

HANSER



Leseprobe

Walter Doberenz, Thomas Gewinnus

Der VB-Programmierer

VB lernen - Professionell anwenden - Lösungen nutzen

ISBN: 978-3-446-42022-9

Weitere Informationen oder Bestellungen unter

<http://www.hanser.de/978-3-446-42022-9>

sowie im Buchhandel.

25

Wie kann ich ... (Oberfläche/Komponenten)

Dieses Kapitel behandelt die folgenden Themen:

- Windows Formulare und Komponenten
- Grafikausgabe
- Drag & Drop

25.1 ... den Inhalt des UI sichern?

Häufig kommt es vor, dass man nach Programmstart das User Interface wieder in dem Zustand vorfinden möchte, in dem man es verlassen hat. Eine einfache Möglichkeit besteht darin, den Inhalt in der Konfigurationsdatei *user.config* zu sichern. Alternativ kann man die Werte aber auch in eine Datei streamen.

Benutzerschnittstelle

Die Abbildung 25.1 zeigt eine primitive Eingabemaske, deren Einstellungen wir sichern wollen.

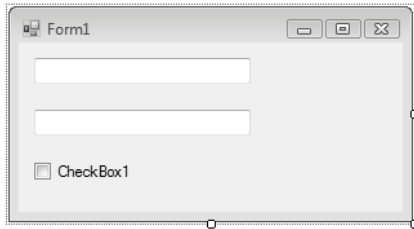


Abbildung 25.1
Entwurfsansicht

Variante 1: Abspeichern in einer Konfigurationsdatei

Öffnen Sie das Eigenschaftenfenster von `TextBox1` und klicken Sie unter `ApplicationSettings` auf `PropertyBinding`. Im Dialogfenster für die Anwendungseinstellungen klappen Sie die `Text`-Eigenschaft auf und klicken ganz unten auf `(Neu...)`. Weisen Sie im nachfolgenden Dialog eine neue Anwendungseinstellung zu, z.B. unter dem Namen `Wert1`. Analog verfahren Sie für `TextBox2`, nehmen aber für die entsprechende Anwendungseinstellung einen anderen Namen (`Wert2`). Die gleiche Vorgehensweise gilt auch für `CheckBox1`, nur dass Sie hier nicht die `Text`-, sondern die `Checked`-Eigenschaft an eine neue Anwendungseinstellung (hier `Wert3`) binden.

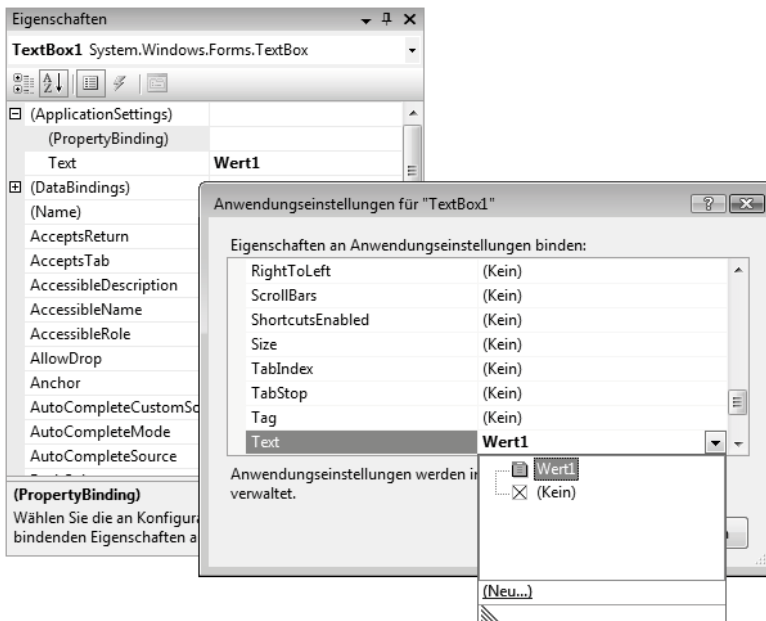


Abbildung 25.2 Binden der Eigenschaften

Nun müssen Sie nur noch dafür sorgen, dass beim Schließen von `Form1` die Anwendungseinstellungen gesichert werden. Öffnen Sie dazu einen Eventhandler für das `FormClosing`-Ereignis und besetzen Sie diesen wie folgt:

```
Private Sub Form1_FormClosing(ByVal sender As Object, _
    ByVal e As System.Windows.Forms.FormClosingEventArgs) Handles Me.FormClosing
    My.Settings.Save()
End Sub
```

Nun können Sie das Beispiel starten und Einträge in die Benutzerschnittstelle vornehmen. Nach dem Schließen des Formulars und einem Neustart sollten diese Einträge wieder zu sehen sein.

