



Dialoge unter VB.NET richtig kapseln

Gewaltenteilung

Unter Visual Basic .NET gestaltet sich der Einsatz von Dialogen wesentlich flexibler als bei früheren VB-Versionen – allerdings auch ganz anders. Besonders VB6-Programmierer tun sich schwer. Die objektorientierte Programmierung erlaubt eine saubere Trennung zwischen Dialog, Daten und dem aufrufenden Programm.

Während in der MFC-Programmierung die strikte Trennung von Daten, Ansichten und Dialogen aufgrund der Document-View-Architektur schon immer gang und gäbe war (siehe auch [1]), setzen VB-Programmierer dieses Konzept nur sehr selten um. Stattdessen wuchern häufig komplizierte „Kreuz und quer“-Referenzen zwischen verschiedenen Fensterklassen, die eine langfristige Wartung der Programme erheblich erschweren und die Wiederverwendung von Dialogklassen in einem anderem Kontext unmöglich machen.

Dabei lassen sich Dialoge in .NET hervorragend auf eine Weise kapseln, dass sie keinerlei Informationen über den Aufrufer benötigen. Auch der Aufrufer muss keine Interna der Dialogklasse kennen. Er muss nicht wissen, welche Steuerelemente im Dialogfenster existieren und wie die Daten übertragen werden müssen.

Im Idealfall veröffentlicht eine Dialogklasse nur eine einzige statische Methode, welche die benötigten Daten übergibt und das Dialogfenster anzeigt. Da der

Datum	Gesprächspartner	Thema
01.01.2004	Bundespräsident	Neujahr
04.02.2004	Stantenisch	Dialoge
11.06.2004	Chakvitz	Schweifing
23.12.2004	Wahlactormann	Geschenke

Abbildung 1 Die Terminliste und der Dialog zum Ändern eines gewählten Eintrags.

Termin

Datum: Donnerstag, 23. Dezember 2004

Gesprächspartner: Wahlactormann

Thema: Geschenke

Wichtig:

OK

Abbrechen

Aufrufer nicht selbst eine Instanz der Dialogklasse anlegen muss, könnten alle Konstruktoren privat oder geschützt sein. In der Praxis wird allerdings zusätzlich ein öffentlicher Standardkonstruktor benötigt, damit das Dialogfenster mit dem Designer der Entwicklungsumgebung bearbeitet werden kann. Das Beispiel eines simplen Dialogs zur Eingabe eines Termins in einer Terminliste zeigt, wie einfach sich eine solche Dialogklasse implementieren lässt (Abbildung 1).

Da in Dialogen meist mehrere Datenelemente bearbeitet werden können, empfiehlt sich der Einsatz einer Klasseninstanz, um die Informationen zwischen Aufrufer und Dialog auszutauschen. Oft steht bereits eine entsprechende Klasse bereit. Müssen nur ganz wenige Parameter übertragen werden, wie zum Beispiel Name und Kennwort, dann können diese auch als Referenzparameter definiert werden. Wird diese Klasse nur im Zusammenhang mit der Dialogklasse verwendet, kann sie auch als Subklasse in der Dialogklasse angelegt werden.

Ein einfacher Dialog mit OK- und Abbrechen-Schaltflächen

Listing 1 zeigt die Klasse *Termin*, die die Daten für das Beispielprojekt aufnehmen soll. Sie besteht aus öffentlichen Mem-

ber-Variablen und Konstruktoren zur vereinfachten Instanzierung. Darüber hinaus wird keine Funktionalität benötigt. Die Dialogklasse ist eine gewöhnliche Fensterklasse, die zusätzlich die statische Methode *CreateAndShow()* bereitstellt.

