Public Class Form1

Private Sub btnDatenLesen_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles btnDatenLesen.Click

    Dim dokPfad As String = System.Environment.GetFolderPath( _
        Environment.SpecialFolder.MyDocuments) & "\"
    Dim serverDok As Tools.ServerDocument
    Try
        Dim dokName As String = Me.txtDokumentName.Text
        If dokName.Length > 0 Then
            Dim dokPfadMitName = dokPfad & dokName
            If System.IO.File.Exists(dokPfadMitName) Then
                serverDok = New Tools.ServerDocument(dokPfadMitName)
                Dim datenAusDemCache As Tools.CachedData = serverDok.CachedData
                Dim datenHostItems As Tools.CachedDataHostItemCollection = _
                    datenAusDemCache.HostItems
                Dim datenHostItem As Tools.CachedDataHostItem
                Dim datenPunkt As Tools.CachedDataItem
                Dim cachedInhalt As String = ""
                For Each datenHostItem In datenHostItems
                    For Each datenPunkt In datenHostItem.CachedData
                        cachedInhalt = cachedInhalt & datenPunkt.Id & ": " & _
                            ExtractTextAusXML(datenPunkt.Xml) & vbCrLf
                    Next
                Next
                'Zeigt den Dateninhalt im Fenster an
                Me.txtDaten.Text = cachedInhalt
            Else
                MessageBox.Show("Das Dokument '" & dokName & "' konnte im Pfad '" & dokPfad & _
                    & "' nicht gefunden werden.")
            End If
        Else
            MessageBox.Show("Bitte oben einen Dokumentnamen eingeben.")
        End If
    Catch ex As Exception
        MessageBox.Show(ex.Message)
    Finally
        If Not serverDok Is Nothing Then serverDok.Close()
    End Try
End Sub

Private Function ExtractTextAusXML(ByVal inhalt As String) As String
    Dim inhaltText As String
    Dim xmlDoc As Xml.XmlDocument = New Xml.XmlDocument
```

```
xmlDoc.LoadXml(inhalt)
inhaltText = xmlDoc.LastChild.InnerText
xmlDoc = Nothing
Return inhaltText
End Function
End Class
```

VSTO-Lösung von einem Dokument abtrennen

Wie in der Einleitung zu diesem Abschnitt »Das VSTO-Dokument vorbereiten« erwähnt, ist es möglich, den VSTO-Code vom Dokument zu trennen. Da sich dieser Code im Gegensatz zu VBA nicht im Dokument befindet, geht das relativ problemlos.

Das VSTO-Team hat zwei Methoden bereitgestellt:

- Über das `ServerDocument`-Objekt. Diese Methode ermöglicht das Entfernen sowie Anfügen einer VSTO-Lösung aus einem bzw. an ein Word- oder Excel-Dokument. Dabei wird jedoch die Word- oder Excel-Anwendung gestartet, weshalb es – trotz des Namens – für den Gebrauch auf einem Server nicht geeignet ist. Diese Methode ist ausreichend im VSTO-Teil der MSDN-Seite beschrieben und wird hier nicht näher behandelt.
- Durch die Methode `RemoveCustomization`. Diese wird auf das in Word (oder Excel) geöffnete VSTO-Dokument ausgeführt, wie in Listing 10.8 ersichtlich.

`RemoveCustomization` entfernt das VSTO-ActiveX-Steuerelement aus dem Dokument. Wird es in diesem Zustand geschlossen, bleibt der *DocumentActions*-Aufgabenbereich in der Liste der Aufgabenbereiche aufgeführt. Zudem, falls der Aufgabenbereich jemals eingeblendet wurde, enthält das Dokument weiterhin die Verknüpfung zum XML-Schema *ActionsPane*. Mit der `Clear`-Methode wird der Aufgabenbereich aus der Liste entfernt; anschließend wird die Verknüpfung zum Schema (falls vorhanden) gelöscht.

Nach Ausführung dieser Handlungen besteht durch Ereignisse (sowohl der Anwendung als auch der Symbolschaltflächen) immer noch eine Verbindung zur VSTO-Lösung. Durch Sperren aller von der Lösung benutzten Objekte der *CommandBar*-Auflistung können diese Befehle dem Benutzer entzogen werden. Unter Umständen sollen Ereignis-Prozeduren testen, ob `RemoveCustomization` durchgeführt wurde, und, wenn ja, keine weitere Handlungen durchführen.

Jedes VSTO-Dokument enthält mindestens zwei benutzerdefinierte Dokumenteigenschaften, die es mit der VSTO-Lösung verbinden: `_AssemblyName` sowie `_AssemblyLocation`. Der Wert der ersten ist lediglich ein Sternchen (in VSTO 2003 enthielt es den Namen der Lösung), das dem »Microsoft Office System loader« mitteilt, dass das Dokument mit einer VSTO-Lösung verbunden ist. Der zweite enthält das GUID für das »Runtime Storage Control« – das ActiveX-Steuerelement, das im Dokument als ein *Shape*-Objekt eingebettet ist. Fehlen diese Einträge, wurde die Lösung erfolgreich vom Dokument getrennt.

Hinweis Beginn

Der »Microsoft Office System loader« (auch als »Visual Studio Tools for Office loader« bekannt) ist Bestandteil der Office 2003- und Office 2007-Anwendungen. Er dient als Schnittstelle zwischen dem .NET-Code einer VSTO-Lösung und der

COM-Umgebung von Office. Durch ihn wird die VSTO-Lösung geladen und werden die Ereignisse der Anwendung überwacht und an die Lösung weitergeleitet. Mehr Informationen zum Loader finden Sie auf den Microsoft-Webseiten [http://msdn2.microsoft.com/en-us/library/zcfbd2sk\(VS.80\).aspx](http://msdn2.microsoft.com/en-us/library/zcfbd2sk(VS.80).aspx) sowie [http://msdn2.microsoft.com/en-us/library/aa194021\(office.11\).aspx](http://msdn2.microsoft.com/en-us/library/aa194021(office.11).aspx).

Hinweis Ende

Das Dokument einer VSTO-Vorlagenlösung bleibt weiterhin mit seiner Vorlage verbunden. Falls es mit einer anderen Vorlage, wie *Normal.dot*, verbunden werden soll, geschieht dies am besten beim Schließen des Dokuments. Dafür wird das Ereignis `ThisApplication_DocumentBeforeClose` benutzt.

Listing 10.8: Die VSTO-Lösung von einem Dokument trennen

