HANSER

Leseprobe

Jan Tittel

Office 2010 Programmierung mit VSTO und .NET 4.0

Word, Excel und Outlook erweitern und anpassen

Herausgegeben von Holger Schwichtenberg

ISBN: 978-3-446-42411-1

Weitere Informationen oder Bestellungen unter

http://www.hanser.de/978-3-446-42411-1

sowie im Buchhandel.

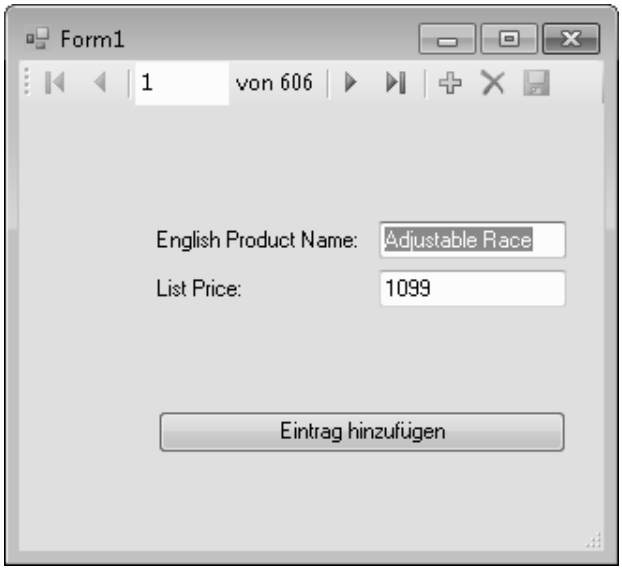

BILD 3.67 Der geänderte Datensatz aus dem Datencache

3.3.2.9 Die fertige Office-Lösung

In diesem Beispiel haben Sie die folgenden Techniken der Office-Programmierung kennengelernt:

- Erstellen von Anpassungen auf Dokumentebene für Arbeitsmappen
- Mit dem Dokumentdesigner arbeiten
- Verwenden von Windows Forms
- Hinzufügen von Windows Forms-Steuerelementen zur Laufzeit
- Einfacher Zugriff auf Arbeitsblätter zur Laufzeit
- Auf Ereignisse reagieren
- Den Datencache verwenden
- Ohne Office auf den Datencache zugreifen

3.4 Praktische Grundlagen von Add-Ins kennenlernen

In diesem Beispiel für Excel kommen die folgenden Techniken beziehungsweise Features von VSTO zum Einsatz:

- Erstellen von Add-Ins auf Anwendungsebene
- Erstellen von Menübändern

- Zugriff auf Menübänder zur Laufzeit
- Erstellen und Anzeigen von benutzerdefinierten Aufgabenbereichen
- Zugriff auf benutzerdefinierte Aufgabenbereiche zur Laufzeit
- Das Office-Menü (Datei-Menü) anpassen
- Dialogfeld-Startprogramme erstellen
- Auf Ereignisse reagieren
- Add-Ins in Office-Anwendungen verwalten

3.4.1 Ein Blick vorab auf das fertige Projekt

Bei diesem Beispiel handelt es sich um ein Add-In auf Anwendungsebene für Excel, anhand dessen einige Grundlagen der Entwicklung von Add-Ins demonstriert werden. Zum Einsatz kommen eine Anpassung des Menübands sowie ein benutzerdefinierter Aufgabenbereich. Dabei liegt ein Schwerpunkt auf das Einblenden und Ausblenden von benutzerdefinierten Aufgabenbereichen sowie auf das Synchronisieren des Zustands zwischen Menüband und Aufgabenbereich.

Praktisches Thema des Beispiels ist die Verwaltung von Kommentaren in Excel. Diese Möglichkeit mag hilfreich sein, da Excel im Vergleich zu Word deutlich weniger Funktionen zum Verwalten von Kommentaren mit sich bringt. Aus dem Menüband heraus kann der Anwender ein Arbeitsblatt auswählen sowie aktivieren und bei Bedarf den benutzerdefinierten Aufgabenbereich mit unterschiedlichen Funktionen für die Verwaltung von Kommentaren einblenden (Bild 3.50).

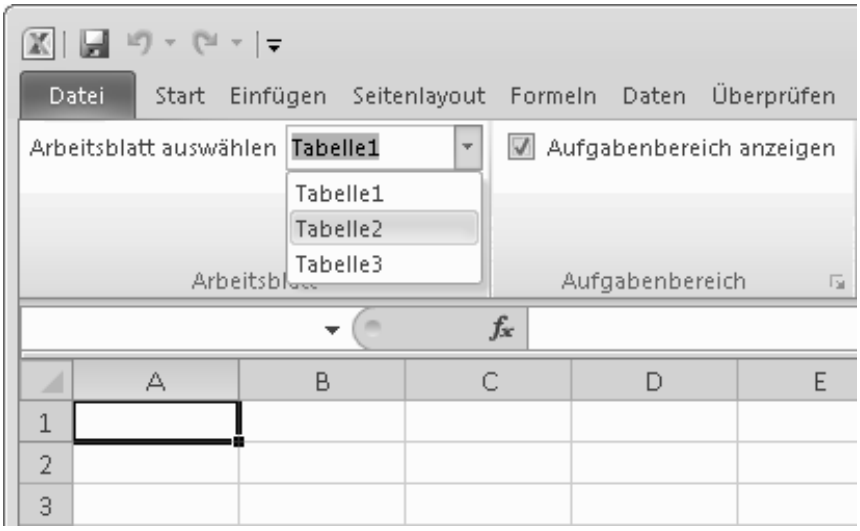

BILD 3.68 Das Menüband des fertigen Beispiels

Die Möglichkeiten zum Verwalten von Kommentaren im benutzerdefinierten Aufgabenbereich umfassen wie auch in Bild 3.51 zu sehen:

- Auslesen der Kommentare aus dem aktiven Arbeitsblatt und deren Anzeige in einer Liste
- Auswählen eines Kommentars in der Liste und einblenden des Kommentars
- Auswählen eines Kommentars in der Liste und ausblenden des Kommentars
- Auswählen eines Kommentars in der Listen und löschen des Kommentars