Alternativ kann das automatische Speichern der Settings auch über das Anwendungsframework realisiert werden (*Projekt|Eigenschaften|Anwendung*):

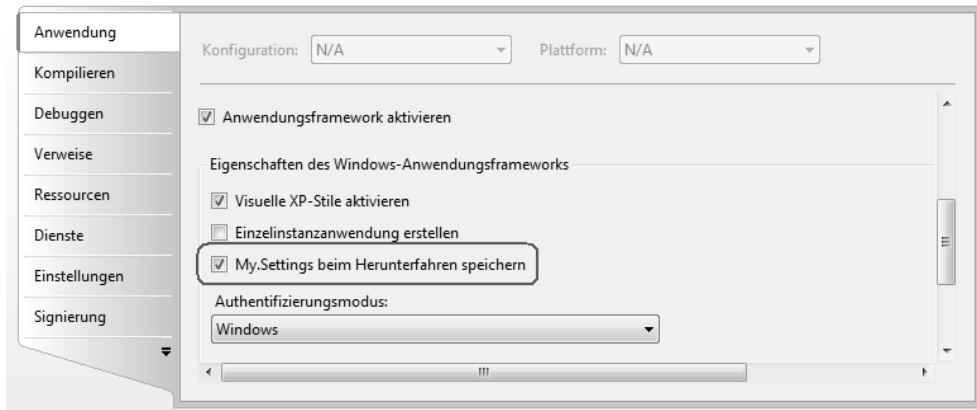


Abbildung 25.3 Automatisches Sichern der Settings

Wenn Sie die Projekteinstellungen öffnen (Menüpunkt *Projekt/Eigenschaften...* oder per Doppelklick auf den Knoten *Settings.settings* im Projektmappen-Explorer) können Sie einen Blick auf die vorhandenen Anwendungseinstellungen werfen (siehe Abbildung 25.4).

	Name	Typ	Bereich	Wert
▶	Wert1	string	Benutzer	
	Wert2	string	Benutzer	
	Wert3	bool	Benutzer	False
*				

Abbildung 25.4
Anwendungseinstellungen

Die aktuell gesicherten Werte finden Sie hier allerdings nicht, sondern in der XML-Datei *user.config* im Verzeichnis *...\Benutzer\IhrName\AppData\Local\IhrAnwendungsname\...* Wenn Sie beispielsweise in *TextBox1* den Text »Wie geht es Ihnen?« und in *TextBox2* »Heute schlechter als gestern, aber besser als morgen!« eingetragen haben und das Häkchen von *CheckBox1* aktiviert haben, dann erhält nach Schließen des Formulars die Datei *user.config* das folgende Aussehen (gespeicherte Werte fett gedruckt):

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <userSettings>
    <WindowsFormsApplication1.Properties.Settings>
      <setting name="Wert1" serializeAs="String">
        <value>Wie geht es Ihnen?</value>
      </setting>
```

```

    <setting name="Wert2" serializeAs="String">
      <value>Heute schlechter als gestern, aber besser als morgen!</value>
    </setting>
    <setting name="Wert3" serializeAs="String">
      <value>True</value>
    </setting>
  </WindowsFormsApplication1.Properties.Settings>
</userSettings>
</configuration>

```

Variante 2: Abspeichern in einer Binärdatei

Mit dieser Variante, die (einmalig) etwas mehr Aufwand erfordert, bleiben Sie völlig unabhängig von den Anwendungseinstellungen.