Diese Methode nimmt als einzigen Parameter die Referenz des *Termin*-Objektes an, instanziiert die Dialogklasse, besetzt die Steuerelemente mit den Daten und zeigt den Dialog an. Der Aufruf von *ShowDialog()* ist erst beendet, wenn der Anwender den Dialog geschlossen hat. Nur beim Betätigen der *OK*-Schaltfläche werden die Daten aus den Steuerelementen zurück in die Instanz der Klasse *Termin* übertragen. Der Rückgabewert von *CreateAndShow()* ist entweder *DialogResult.OK* oder *DialogResult.Cancel*, je nach Anwenderaktion. Die gesamte Funktionalität des Dialogs beschränkt sich in diesem Beispiel auf die Methode *CreateAndShow()*. Für das Schließen des Dialogs über die Schaltflächen oder Tastenkombis wird kein zusätzlicher Code benötigt. Hierfür genügen einige Steuerelementeinstellungen, die in Tabelle 1 näher erläutert werden.

Zum Anzeigen der Termine nutzt das Hauptprogramm ein *ListView*-Control,

Auf einen Blick

Autor

Dr. Joachim Fuchs ist Software-Entwickler und Dozent, seit zwei Jahren mit dem Schwerpunkt .NET. Sie erreichen ihn über www.fuechse-online.de/beruflich/index.html.



dotnetpro.code
A0405Dialoge



Sprachen VB.NET, C#

Technik Windows Forms

Listing 1

Die Hilfsklasse Termin für den Datenaustausch.

```
Public Class Termin

    Public Datum As DateTime
    Public Gesprächspartner As String
    Public Thema As String
    Public Wichtig As Boolean

    Public Sub New(ByVal datum As String, _
        ByVal gesprächspartner As String, _
        ByVal thema As String, _
        ByVal wichtig As Boolean)

        Me.Datum = DateTime.Parse(datum)
        Me.Gesprächspartner = gesprächspartner
        Me.Thema = thema
        Me.Wichtig = wichtig
    End Sub

    Public Sub New()
        Datum = DateTime.Now
    End Sub

End Class
```

dessen *View*-Eigenschaft auf *Details* gesetzt ist. Die zwei Methoden *NeuenTerminHinzufügen()* und *TerminEintragen()* in Listing 3 organisieren die Einträge in der Listendarstellung und werden sowohl zum Einrichten der Liste als auch im Zusammenhang mit den Dialogen genutzt. Die Verbindung zwischen einem *ListView*-Eintrag und dem zugehörigen Datenobjekt erfolgt durch Zuweisen der Referenz an die *Tag*-Eigenschaft des *ListViewItem*-Objektes.

Tabelle 1

Fensterfunktionen über Eigenschaften steuern.

Steuerelement	Eigenschaft	Wert
Form	AcceptButton	BTNOK
	CancelButton	BTNAbbrechen
BTNOK	DialogResult	OK
BTNAbbrechen	DialogResult	Cancel

Nach diesen Vorbereitungen gestaltet sich der Aufruf des Dialogs äußerst simpel. Über zwei Menüpunkte kann er entweder für ein neues *Termin*-Objekt oder für ein ausgewähltes aufgerufen werden. Den für den Aufruf notwendigen Code enthalten die Routinen *MNUDialogNeueinfach_Click()* und *MNUÄndernDialogTerminEinfach_Click()*. Die einzige Aktion besteht jeweils darin, die statische Methode *CreateAndShow()* aufzurufen und die Referenz entweder eines neuen oder des ausgewählten *Termin*-Objektes zu übergeben. An dieser Stelle wird keine Instanz der Dialogklasse erzeugt und auch nicht auf interne Methoden oder Member-Variablen dieser Klasse zugegriffen. Alles dies ist ja bereits in der Methode *CreateAndShow()* gekapselt (siehe Listing 2).

Dialoge mit einer Übernehmen-Schaltfläche

In manchen Situationen ist es sinnvoll, eine *Übernehmen*- oder *Vorschau*-Schaltfläche im Dialogfenster anzubieten. Der Anwender kann dann bereits vorgenommene Änderungen akzeptieren und die Auswirkung in den jeweiligen Ansichten

betrachten. Im Beispielprogramm werden beim Betätigen der *Übernehmen*-Schaltfläche alle Eingabefelder eingeleistet und die Darstellung im *ListView*-Control aktualisiert.