```
Private Sub InNormalesDokWandeln(ByVal ctrl As Office.CommandBarButton, _
    ByRef Cancel As Boolean)
    Try
        'Entfernt das VSTO ActiveX-Steuerelement aus dem Dokument.
        Me.RemoveCustomization()
        'Den Aufgabenbereich aus der Liste entfernen,
        ActionsPane.Clear()
        'sowie das Schema-Referenz aus dem Dokument,
        'sofern der Aufgabenbereich eingeblendet wurde.
        Dim xmlSchema As Word.XMLSchemaReference
        For Each xmlSchema In Me.XMLSchemaReferences
            If xmlSchema.NamespaceURI = "ActionsPane" Then
                xmlSchema.Delete()
            Exit For
        End If
    Next
    Catch ex As Exception
        MessageBox.Show(ex.Message)
    Finally
        cb.Enabled = False
    End Try
End Sub

Private Sub ThisApplication_DocumentBeforeClose( _
    ByVal Doc As Microsoft.Office.Interop.Word.Document, _
    ByRef Cancel As Boolean) Handles ThisApplication.DocumentBeforeClose
    'Stellt sicher, dass das Dokument die VSTO-Vorlage nicht mehr anpeilt.
    'Gewisse Ereignisse werden jedoch noch ausführbar sein, bis das
    'Dokument geschlossen wurde.

    Dim bVstoDocProp As Boolean = False
    Dim prop As Office.DocumentProperty
    For Each prop In Me.InnerObject.CustomDocumentProperties
        If prop.Name = "_AssemblyName" Or prop.Name = "_AssemblyLocation" Then
            bVstoDocProp = True
        Exit For
    End If
    Next
    If Not bVstoDocProp Then Me.InnerObject.AttachedTemplate = _
```