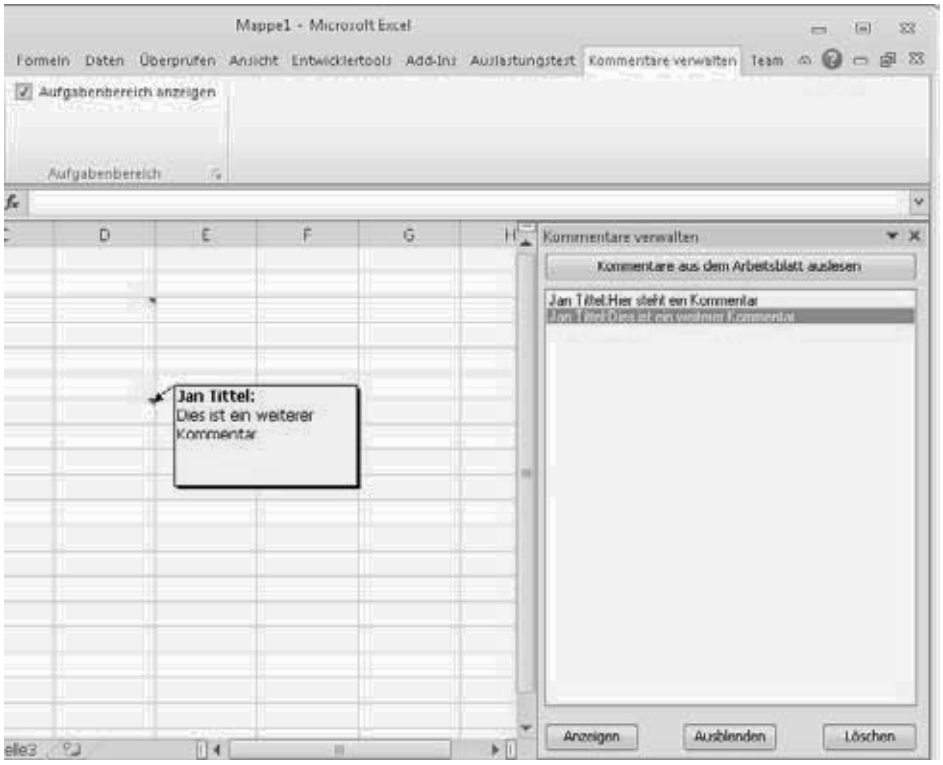

BILD 3.69 Der benutzerdefinierte Aufgabenbereich des fertigen Beispiels

Als kleine Zusatzfunktionalität umfasst das Beispiel ebenso die Erweiterung des Office-Menüs wie das Erstellen von sogenannten *Dialogfeld-Startprogrammen*.

3.4.2 Schrittweise Erstellung des Projekts

In den folgenden Abschnitten wird die Entwicklung des Beispiels Schritt für Schritt zum Nachvollziehen beziehungsweise zum Mitmachen beschrieben.

3.4.2.1 Anlegen und Erkunden eines neues Projekts

Gehen Sie wie folgt vor, um ein neues Add-In auf Anwendungsebene für Excel zu erstellen:

1. Legen Sie in Visual Studio zuerst ein neues Projekt basierend auf der Vorlage *Excel 2010-Add-In* an (Bild 3.52). Geben Sie dem Projekt einen Namen und bestätigen Sie Ihre Angaben mit OK.

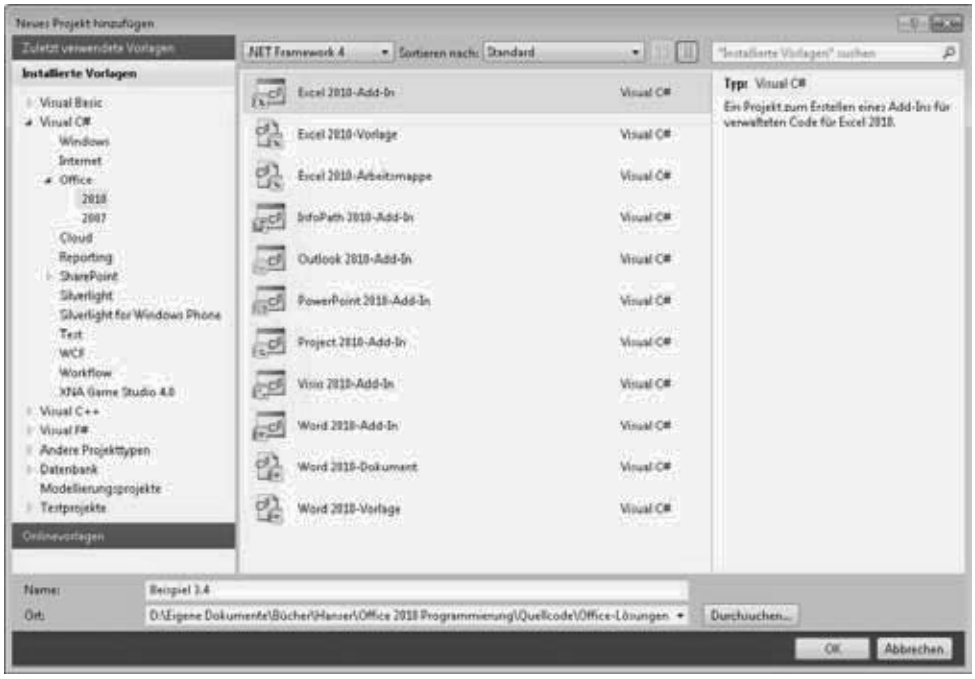

BILD 3.70 Neues Projekt basierend auf der Vorlage Excel 2010-Add-In erstellen

Das Projekt wird nun von Visual Studio erstellt. Wenn Sie die vorherigen Beispiele umgesetzt haben, wirkt die Ausgangssituation eines Add-Ins im Vergleich zu einer Anpassung auf Dokumentebene zunächst etwas nüchtern. So gibt es bei einem Add-In auf Anwendungsebene kein Dokument und keinen Dokumentdesigner, sondern lediglich eine Codedatei *ThisAddIn.cs* (Bild 3.53), die im Editor angezeigt wird. Bei der Klasse *ThisAddIn* handelt es sich um die zentrale Klasse des Projekts, die das Add-In selbst repräsentiert.

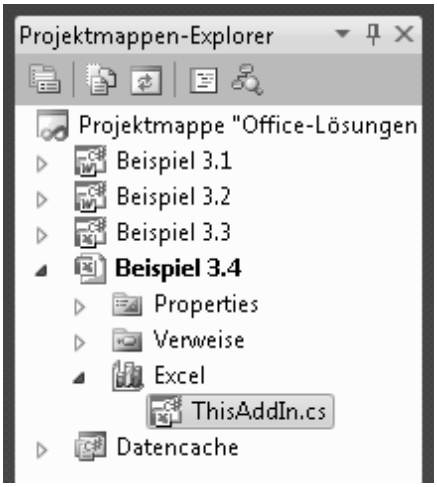

BILD 3.71 Excel-Add-In mit zugeordneter Codedatei im Projektmappen-Explorer

Ein Blick auf die Verweise im Projektmappen-Explorer sowie auf die in der Codedatei eingebundenen Namespaces zeigt, dass Visual Studio bereits alle notwendigen Verweise in das Projekt aufgenommen hat.

```
using Excel = Microsoft.Office.Interop.Excel;
using Office = Microsoft.Office.Core;
using Microsoft.Office.Tools.Excel;
```

Zusätzlich zu den using-Anweisungen finden sich in der Codedatei die beiden noch leeren Ereignis-Behandlungsmethoden *ThisAddIn_Startup* und *ThisAddIn_Shutdown* für die entsprechenden Ereignisse, die beim Starten und Beenden des Add-Ins ausgelöst werden.

```
public partial class ThisAddIn
{
    private void ThisAddIn_Startup(object sender, System.EventArgs e)
    {
    }

    private void ThisAddIn_Shutdown(object sender, System.EventArgs e)
    {
    }

    // Vom VSTO-Designer generierter Code…
}
```

Hierbei handelt es sich um die zentralen Ereignisse eines Add-Ins auf Anwendungsebene, die analog zu einem Konstruktor und Destruktor zu verstehen sind. In der Regel werden Sie zumindest in die *ThisAddIn_Startup*-Methode Anweisungen aufnehmen, um Ihr Add-In zu initialisieren und beispielsweise Datensätze von einer Datenbank abzufragen.