Da die *Übernehmen*-Schaltfläche das Dialogfenster nicht schließen soll, die Methode *CreateAndShow()* aber erst nach dem Schließen beendet wird, müssen einige zusätzliche Maßnahmen getroffen werden. Hierbei übergibt der Aufrufer mit einem Delegate die Adresse einer Rückrufmethode, die als Reaktion auf die Betätigung der *Übernehmen*-Schaltfläche aufgerufen wird (Listing 4). Die Definition des Delegates erfolgt in der Dialogklasse:

```
Public Delegate Sub UebernehmenDelegate _
    (ByVal data As Termin)
```

Der Aufrufer stellt eine Methode mit dieser Signatur zur Verfügung. Im Beispiel ist das die Methode *AenderungUebernehmen()*. Die Methode selbst ist in der Dialogklasse nicht bekannt. Nur ihre Signatur wird mit der Delegate-Definition festgelegt. Daher kann die Rückrufmethode auch als *Private* deklariert werden. Ihre Adresse wird mithilfe des *AddressOf*-Operators er-

Listing 2

CreateAndShow – die Schnittstelle zur Außenwelt.

```
Public Shared Function CreateAndShow(ByVal termin As Termin) _
    As DialogResult

    Dim dialog As New DialogTerminEinfach
    ' Steuerelemente initialisieren
    dialog.DTPDatum.Value = termin.Datum
    dialog.TBThema.Text = termin.Thema
    dialog.TBGesprPartner.Text = termin.Gesprächspartner
    dialog.CHKWichtig.Checked = termin.Wichtig

    ' Dialog modal anzeigen
    Dim dr As DialogResult = dialog.ShowDialog()

    ' OK oder Abbrechen gedrückt?

    If dr = System.Windows.Forms.DialogResult.OK Then
        ' Daten aus Steuerelementen zurückerlesen
        termin.Datum = dialog.DTPDatum.Value
        termin.Thema = dialog.TBThema.Text
        termin.Gesprächspartner = dialog.TBGesprPartner.Text
        termin.Wichtig = dialog.CHKWichtig.Checked
    End If

    ' Ressourcen freigeben
    dialog.Dispose()

    Return dr

End Function
```

Listing 3

Termine im ListView-Control eintragen.

```

Public Function NeuenTerminHinzufügen(ByVal termin As Termin) _
    As ListViewItem
    ' Neues Item einrichten
    Dim lvi As New ListViewItem
    TerminEintragen(lvi, termin)

    ' Item an ListView anhängen und Referenz zurückgeben
    Return LVTermine.Items.Add(lvi)
End Function

Public Sub TerminEintragen(ByVal lvi As ListViewItem, _
    ByVal termin As Termin)
    ' Alle untergeordneten Items löschen
    lvi.SubItems.Clear()

    ' Die Tag-Referenz verweist auf die zugehörigen Daten
    lvi.Tag = termin

    ' Gleiche Darstellung für alle untergeordneten Einträge
    lvi.UseItemStyleForSubItems = True

    ' Datum, Gesprächspartner und Thema eintragen
    lvi.Text = termin.Datum.ToShortDateString()
    lvi.SubItems.Add(termin.Gesprächspartner)
    lvi.SubItems.Add(termin.Thema)

    ' Wichtige Termine farblich kennzeichnen
    If termin.Wichtig Then
        lvi.ForeColor = Color.DarkBlue
    Else
        lvi.ForeColor = Color.Black
    End If
End Sub

```

mittelt und an den Konstruktor von *ÜbernehmenDelegate* übergeben.