```
ThisApplication.NormalTemplate  
End Sub
```

VSTO-Dokumentlösungen in Word 2007

Meistens können mit VSTO 2005 erstellte Dokumentlösungen auch in Word 2007 benutzt werden. Dabei ist zu bedenken, dass alle in der Lösung definierten Symbolleisten und -schaltflächen unter der Registerkarte *Add-Ins* erscheinen werden. Es ist nicht möglich, das Ribbon in die Lösung einzubinden; dies wird erst mit der nächsten VSTO-Version möglich sein.

Das Setup-Programm muss die entsprechende VSTO-Runtime verteilen.

Hinweis Beginn

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).

Hinweis Ende

Der Entwickler wird eine Dokumentlösung nicht auf einem Rechner mit installiertem Office 2007-Rechner bearbeiten können. VSTO 2005 lässt sich nur auf einem Rechner mit Office 2003 Professional installieren. Parallelinstallationen mehrerer Office-Versionen sind in diesem Zusammenhang nicht zulässig.

Verteilung von VSTO-Lösungen

Dieses Thema ist sehr komplex und wird hier nicht im Detail behandelt. Dazu stehen seitenlange Beschreibungen im Internet bereit. Wir geben nur einen kurzen Überblick.

Hinweis Beginn

Deutschsprachige Informationen zum Thema Verteilung (mit Links zu weiteren Artikeln) finden Sie in Jens Häupels Blog: http://blogs.msdn.com/jensha/archive/tags/Setup+_2600_amp_3B00_+Deployment/default.aspx.

Hinweis Ende

Da VSTO Teil von .NET Framework ist, folgt es dessen Sicherheitsrichtlinien. Wichtigster Punkt für VSTO-Entwickler ist: *allen Teilen* einer VSTO-Lösung muss der Sicherheitsstatus volles Vertrauen (»Full Trust«) zugeordnet werden. Das schließt sowohl die *DLL* als auch das Dokument sowie allfällige »Helfer *dll*« ein. Die Zuteilung der Sicherheitsstufe erfolgt entweder manuell über den *Microsoft .Net Framework Konfiguration*-Assistenten oder im Setup programmtechnisch über »CASPOL« (Code Access Security **POL**icy).

Die .NET 2.0-Sicherheit kennt drei Arten von Beweis, welche Ausführungsrechte .NET-Code zugeteilt wurden: Digitale Signatur, Speicherort (URI) sowie »Strong Name«. Die Sicherheit kann auf einer der drei Arten oder einer beliebigen Kombination basieren. Für den Eigenbedarf ist der Speicherort einfach einzurichten, entspricht jedoch der schwächsten Sicherheitsstufe.

Hinweis Beginn

VSTO 2005 unterstützt keine »ClickOnce«-Verteilung. Diese Funktionalität ist jedoch für die nächste Version vorgesehen.

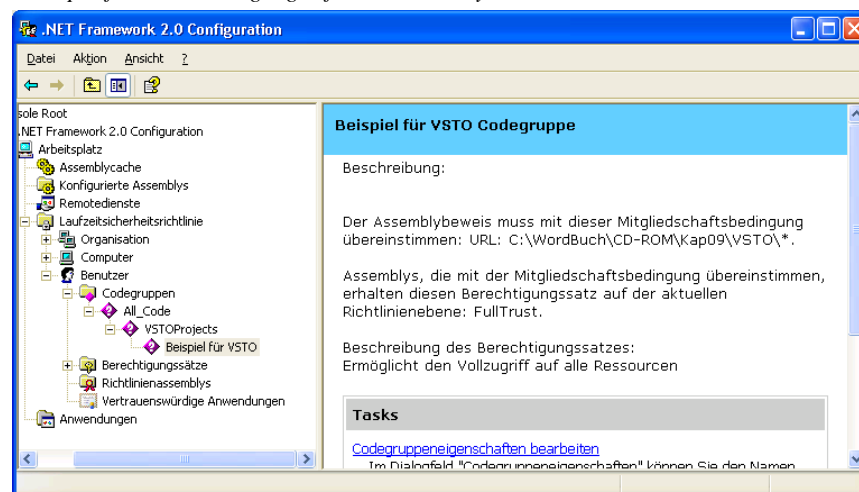
Hinweis Ende

Das Beispiel lauffähig machen

Die folgenden einfachen Schritte sind nur für die Anwendung des Beispiels und als Einführung in die Verteilung gedacht. Visual Studio muss nicht installiert sein, wohl aber das .NET-Framework 2.0 oder 3.0 sowie VSTO Runtime (siehe den Abschnitt »VSTO-Dokumentlösungen in Word 2007«). Sind beide installiert, gehen Sie wie folgt vor:

1. Die Assembly *Bsp_VSTO_VB.dll*, die Lösungsvorlage *Bsp_VSTO_VB.dot* und die XML-Datei *KundenDaten* müssen sich im selben Ordner befinden. Möchten Sie auch das Daten lesen des Zwischenspeichers testen, kopieren Sie die *DatenLesen_VB.exe* ebenfalls in diesen Ordner.