3.4.2.2 Ein Menüband erstellen

Als Nächstes erweitern Sie das Add-In wie folgt um ein eigenes Menüband:

1. Rufen Sie aus dem Menü PROJEKT den Befehl NEUES ELEMENT HINZUFÜGEN auf.
2. Wählen Sie die Vorlage *Menüband (Visueller Designer)* aus und bestätigen Sie mit HINZUFÜGEN, worauf das neue Menüband im Entwurfsmodus geöffnet wird (Bild 3.54).

BILD 3.72 Ein neues Menüband im Entwurfsmodus

Das Menüband verfügt bereits über eine Registerkarte, die wiederum bereits über ein *Group*-Steuerelement verfügt. Standardmäßig handelt es sich um eine integrierte Registerkarte, die von Office automatisch verwaltet wird. Dies ist auch an der Bezeichnung *TabAddIns (Integriert)* zu erkennen.

3. Ändern Sie den Wert der Eigenschaft *ControlIdType* des Menübands von *Office* auf *Custom* (Bild 3.55). Dadurch wird die Menübanderweiterung nicht mehr zusammen mit eventuell anderen vorhandenen Menübanderweiterungen als *integrierte* Registerkarte automatisch durch Office verwaltet, sondern es wird dem Menüband eine eigene *benutzerdefinierte* Registerkarte hinzugefügt.

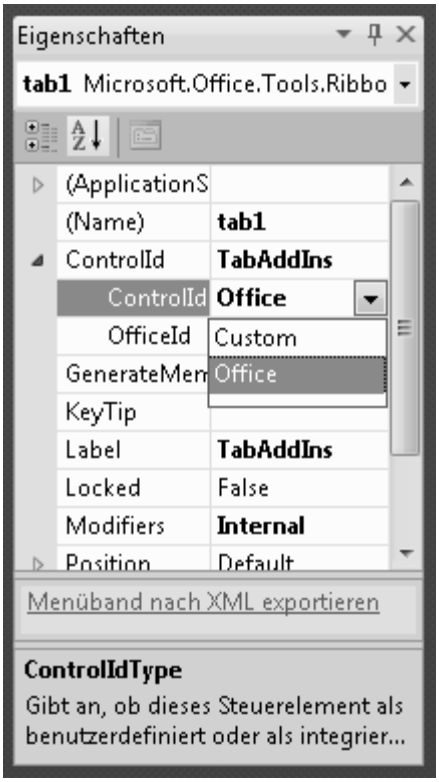

BILD 3.73 Auswahl zwischen integrierter und benutzerdefinierter Registerkarte

4. Fügen Sie dem Menüband aus der Kategorie *Steuerelemente für Office-Menübänder* in der Toolbox ein *ComboBox*-Steuerelement hinzu, indem Sie dieses auf gewohnte Weise per Drag & Drop aus der Toolbox in die Entwurfsoberfläche ziehen. Das Menüband sollte anschließend wie in Bild 3.56 aussehen.

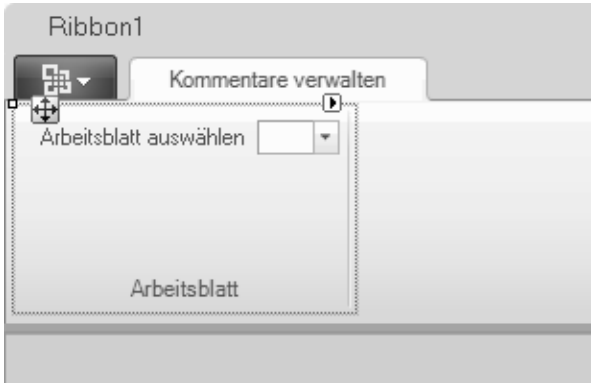

BILD 3.74 Entwurf des Menübands

5. Erstellen Sie mit einem Doppelklick auf das *ComboBox*-Steuerelement die Ereignis-Behandlungsmethode für das *TextChanged*-Ereignis.
6. Fügen Sie der Datei *Ribbon1.cs* eine using-Anweisung für das Objektmodell von Excel hinzu.

```
using Excel = Microsoft.Office.Interop.Excel;
```

7. Nehmen Sie in die Methode *Ribbon1_Load* den Code aus Listing 3.9 auf, um der ComboBox Einträge für die in der Arbeitsmappe enthaltenen Arbeitsblätter hinzuzufügen.

LISTING 3.9 Einträge einer ComboBox hinzufügen

```
private void Ribbon1_Load(object sender, RibbonUIEventArgs e)
{
    Excel.Sheets arbeitsblaetter =
(Excel.Sheets)Globals.ThisAddIn.Application.ActiveWorkbook.Sheets;

    foreach (Excel.Worksheet arbeitsblatt in arbeitsblaetter)
    {
        RibbonDropDownItem element = Factory.CreateRibbonDropDownItem();
        element.Label = arbeitsblatt.Name;
        comboBox1.Items.Add(element);
    }
}
```

Mit der ersten Anweisung wird eine Auflistung vom Typ *Sheets* erstellt, die einen Verweis auf alle in der aktiven Arbeitsmappe enthaltenen Arbeitsblätter erhält. Die einzelnen Arbeitsblätter vom Typ *Worksheet* werden anschließend in einer foreach-Schleife durchlaufen. Dabei wird in jedem Durchgang zunächst ein neues Objekt vom Typ *RibbonDropDownItem* mithilfe der Methode *CreateRibbonDropDownItem* erzeugt. Bei *RibbonDropDownItem* handelt es sich um ein Element, das im weiteren Verlauf der ComboBox hinzugefügt wird. Die Methode *CreateRibbonDropDownItem* stammt aus der *RibbonFactory*-Schnittstelle, die innerhalb einer Menüband-Klasse auch über die Eigenschaft *Factory* zur Verfügung steht. Für die Beschriftung des Elements wird nun der Name des Arbeitsblatts festgelegt, worauf das Element abschließend der Auflistung *Items* der ComboBox mit der Methode *Add* hinzugefügt wird.