In der Dialogklasse werden zusätzliche Member-Variablen benötigt, um in Ereignissen auf die Daten zugreifen zu können und die Rückrufmethode aufzurufen:

```

' Referenz der Daten
Protected Data As Termin

' Referenz des Rückruf-Delegates
Protected UebernehmenCallback As _
    UebernehmenDelegate

```

Die Methode *CreateAndShow()* nimmt in dieser Variante als zusätzlichen Para-

meter die Referenz des Delegate-Objekts an. Ist diese ungleich *Nothing*, dann wird die *Übernehmen*-Schaltfläche freigegeben und die Referenzen werden in den beiden Member-Variablen gespeichert.

Wird keine Rückrufmethode übergeben, dann wird die Schaltfläche gesperrt und der Dialog verhält sich so, wie im vorangegangenen Beispiel.

Wenn der Anwender die *Übernehmen*-Schaltfläche aktiviert, dann wird der Ereignis-Handler *BTNUebernehmen_Click* aufgerufen. Über die Member-Variable *Data* greift er auf das Datenobjekt zu und trägt die geänderten Informationen aus

den Eingabefeldern ein. Anschließend ruft er die Rückrufmethode auf, die der Aufrufer übergeben hat.

Zum Einrichten neuer Termin-Objekte wird die *MainWindow*-Klasse um die Member-Variable *NeuerEintrag* und die Methode *NeuenTerminHinzufuegenOderAendern* ergänzt. Diese Ergänzungen sowie die Implementierungen der anderen Ereignis-Handler des Menüs finden Sie im Beispielprojekt auf der Heft-CD.

Auch beim Einbau einer Rückrufmethode für eine *Übernehmen*-Schaltfläche wird die strikte Trennung von Aufrufer und Dialogklasse beibehalten. Die Dialogklas-

Listing 4

Die erweiterte CreateAndShow-Methode nimmt auch eine Rückrufmethode entgegen.

```

Public Shared Function CreateAndShow(ByVal termin As Termin, _
    ByVal uebernehmenCallback As UebernehmenDelegate) As DialogResult

    ' Dialog anzeigen
    Dim dr As DialogResult = dialog.ShowDialog()

    ' Dialog instanzieren und initialisieren
    Dim dialog As New DialogTerminRückruf
    dialog.DTPDatum.Value = termin.Datum
    dialog.TBThema.Text = termin.Thema
    dialog.TBGesprPartner.Text = termin.Gesprächspartner
    dialog.CHKWichtig.Checked = termin.Wichtig

    ' Bei OK Daten übertragen
    If dr = System.Windows.Forms.DialogResult.OK Then
        termin.Datum = dialog.DTPDatum.Value
        termin.Thema = dialog.TBThema.Text
        termin.Gesprächspartner = dialog.TBGesprPartner.Text
        termin.Wichtig = dialog.CHKWichtig.Checked
    End If

    ' Wenn eine Rückrufmethode angegeben wurde, Schaltfläche freigeben
    ' und Referenzvariablen setzen
    If Not uebernehmenCallback Is Nothing Then
        dialog.BTNUebernehmen.Enabled = True
        dialog.Data = termin
        dialog.UebernehmenCallback = uebernehmenCallback
    End If

    dialog.Dispose()

    Return dr
End Function