```

...
Der Namespace für Dateioperationen:
Imports System.IO

Public Class Form1
  ...
Der Name der Binärdatei, die sich hier direkt im Anwendungsverzeichnis befindet:
  Private pfad As String = "Einstellungen.dat"
Die Methode zum Laden der Einstellungen:
  Dim rStream As New FileStream(pfad, FileMode.OpenOrCreate, FileAccess.Read)
  Dim binReader As New BinaryReader(rStream)

  If rStream.Length > 0 Then
Hier beginnt der UI-spezifische Code:
    TextBox1.Text = binReader.ReadString()
    TextBox2.Text = binReader.ReadString()
    CheckBox1.Checked = binReader.ReadBoolean()
    ...
Hier endet der UI-spezifische Code.
  End If
  binReader.Close()
  rStream.Close()
End Sub
Die Methode zum Speichern der Einstellungen:
  Private Sub writeFile()
    Dim wStream As New FileStream(pfad, FileMode.OpenOrCreate, FileAccess.Write)
    Dim binWriter As New BinaryWriter(wStream)

```

Hier beginnt der UI-spezifische Code:

```
binWriter.Write(textBox1.Text)
binWriter.Write(textBox2.Text)
binWriter.Write(checkBox1.Checked)
...
```

Hier endet der UI-spezifische Code.

```
binWriter.Flush()
binWriter.Close()
wStream.Close()
End Sub
```

Beim Laden des Formulars werden die UI-Einstellungen eingelesen. Falls die Datei *Einstellungen.dat* nicht vorhanden ist, wird automatisch eine neue leere Datei angelegt:

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles MyBase.Load
    readFile()
End Sub
```

Beim Schließen des Formulars werden die Einstellungen automatisch abgespeichert:

```
Private Sub Form1_FormClosing(ByVal sender As Object, _
    ByVal e As System.Windows.Forms.FormClosingEventArgs) Handles Me.FormClosing
    writeFile()
End Sub
End Class
```

Wie Sie sehen, ist der zusätzliche Aufwand für Erzeugen, Lesen und Schreiben der Binärdatei nur einmal erforderlich. Für jedes UI-Control ist dann sowohl in der *readFile*- als auch in der *writeFile*-Methode jeweils nur eine Codezeile hinzuzufügen.

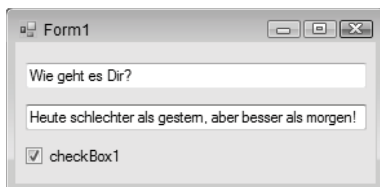


Abbildung 25.5
Laufzeitansicht

25.2 ... die Anzeige löschen?

Hat eine Benutzerschnittstelle zahlreiche gleichartige Steuerelemente, wie Textboxen oder Kontrollkästchen, so wäre es ziemlich aufwändig, deren Inhalte einzeln zu löschen bzw. zurückzusetzen. Einfacher kommt man zum Ziel, wenn man über alle Steuerelemente des Formulars iteriert und dabei die einzelnen Typen herausfiltert und gemeinsam behandelt.

**Hinweis**

Dieses Rezept ist auch ein gutes Beispiel für Typisierung. ■

Benutzerschnittstelle

Fügen Sie einem Windows Form einige Text- und ComboBox-Komponenten hinzu (siehe Laufzeitansicht).

Programmierung

```
Public Class Form1

    Private Sub button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
        Handles button1.Click
        For i As Integer = 0 To Controls.Count - 1
            Dim c As Control = Controls(i)
            If TypeOf c Is TextBox Then c.Text = String.Empty
            If TypeOf c Is CheckBox Then CType(c, CheckBox).Checked = False
        Next
    End Sub

End Class
```

Test

Nach dem Klick auf die Schaltfläche verschwinden die Inhalte aller Textfelder und Check-boxen:

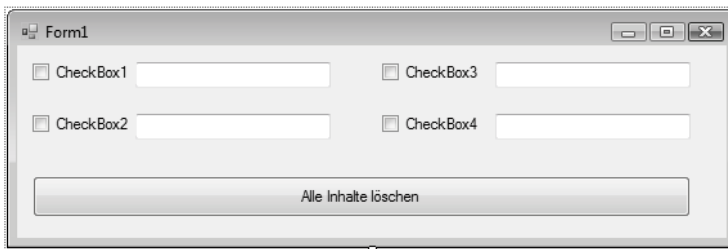


Abbildung 25.6 Laufzeitansicht

25.3 ... die Maus abfragen?

Im Ereignisparameter *e* eines Maus-Ereignisses werden sowohl die aktuellen Koordinaten übermittelt als auch die Information darüber, welche Maustaste gedrückt wurde. Das vorliegende Rezept zeigt dazu ein einfaches Beispiel.