HINWEIS: Die *RibbonFactory*-Schnittstelle stellt eine Vielzahl von Methoden bereit, mit denen Steuerelemente zur Anpassung eines Office-Menübands erstellt werden können. Weitere Informationen sowie eine Auflistung der vorhandenen Methoden finden sich in der MSDN Library unter *http://msdn.microsoft.com/de-de/library/microsoft.office.tools.ribbon.ribbonfactory.aspx*.

Das dynamische Erstellen von Menüband-Steuerelementen unterscheidet sich in VSTO 4.0 von der Vorgehensweise in VSTO 3.0, da in VSTO 4.0 einige aus VSTO 3.0 bekannte Klassen durch Schnittstellen ersetzt wurden.

8. Fügen Sie der Methode *comboBox1_TextChanged* nun noch Code hinzu, um das in der ComboBox ausgewählte Arbeitsblatt zu aktivieren.

```
Excel.Worksheet arbeitsblatt = (Excel.Worksheet)Globals.ThisAddIn
.Application.ActiveWorkbook.Sheets[comboBox1.Text];
arbeitsblatt.Activate();
```

Über die Auflistung *Sheets* der aktiven Arbeitsmappe wird durch Angabe des in der ComboBox ausgewählten Eintrags ein Verweis auf das zu aktivierende Arbeitsblatt geholt. Anschließend wird das Arbeitsblatt mit der Methode *Activate* aktiviert.

3.4.2.3 Einen benutzerdefinierten Aufgabenbereich hinzufügen

In diesem Teil des Beispiels fügen Sie dem Projekt einen benutzerdefinierten Aufgabenbereich hinzu, in dem die in einem Arbeitsblatt enthaltenen Kommentare angezeigt und grundlegend verwaltet werden können.

Führen Sie die folgenden Schritte aus, um das Add-In um einen benutzerdefinierten Aufgabenbereich zu erweitern:

1. Rufen Sie aus dem Menü PROJEKT den Befehl NEUES ELEMENT HINZUFÜGEN auf.
2. Wählen Sie die Vorlage *Benutzersteuerelement* aus und bestätigen Sie mit HINZUFÜGEN (Bild 3.57), worauf das Benutzersteuerelement im Entwurfsmodus geöffnet wird (Bild 3.58).

HINWEIS: Analog zum *Aktionsbereich* für Anpassungen auf Dokumentebene gibt es **kein** Element vom Typ *Benutzerdefinierter Aufgabenbereich* für Add-Ins auf Anwendungsebene. Stattdessen muss für einen benutzerdefinierten Aufgabenbereich ein *Benutzersteuerelement* erstellt werden, das später der Auflistung *CustomTaskPanes* hinzugefügt wird.

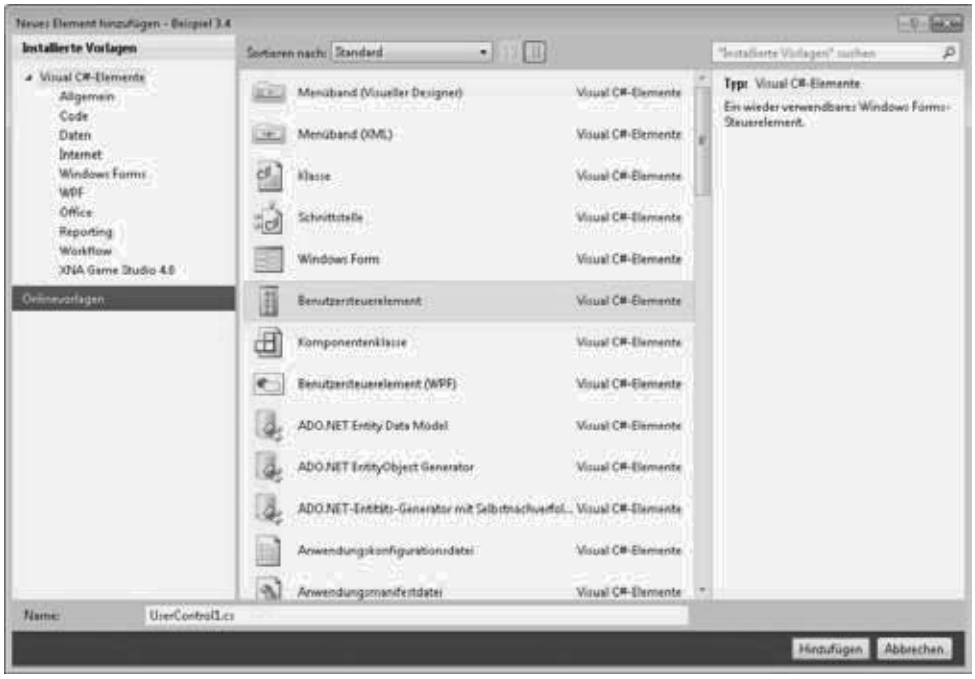

BILD 3.75 Hinzufügen eines Benutzersteuerelements

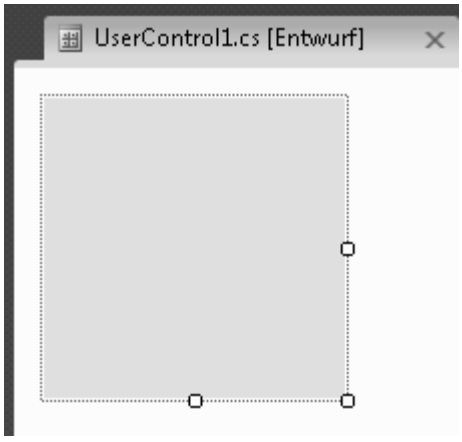

BILD 3.76 Ein Benutzersteuerelement im Entwurfsmodus

3. Passen Sie zunächst die Größe des Benutzersteuerelements an, indem Sie im Eigenschaften-Fenster unter *Size* für die Eigenschaft *Width* einen Wert von *300* und für die Eigenschaft *Height* einen Wert von *400* festlegen.
4. Fügen Sie dem Benutzersteuerelement insgesamt vier *Button*- und ein *ListBox*-Steuerelement aus der Toolbox hinzu.

5. Beschriften Sie die Schaltflächen sinngemäß entsprechend ihrer späteren Funktionalität zum Auslesen, Anzeigen, Ausblenden und Löschen von Kommentaren. Das fertige Benutzersteuerelement sollte ungefähr wie in Bild 3.59 aussehen.