```

Listing 5

Die Grundfunktionen der Dialoge lassen sich in einer Basisklasse implementieren

```
Public Class Basisdialog
    Inherits System.Windows.Forms.Form
    ...
    ' Signatur der Rückrufmethode mit allgemeinem Parameter
    Public Delegate Sub RueckrufDelegate(ByVal data As Object)

    ' Referenzen der Daten und des RueckrufDelegates
    Protected Data As Object
    Protected Rueckrufmethode As RueckrufDelegate

    ' Transfermethoden, die überschrieben werden müssen
    Protected Overridable Sub DatenInSteuerelementeUebertragen()
        Throw New Exception _
            ("DatenInSteuerelementeUebertragen muss überschrieben werden")
    End Sub

    Protected Overridable Sub DatenAusSteuerelementenLesen()
        Throw New Exception _
            ("DatenAusSteuerelementenLesen muss überschrieben werden")
    End Sub

    Protected Shared Function CreateAndShow( _
        ByVal dialog As Basisdialog, ByVal data As Object, _
        ByVal rueckrufmethode As RueckrufDelegate) As DialogResult

        ' Informationen für Rückruf speichern
        dialog.Data = data
        dialog.Rueckrufmethode = rueckrufmethode

        ' Übernehmen-Schaltfläche einrichten
        dialog.BTNUebernehmen.Enabled = Not rueckrufmethode Is Nothing

        ' Daten in Steuerelemente kopieren
        dialog.DatenInSteuerelementeUebertragen()

        ' Dialog modal anzeigen
        Dim dr As DialogResult = dialog.ShowDialog()

        ' Bei OK, Daten zurückübertragen
        If dr = System.Windows.Forms.DialogResult.OK Then
            dialog.DatenAusSteuerelementenLesen()
        End If

        dialog.Dispose()
        Return dr
    End Function

    Private Sub BTNUebernehmen_Click(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles BTNUebernehmen.Click
        ' Daten aus Steuerelementen transferieren
        DatenAusSteuerelementenLesen()

        ' Rückrufmethode aufrufen, sofern vorhanden
        If Not Rueckrufmethode Is Nothing Then Rueckrufmethode(Data)
    End Sub
End Class
```

se hat keinerlei Kenntnis davon, wie die Methode implementiert wurde und was sie bewirkt. Sie kennt nur die Signatur – also die Parameterliste und den Rückgabebetyp – und speichert die Referenz des Delegate-Objektes für den späteren Aufruf.

Der Aufrufer seinerseits benötigt keine Detailkenntnisse über Steuerelementereignisse innerhalb des Dialogs. Er gibt nur an, welche Methode im Falle eines Rückrufs aufzurufen ist. Die Schnittstelle zwischen Aufrufer und Dialog ist nach wie vor die statische Methode *CreateAndShow()* und das Datenobjekt.

Verallgemeinerung durch eine Dialog-Basisklasse

Um ein einheitliches Aussehen aller Dialoge einer Anwendung zu gewährleisten, bietet es sich an, eine Basisklasse mit gemeinsamen Einstellungen und Komponenten wie zum Beispiel den Schaltflächen *OK*, *Abbrechen* und *Übernehmen* einzurichten. Von dieser können dann alle benötigten Dialogklassen abgeleitet werden. Ein Beispiel für eine solche Ba-

sisklasse sehen Sie in Listing 5. Die Klasse *Basisdialog* definiert ein Dialogfenster, das die drei beschriebenen Schaltflächen und die grundlegenden Aktionen enthält. Die beiden Member-Variablen *Data* und *Rueckrufmethode* dienen wie im vorherigen Beispiel zum Speichern der Informationen für die *Übernehmen*-Funktion.

Zwei Methoden müssen von jeder abgeleiteten Klasse überschrieben werden, da sie dialogspezifisch sind: *DatenInSteuerelementeUebertragen()* und *DatenAusSteuerelementenLesen()*. Sie organisieren den Datentransfer zwischen dem Datenobjekt und den Steuerelementen des Dialogfensters. Aus OOP-Gesichtspunkten müssten beide abstrakt (*MustInherit*) definiert werden, aber aus praktischen Erwägungen – der Designer könnte dann nicht mit der Fensterklasse arbeiten – wird darauf verzichtet.

Die statische Methode *CreateAndShow()* wird hier geschützt definiert, da sie für den Anwender der abgeleiteten Dialogklassen ohne Bedeutung ist. Der Datentransfer von und zu den Steuerelementen wird mit den beiden oben be-

schriebenen Methoden realisiert. In der Methode *CreateAndShow()* der Basisklasse kann der Dialog nicht instanziiert werden, da der Typ ja nicht bekannt ist. Stattdessen wird die Referenz eines bereits erzeugten Dialog-Objektes als zusätzlicher Parameter (*dialog*) übergeben.