2. In der Systemsteuerung, unter der Rubrik *Verwaltung*, starten Sie den *Microsoft .NET Framework 2.0-Configuration-Assistenten*.
3. Unter *Arbeitsplatz/Laufzeitsicherheitsrichtlinie/Benutzer/Codegruppen* (Abbildung 10.11) wählen Sie den Eintrag *All_Code*. Falls noch kein Eintrag »VSTOProjects« vorhanden ist, erstellen Sie ihn wie folgt:
 - Wählen Sie den Eintrag *All_Code*.
 - Klicken Sie im rechten Fenster *Untergeordnete Codegruppe hinzufügen* an, um den Assistenten zu starten.
 - Tragen Sie im ersten Schritt den Namen *VSTOProjects* ein und klicken Sie auf *Weiter*.
 - Im folgenden Schritt akzeptieren Sie den Vorschlag *Gesamter Code* und klicken auf *Weiter*.
 - Im dritten Schritt legen Sie *Nothing* als *Berechtigungssatz* fest und klicken auf *Weiter*.
 - Im letzten Schritt wählen Sie *Fertig st.*
 - Falls dem Assistenten der Eintrag *Kopie von* vorangestellt ist, klicken Sie ihn mit rechts an, wählen im Kontextmenü *Umbenennen* und korrigieren den Namen.

Abbildung 10.11: Benutzerspezifische Berechtigungen für ein Assembly erstellen



4. Wählen Sie den (neu erstellten) Eintrag *VSTOProjects* und erstellen Sie einen Eintrag für den Ordner, in dem sich das Assembly befindet:
 - Klicken Sie im rechten Fenster auf *Untergeordnete Codegruppe hinzufügen*, um den Assistenten zu starten.
 - Tragen Sie einen aussagekräftigen Namen ein und klicken Sie auf *Weiter*.
 - Im folgenden Schritt wählen Sie *URL* aus der Liste und geben in das eingblendete Textfeld *URL* die vollständige Pfadangabe zur *.dll* ein, evtl. gefolgt von ***. Klicken Sie auf *Weiter*.
 - Im letzten Schritt klicken Sie auf *Fertig st.*
 - Falls dem Assistenten der Eintrag *Kopie von* vorangestellt ist, korrigieren Sie den Namen über den Befehl *Umbenennen* seines Kontextmenüs.

CDROM Beginn

Die vorgestellte Lösung befindet sich in der Datei *Bsp_VSTO.zip* auf der CD-ROM zum Buch im Ordner *\Beilagen\Kap10_VSTO-Dokumentlösung\Bsp*.

Um damit zu arbeiten, öffnen Sie die Lösung *Bsp_VSTO_VB.sln* in Visual Studio 2005 Tools for Office. Drücken Sie (F5), um sie im Debug-Modus auszuführen.

CDROM Ende

Hinweis Beginn

Weitere Informationen zu Office-Entwicklerthemen in deutscher Sprache finden Sie im Blog von

Jens

Häupel

(<http://blogs.msdn.com/jensha/archive/tags/.NET+Dev+mit+Office/default.aspx>)

sowie auf den deutschen MSDN Office Developer-Seiten (<http://www.microsoft.com/germany/msdn/office/default.msp>).

Die englischsprachigen MSDN-Seiten zum Thema VSTO starten auf der Seite [http://msdn2.microsoft.com/en-us/library/23cw517s\(VS.80\).aspx](http://msdn2.microsoft.com/en-us/library/23cw517s(VS.80).aspx).

Außerdem gibt es zwei nützliche Bücher in englischer Sprache:

»VSTO for Mere Mortals« von Kathleen McGrath und Paul Stubbs, herausgegeben von Addison-Wesley.

»Visual Studio Tools for Office« von Eric Carter und Eric Lippert, herausgegeben von Addison-Wesley. Je eine Version für VB.NET sowie C# ist erhältlich.

Hinweis Ende