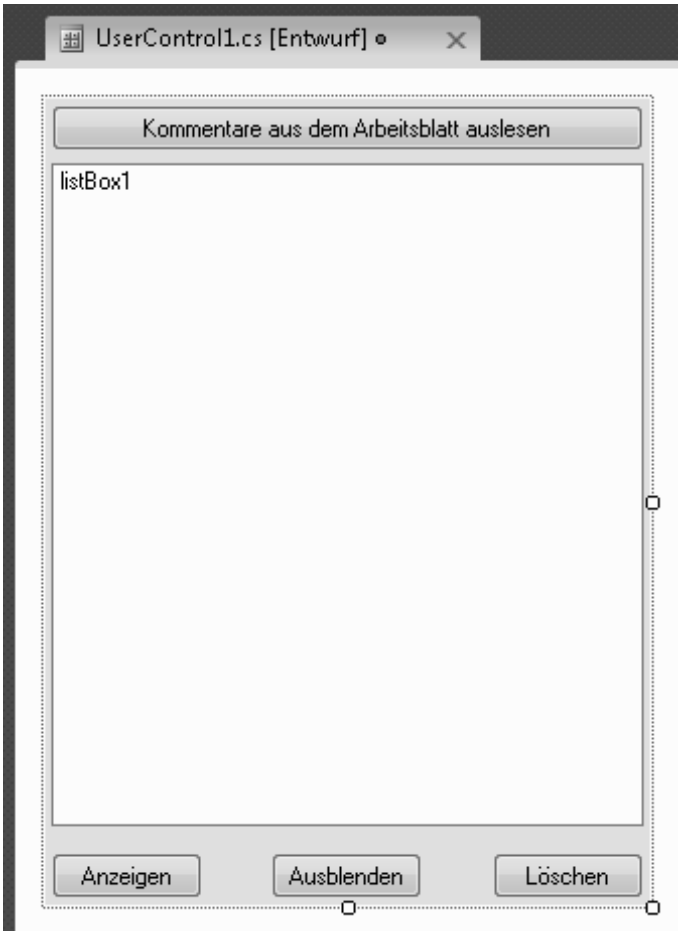

BILD 3.77 Entwurf des benutzerdefinierten Aufgabenbereichs

6. Erstellen Sie für alle vier Schaltflächen die Ereignis-Behandlungsmethoden für das *Click*-Ereignis, indem Sie auf die Schaltflächen im Entwurfsmodus nacheinander einen Doppelklick ausführen.
7. Fügen Sie der Datei *UserControl1.cs* eine using-Anweisung für das Objektmodell von Excel hinzu.

```
using Excel = Microsoft.Office.Interop.Excel;
```

8. Fügen Sie der Klasse *UserControl1* die Methode *KommentareAnzeigen* aus Listing 3.10 hinzu.

LISTING 3.10 Methode zum Auslesen von Kommentaren

```
private void KommentareAnzeigen()
{
    Excel.Worksheet arbeitsblatt =
(Excel.Worksheet)Globals.ThisAddIn.Application.ActiveWorkbook.ActiveSheet;

    listBox1.Items.Clear();
    foreach (Excel.Comment kommentar in arbeitsblatt.Comments)
    {
        listBox1.Items.Add(kommentar.Text());
    }
}
```

In der Methode *KommentareAnzeigen* wird auf die bereits bekannte Weise zuerst ein Verweis auf das aktive Arbeitsblatt geholt. Damit die Einträge der ListBox beim wiederholten Aufruf der Methode nicht mehrmals hinzugefügt werden, werden die vorhandenen Einträge in der ListBox anschließend mit `listBox1.Items.Clear()` entfernt. Nun wird in einer foreach-Schleife die Auflistung *Comments* des Arbeitsblatts durchlaufen, dessen Elemente vom Typ *Comment* sind. Mit der Methode *Text* wird der Inhalt des Kommentars schließlich ausgelesen und der Methode *Add* der Auflistung *Items* der ListBox übergeben, wodurch die Kommentare in der ListBox angezeigt werden.

9. Fügen Sie der Methode *buttonAuslesen_Click* einen Aufruf der Methode *KommentareAnzeigen* hinzu.

   ```
   KommentareAnzeigen();
   ```

10. Fügen Sie der Methode *buttonAnzeigen_Click* Code hinzu, um einen in der ListBox ausgewählten Kommentar im Arbeitsblatt anzuzeigen. Den Quellcode der Methode finden Sie in Listing 3.11.
11. Fügen Sie der Methode *buttonAusblenden_Click* Code hinzu, um einen in der ListBox ausgewählten Kommentar im Arbeitsblatt auszublenden. Den Quellcode der Methode finden Sie in Listing 3.11.
12. Fügen Sie der Methode *buttonLoeschen_Click* Code hinzu, um einen in der ListBox ausgewählten Kommentar aus dem Arbeitsblatt zu löschen. Den Quellcode der Methode finden Sie in Listing 3.11.

LISTING 3.11 Die Klasse UserControl1

```
public partial class UserControl1 : UserControl
{
    public UserControl1()
    {
        InitializeComponent();
    }

    private void KommentareAnzeigen()
    {
        Excel.Worksheet arbeitsblatt =
(Excel.Worksheet)Globals.ThisAddIn.Application.ActiveWorkbook.ActiveSheet;

        listBox1.Items.Clear();
        foreach (Excel.Comment kommentar in arbeitsblatt.Comments)
```

```
        {
            listBox1.Items.Add(kommentar.Text());
        }
    }

    private void buttonAuslesen_Click(object sender, EventArgs e)
    {
        KommentareAnzeigen();
    }

    private void buttonAnzeigen_Click(object sender, EventArgs e)
    {
        Excel.Worksheet arbeitsblatt =
(Excel.Worksheet)Globals.ThisAddIn.Application.ActiveWorkbook.ActiveSheet;
        arbeitsblatt.Comments[listBox1.SelectedIndex + 1].Visible = true;
    }

    private void buttonAusblenden_Click(object sender, EventArgs e)
    {
        Excel.Worksheet arbeitsblatt =
(Excel.Worksheet)Globals.ThisAddIn.Application.ActiveWorkbook.ActiveSheet;
        arbeitsblatt.Comments[listBox1.SelectedIndex + 1].Visible = false;
    }

    private void buttonLoeschen_Click(object sender, EventArgs e)
    {
        Excel.Worksheet arbeitsblatt =
(Excel.Worksheet)Globals.ThisAddIn.Application.ActiveWorkbook.ActiveSheet;
        arbeitsblatt.Comments[listBox1.SelectedIndex + 1].Delete();

        KommentareAnzeigen();
    }
}
```

In den Methoden zum Anzeigen, Ausblenden und Löschen von Kommentaren wird jeweils zuerst ein Verweis auf das aktive Arbeitsblatt geholt. Über die Auflistung *Comments* des Arbeitsblatts erfolgt dann der Zugriff auf die Kommentare. Zur gezielten Auswahl eines Kommentars wird dazu die Eigenschaft *SelectedIndex* der ListBox herangezogen, die den Index des aktuell ausgewählten Elements wiedergibt. Da die Zählung der Kommentare – wie bei vielen Auflistungen in den Objektmodellen von Office – mit 1 beginnt, in der .NET-Programmierung aber der Index 0 für das erste Element verwendet wird, muss zu dem Wert noch 1 hinzuaddiert werden. Bis zu diesem Punkt ist der Code in allen drei Methoden gleich. Zum Anzeigen oder Ausblenden eines Kommentars wird jetzt die Eigenschaft *Visible* in den entsprechenden Methoden auf *true* beziehungsweise *false* festgelegt. Zum Löschen eines Kommentars wird hingegen die Methode *Delete* aufgerufen. Nach dem Löschen wird die ListBox durch den Aufruf von *KommentareAnzeigen* neu initialisiert.