Auch um die *Übernehmen*-Schaltfläche braucht sich eine abgeleitete Dialogklasse nicht selbst zu kümmern. Im *Click*-Eventhandler werden die Transfermethode *DatenAusSteuerelementenLesen()* und anschließend die angegebene Rückrufmethode aufgerufen. Ihre Verfügbarkeit wird über den Parameter *rueckrufmethode* gesteuert. Im Beispiel erfolgt das über die *Enabled*-Eigenschaft. Alternativ können Sie die Taste aber auch ganz ausblenden, wenn sie nicht benötigt wird.

Für einen einfachen Dialog mit einer *Übernehmen*-Schaltfläche sind nur noch wenige zusätzliche Schritte erforderlich (Listing 6). Mit diesen wenigen Codezeilen wird der gleiche Dialog realisiert, den Sie bereits kennen gelernt haben. Lediglich die Signatur der Rückrufmethode unterscheidet sich, da hier eine allgemei-

Listing 6

Mithilfe einer Dialog-Basisklasse lässt sich der gewünschte Dialog sehr einfach implementieren.

```
Public Class DialogTerminAbgeleitet
    Inherits DialogUsage.Basisdialog

    ...

Public Overloads Shared Function CreateAndShow( _
    ByVal data As Termin, ByVal rueckrufmethode As RueckrufDelegate) _
    As DialogResult
    ' Abarbeitung an statische Methode der Basisklasse delegieren
    Return CreateAndShow(New DialogTerminAbgeleitet, data, rueckrufmethode)
End Function

Protected Overrides Sub DatenInSteuerelementeUebertragen()

    Dim data As Termin = DirectCast(Me.Data, Termin)

    ' Daten in Steuerelemente eintragen
    Me.DTPDatum.Value = data.Datum

    Me.TBThema.Text = data.Thema
    Me.TBGesprPartner.Text = data.Gesprächspartner
    Me.CHKWichtig.Checked = data.Wichtig

End Sub

Protected Overrides Sub DatenAusSteuerelementenLesen()

    Dim data As Termin = DirectCast(Me.Data, Termin)

    ' Daten aus Eingabefeldern übernehmen
    data.Datum = Me.DTPDatum.Value
    data.Thema = Me.TBThema.Text
    data.Gesprächspartner = Me.TBGesprPartner.Text
    data.Wichtig = Me.CHKWichtig.Checked

End Sub
End Class
```

ne *Object*-Referenz benutzt werden muss und gegebenenfalls ein *Typecast* für den Zugriff erforderlich ist.

Einen Wermutstropfen hat die Fenstervererbung dennoch: Der Visual-Studio-Designer hat noch allerlei Bugs. Beispielsweise verschieben sich die Schaltflächen in abgeleiteten Fenstern nach und nach abwärts, wenn man deren *Anchor*-Eigenschaft in der Basisklasse auf *Bottom+Right* einstellt. Dennoch sollte man sich nicht durch die Unzulänglichkeiten des Designers davon abhalten lassen, die Vorteile der Vererbung auch für Dialoge zu nutzen. Das

beschriebene Konzept lässt sich natürlich auch mit C# umsetzen [2].

Fazit

Der konsequente Einsatz der objektorientierten Programmierung führt zu klaren Strukturen beim Aufbau von Dialogen und fördert die Wiederverwendbarkeit. Mit wenigen Handgriffen lässt sich eine Basisklasse als Grundbaustein für alle Dialoge einer Anwendung einrichten, welche die Implementierung der benötigten Dialoge stark vereinfachen. We-

sentlich ist die Trennung zwischen Aufruf eines Dialogs auf der einen Seite und der gekapselten Implementierung auf der anderen Seite. Zugriffe des Aufrufers auf Elemente des Dialogs und Zugriffe der Dialogklasse auf Interna der aufrufenden Klasse müssen vermieden werden. |||||

[1] Torsten Zimmermann, Langlebige Anwendungsarchitekturen leicht gemacht, dotnetpro 7/8 2003, Seite 80 ff.

[2] C#-Beispiel unter www.fuechse-online.de/beruflich/Beispiele/DialogeCSharpTeil1.htm