Benutzerschnittstelle

Auf das Startformular Form1 setzen wir lediglich zwei dicke fette Label zwecks Koordinatenanzeige (siehe Laufzeitabbildung).

Programmierung

```
Public Class Form1
    Private rec As Rectangle = New Rectangle(50, 50, 100, 150)
```

Im Paint-Ereignis des Formulars wird das Rechteck gezeichnet:

```
Private Sub Form1_Paint(ByVal sender As System.Object, _
    ByVal e As System.Windows.Forms.PaintEventArgs) Handles MyBase.Paint
    e.Graphics.DrawRectangle(New Pen(Color.Black), rec)
End Sub
```

Jede Mausbewegung löst das MouseMove-Event aus, welches wir wie folgt besetzen:

```
Private Sub Form1_MouseMove(ByVal sender As System.Object, _
    ByVal e As System.Windows.Forms.MouseEventArgs) Handles MyBase.MouseMove
    Label1.Text = "X = " & e.X.ToString()
    Label2.Text = "Y = " & e.Y.ToString()
```

Die Contains-Methode des Rectangle-Objekts ermittelt, ob die Mauskoordinaten innerhalb des Rechtecks liegen. Die Ereignisdaten werden dem übergebenen Objekt e entnommen:

```
If (rec.Contains(e.X, e.Y)) AndAlso _
    (e.Button = System.Windows.Forms.MouseButtons.Right) Then
    Me.Cursor = Cursors.WaitCursor ' Mauszeiger ändert sich (Sanduhr)
Else
    Me.Cursor = Cursors.Default ' Mauszeiger wird zurückgesetzt (Pfeil)
End If
End Sub
End Class
```

Test

Die aktuellen Mauskoordinaten werden kontinuierlich angezeigt. Sobald Sie den Cursor im Bereich des Rechtecks bewegen und dabei die rechte Maustaste drücken, wird er sein Aussehen ändern.

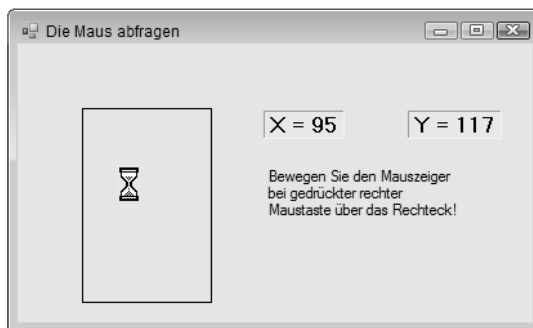


Abbildung 25.7
Laufzeitansicht

25.4 ... Dezimalkomma in Dezimalpunkt umwandeln?

Das Rezept zeigt, wie ein eingegebenes Zeichen nach jedem Tastendruck auf das Komma geprüft und dann »gewaltsam« in einen Punkt verwandelt wird.

Benutzerschnittstelle

Eine `TextBox` – mehr brauchen Sie nicht für diesen kleinen Test. Sie können – müssen aber nicht – die `TextAlign`-Eigenschaft der `TextBox` auf `Right` setzen und außerdem die Schrift über die `Font.Size`-Eigenschaft etwas vergrößern.

Programmierung

```
Public Class Form1
```

Wir verwenden das `KeyPress`-Event der `TextBox`, um das Komma (aus dem übergebenen Ereignisobjekt `e`) herauszufiltern und stattdessen einen Punkt einzufügen.

```
Private Sub TextBox1_KeyPress(ByVal sender As Object, ByVal e As KeyPressEventArgs) _  
    Handles TextBox1.KeyPress  
    If e.KeyChar = "," Then e.KeyChar = CChar(".")  
End Sub
```

```
End Class
```

Test

Immer wenn Sie versuchen, ein Komma einzugeben, wird es »wie von Geisterhand« in einen Punkt umgewandelt.

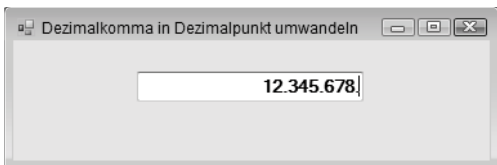


Abbildung 25.8
Laufzeitansicht