Wenn Sie das Projekt nun starten, werden Sie feststellen, dass der soeben erstellte benutzerdefinierte Aufgabenbereich noch nicht angezeigt wird. Hierzu sind weitere Schritte erforderlich. So muss in der Klasse *ThisAddIn* erst ein Objekt des Benutzersteuerelements erstellt und dieses der Auflistung *CustomTaskPanes* hinzugefügt werden. Führen Sie dazu die folgenden Schritte aus:

13. Fügen Sie der Klasse *ThisAddIn* zwei Klassenmember für das Benutzersteuerelement sowie einen benutzerdefinierten Aufgabenbereich hinzu.

```
private UserControl1 benutzersteuerelement;
private Microsoft.Office.Tools.CustomTaskPane aufgabenbereich;
```

14. Fügen Sie der Methode *ThisAddIn_Startup* Code hinzu, um den benutzerdefinierten Aufgabenbereich zu initialisieren und anzuzeigen.

```
benutzersteuerelement = new UserControl1();
aufgabenbereich = this.CustomTaskPanes.Add(benutzersteuerelement,
"Kommentare verwalten");
aufgabenbereich.Width = 310;
aufgabenbereich.Visible = true;
```

In der Methode *ThisAddIn_Startup* wird zunächst eine Instanz des Benutzersteuerelements erstellt und diese anschließend der Auflistung *CustomTaskPanes* mit der Methode *Add* hinzugefügt. Dazu werden der Add-Methode das Benutzersteuerelement sowie eine Bezeichnung übergeben. Auch wenn die Größe des Benutzersteuerelements bereits im Entwurf festgelegt wurde, wird die eingestellte Breite nicht automatisch für den benutzerdefinierten Aufgabenbereich übernommen. Daher muss die Eigenschaft *Width* noch angepasst werden, worauf die Eigenschaft *Visible* des benutzerdefinierten Aufgabenbereichs schließlich auf *true* festgelegt werden muss, damit der benutzerdefinierte Aufgabenbereich auch angezeigt wird.

3.4.2.4 Menüband und Aufgabenbereich synchron halten

Derzeit wird der benutzerdefinierte Aufgabenbereich mit dem Starten des Add-Ins automatisch geladen. Grundsätzlich kann der Anwender einen benutzerdefinierten Aufgabenbereich über dessen Fensterleiste jederzeit schließen. Mit den folgenden Schritten wird dem Add-In die notwendige Funktionalität hinzugefügt, um den Aufgabenbereich aus dem Menüband heraus jederzeit ausblenden und auch wieder einblenden zu können:

1. Fügen Sie dem Menüband ein neues *Group*-Steuerelement sowie ein *CheckBox*-Steuerelement hinzu, sodass das Menüband wie in folgender Abbildung aussieht.

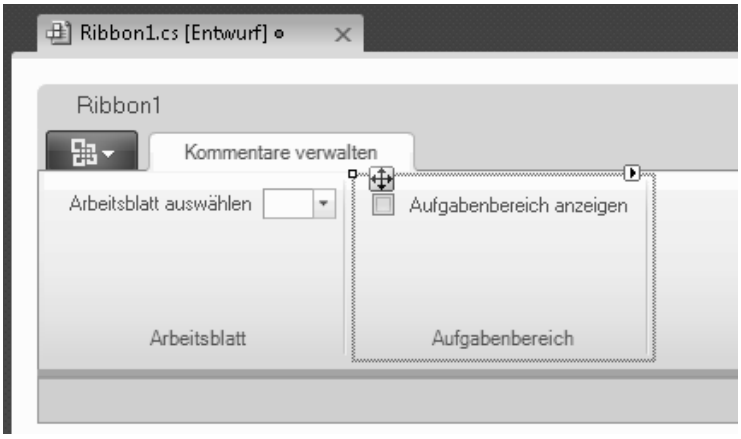

BILD 3.78 Entwurf des erweiterten Menübands

2. Legen Sie für die Eigenschaft *Checked* des Kontrollkästchens den Wert *True* fest.

3. Erweitern Sie die Klasse *ThisAddIn* um die Eigenschaft *Aufgabenbereich*, die den benutzerdefinierten Aufgabenbereich zurückgibt.

```
public Microsoft.Office.Tools.CustomTaskPane Aufgabenbereich
{
    get
    {
        return aufgabenbereich;
    }
}
```

4. Erstellen Sie mit einem Doppelklick auf das Kontrollkästchen im Menüband die Ereignis-Behandlungsmethode für das *Click*-Ereignis.
5. Fügen Sie der Methode *checkBox1_Click* eine Anweisung hinzu, um die Sichtbarkeit des Aufgabenbereichs von der Eigenschaft *Checked* des Kontrollkästchens abhängig zu machen.

```
Globals.ThisAddIn.Aufgabenbereich.Visible = checkBox1.Checked;
```

Zusätzlich soll das Menüband auf ein Schließen des benutzerdefinierten Aufgabenbereichs reagieren und den Status des Kontrollkästchens im Menüband anpassen, damit Menüband und Aufgabenbereich synchron sind:

1. Nehmen Sie in der Klasse *ThisAddIn* in die Methode *ThisAddIn_Startup* eine weitere Anweisung auf, um einen EventHandler für das *VisibleChanged*-Ereignis des Aufgabenbereichs zu erstellen.

```
aufgabenbereich.VisibleChanged += new
EventHandler(aufgabenbereich_VisibleChanged);
```

2. Fügen Sie der vom Editor erzeugten Ereignis-Behandlungsmethode *aufgabenbereich_VisibleChanged* den noch fehlenden Code hinzu, um den Zustand des Kontrollkästchens beim Schließen des Aufgabenbereichs anzupassen. Den vollständigen Quellcode der Klasse *ThisAddIn* finden Sie in Listing 3.12.

```
Globals.Ribbons.Ribbon1.checkBox1.Checked = aufgabenbereich.Visible;
```

LISTING 3.12 Die Klasse ThisAddIn

```
public partial class ThisAddIn
{
    private UserControl1 benutzersteuerelement;
    private Microsoft.Office.Tools.CustomTaskPane aufgabenbereich;

    private void ThisAddIn_Startup(object sender, System.EventArgs e)
    {
        benutzersteuerelement = new UserControl1();
        aufgabenbereich = this.CustomTaskPanes.Add(benutzersteuerelement,
"Kommentare verwalten");
        aufgabenbereich.Width = 310;
        aufgabenbereich.Visible = true;
        aufgabenbereich.VisibleChanged += new EventHandler(aufgabenbereich_
VisibleChanged);
    }

    void aufgabenbereich_VisibleChanged(object sender, EventArgs e)
    {
        Globals.Ribbons.Ribbon1.checkBox1.Checked = aufgabenbereich.Visible;
    }

    public Microsoft.Office.Tools.CustomTaskPane Aufgabenbereich
```

```
    {
        get
        {
            return aufgabenbereich;
        }
    }

    private void ThisAddIn_Shutdown(object sender, System.EventArgs e)
    {
    }

    //VSTO generated code...
}
```

3.4.2.5 Ein Dialogfeld-Startprogramm hinzufügen

Bei einem Dialogfeld-Startprogramm handelt es sich um ein kleines Symbol, das in der unteren rechten Ecke innerhalb einer Gruppe in einer Registerkarte dargestellt wird. Jede Gruppe kann über ein eigenes Dialogfeld-Startprogramm verfügen. Üblicherweise werden Dialogfeld-Startprogramme eingesetzt, um Dialogfelder oder benutzerdefinierte Aufgabenbereiche anzuzeigen. Gehen Sie dazu wie folgt vor:

1. Wählen Sie die gewünschte Gruppe in der Registerkarte aus und klicken Sie auf den kleinen Pfeil in der oberen rechten Ecke, um die *GroupView-Aufgaben* anzuzeigen (Bild 3.61).

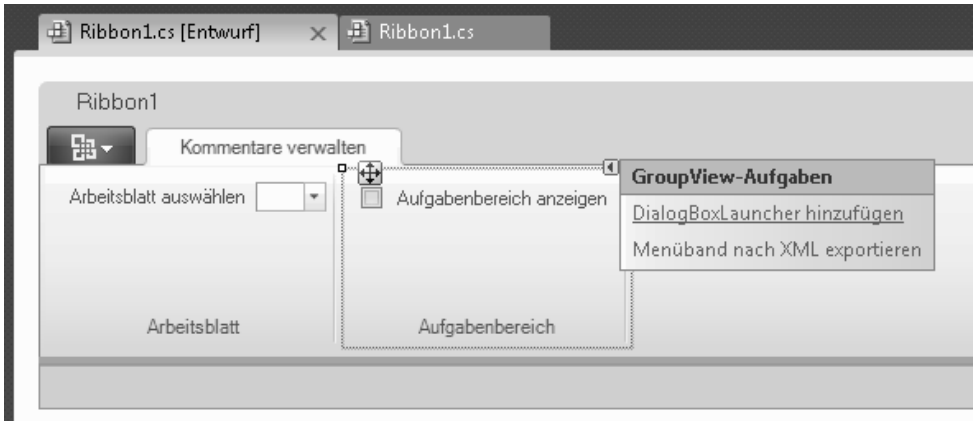

BILD 3.79 DialogBoxLauncher aus den GroupView-Aufgaben hinzufügen

2. Klicken Sie in den *GroupView-Aufgaben* auf den Befehl DialogBoxLauncher hinzufügen, worauf das Symbol in der unteren rechten Ecke der Gruppe angezeigt wird wie in Bild 3.62 zu sehen.

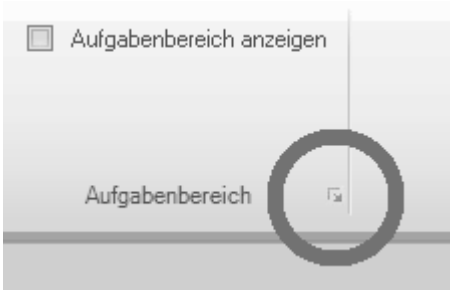

BILD 3.80 Symbol für das Dialogfeld-Startprogramm

3. Erstellen Sie im Eigenschaften-Fenster in der Ansicht *Ereignisse* mit einem Doppelklick auf den Eintrag *DialogLauncherClick* die entsprechende Ereignis-Behandlungsmethode für die Gruppe (Bild 3.63).

BILD 3.81 Ereignis für das Dialogfeld-Startprogramm hinzufügen

4. Fügen Sie in diesem Beispiel der Methode *group2_DialogLauncherClick* eine Anweisung hinzu, um den Aufgabenbereich einzublenden.

```
Globals.ThisAddIn.Aufgabenbereich.Visible = true;
```

3.4.2.6 Das Office-Menü anpassen

Eine weitere Möglichkeit im Rahmen von Menübändern ist die Anpassung des Office-Menüs. Das Office-Menü ist eine geeignete Stelle, um unter anderem eigene Befehle zum Importieren von Daten aufzurufen. So liegen beispielsweise Daten, die von Maschinen in der Produktion erzeugt werden, oftmals in einem nicht dem Standard entsprechenden CSV-Format vor, wodurch individuelle Import-Routinen implementiert werden müssen.

Zur Anpassung des Office-Menüs können aus der Toolbox in der Kategorie *Steuerelemente für Office-Menübänder* die folgenden Steuerelemente verwendet werden:

- Button
- CheckBox
- Gallery
- Menu
- Separator
- SplitButton
- ToggleButton

Gehen Sie zum Erweitern des Office-Menüs wie folgt vor:

1. Öffnen Sie den Menü-Designer, indem Sie im Menüband-Designer auf das Symbol für das Office-Menü klicken, das sich im oberen linken Bereich des Menübands befindet (Bild 3.64).

BILD 3.82 Der Menü-Designer zum Erweitern des Office-Menüs

2. Ziehen Sie nun die gewünschten Steuerelemente aus der Toolbox in den Menü-Designer.

Die Konfiguration und Behandlung von Ereignissen für die einzelnen Steuerelemente erfolgt auf die gleiche Weise wie für Steuerelemente, mit denen die Registerkarten des Menübands angepasst werden. Zur Laufzeit können Erweiterungen des Office-Menüs in der Backstage-Ansicht unter dem Punkt *Add-Ins* aufgerufen werden (Bild 3.65).

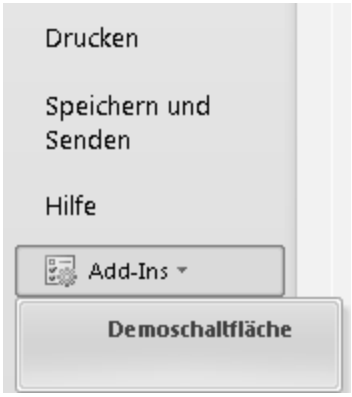

BILD 3.83 Beispiel einer Erweiterung des Office-Menüs zur Laufzeit

3.4.2.7 Add-Ins aus Office verwalten

Wenn Add-Ins zur Entwicklungszeit aus Visual Studio heraus gestartet werden, wird die zugehörige Office-Anwendung geöffnet und das Add-In in der Anwendung geladen. Beim Beenden der Anwendung befindet man sich als Entwickler wieder in der Oberfläche von Visual Studio. Wenn man nun davon ausgeht, dass das Add-In nicht mehr in der Office-Anwendung registriert ist, irrt man sich allerdings. Das Add-In steht weiterhin in der Office-Anwendung zur Verfügung. Dies kann man ganz einfach nachvollziehen, indem man die Office-Anwendung nach dem Testen einmal direkt startet. Sofort fällt auf, dass das Add-In, das vielleicht nur ein Demo-Projekt ist, weiterhin von der Office-Anwendung geladen wird.

Somit müssen Add-Ins, sofern sie nicht weiter benötigt werden, manuell aus der Office-Anwendung wieder entfernt werden. Gehen Sie dazu wie folgt vor:

1. Öffnen Sie aus dem *Datei*-Menü die *Optionen* der Office-Anwendung und wechseln Sie anschließend zu dem Punkt *Add-Ins*.
2. Es erscheint eine Übersicht der in der Office-Anwendung registrierten Add-Ins, die von dieser Stelle aus auch verwaltet werden können. Wie in Bild 3.66 zu sehen, ist in der Auflistung auch das aus Visual Studio heraus gestartete Add-In aufgeführt.

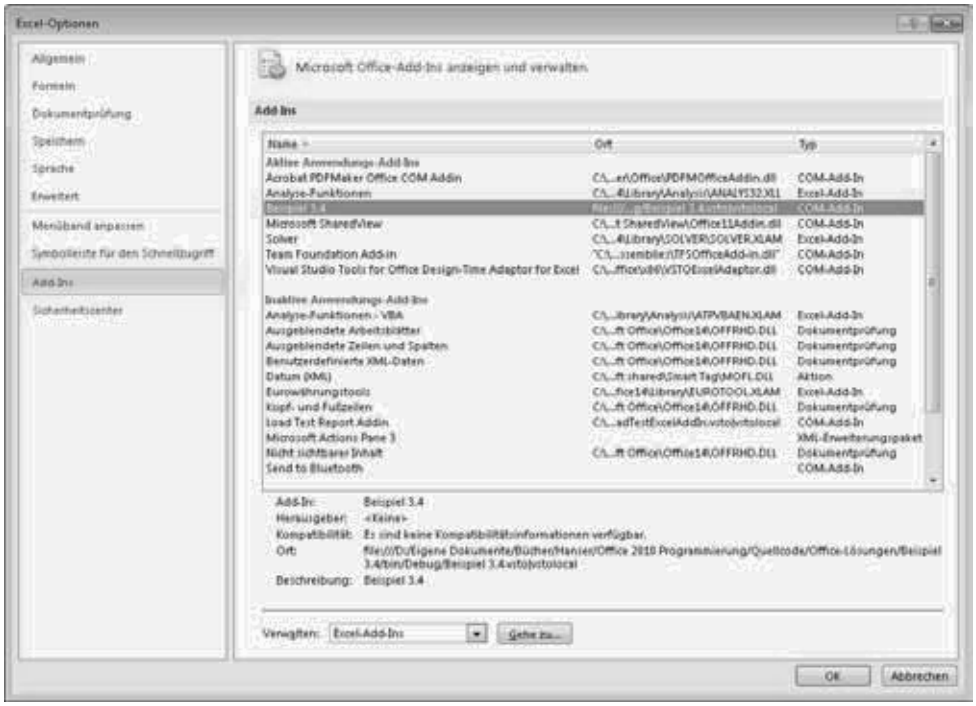

BILD 3.84 Add-Ins in Office anzeigen und verwalten

3. Um ein Add-In aus Office zu entfernen, wählen Sie unter *Verwalten* den Typ des Add-Ins aus. Da es sich im Fall eines mit VSTO entwickelten Add-Ins um ein COM-Add-In handelt, wählen Sie den Eintrag *COM-Add-Ins* aus dem Drop-down-Menü aus und bestätigen Sie die Auswahl mit GEHE ZU (Bild 3.67).

BILD 3.85 COM-Add-Ins verwalten

4. Es erscheint das Dialogfeld *COM-Add-Ins* (Bild 3.68), in dem Sie das gewünschte Add-In markieren und mit ENTFERNEN vollständig aus der Office-Anwendung entfernen können. Alternativ können Sie auch lediglich die Auswahl des Add-Ins deaktivieren, wodurch dieses beim Starten der Anwendung zwar nicht mehr geladen wird, aber weiterhin in Office registriert ist.

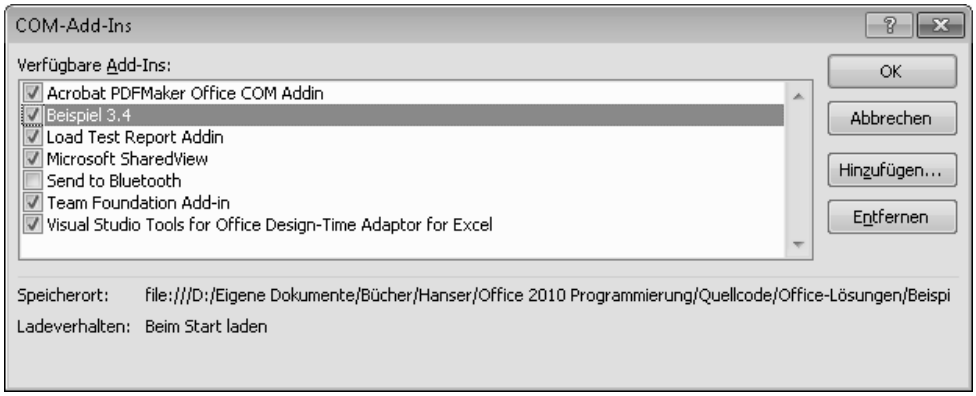

BILD 3.86 Entfernen und Deaktivieren von COM-Add-Ins

3.4.2.8 Die fertige Office-Lösung

Wenn Sie das Projekt nun zum Testen starten, sollte die Office-Lösung wie anfangs beschrieben funktionieren. In diesem Beispiel haben Sie die folgenden Techniken der Office-Programmierung kennengelernt:

- Erstellen von Add-Ins auf Anwendungsebene
- Erstellen von Menübändern
- Zugriff auf Menübänder zur Laufzeit
- Erstellen und Anzeigen von benutzerdefinierten Aufgabenbereichen
- Zugriff auf benutzerdefinierte Aufgabenbereiche zur Laufzeit
- Das Office-Menü (Datei-Menü) anpassen
- Dialogfeld-Startprogramme erstellen
- Auf Ereignisse reagieren
- Add-Ins in Office-Anwendungen verwalten

3.5 Mit Daten in Excel arbeiten

In diesem Beispiel für Excel kommen die folgenden Techniken beziehungsweise Features von VSTO zum Einsatz:

- Erstellen von Add-Ins auf Anwendungsebene
- Erstellen von Menübändern
- Das ListObject für Daten verwenden
- Zugriff auf Arbeitsblätter zur Laufzeit
- Daten an NamedRange binden
- Erstellen von Diagrammen zur Laufzeit