

User-Controls designerkonform ausstatten

Eigenschaften hinter Gittern

Eigene Benutzersteuerelemente sind mit .NET schnell erstellt. Ein Anwender möchte aber gern schon im Entwurfsmodus ein Steuerelement komfortabel mit einem Eigenschaftenfenster konfigurieren können. Hierzu bedarf es einiger Ergänzungen im Steuerelement.

Die Vorgehensweise ist jedem vertraut, der mit Visual Studio arbeitet: Ein Fenster im Entwurfsmodus anzeigen, auf ein Steuerelement klicken und dann die verfügbaren Eigenschaften dieses Steuerelements im Eigenschaftenfenster (PropertyGrid) anpassen. Windows- und Webanwendungen verhalten sich sehr ähnlich. Für beide Plattformen verwendet die IDE das gleiche Eigenschaftenfenster, und auch alle grundlegenden Techniken zum Darstellen der Eigenschaften von Steuerelementen sind identisch. Die hier beschriebenen Techniken lassen sich daher sowohl für Windows- als auch für Webcontrols einsetzen.

Darstellung der Eigenschaften im Entwurfsmodus

Selbst erstellte Steuerelemente enthalten in der Regel zusätzliche Funktionen, die

gleichermaßen im Eigenschaftenfenster sichtbar sein sollten. Der erste Schritt besteht daher darin, zum Einstellen und Anzeigen von Parametern Eigenschaften (Properties) zu definieren. Das Eigenschaftenfenster zeigt dann alle Eigenschaften an, die als öffentlich deklariert wurden. Die Informationen hierzu bezieht es via Reflection über das *Type*-Objekt. Eine Property-Definition wie

```
private int neueEigenschaft;
public int NeueEigenschaft
{
    get {return neueEigenschaft;}
    set {neueEigenschaft = value;}
}
```

führt bei der Auswahl des Steuerelements im Designer zu der in Abbildung 1 gezeigten Darstellung im Eigenschaftenfenster. Allerdings erscheint sie in der Rubrik Sonstiges und enthält noch keine aussagekräftige Beschreibung. Stattdessen wird vorläufig einfach der Name *NeueEigenschaft* eingesetzt.

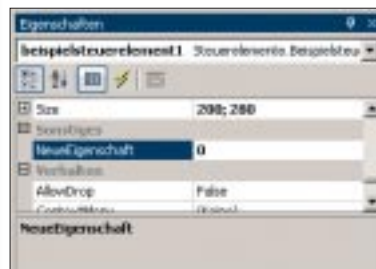


Abbildung 1 Der erste Schritt: Die Eigenschaft wird angezeigt.

Allgemeine Richtlinien

Damit eine Eigenschaft angezeigt werden kann, muss sie öffentlich sein und einen *get*-Accessor besitzen. Dieser wird immer

dann aufgerufen, wenn das Eigenschaftenfenster den Wert abrufen muss. Daher ist es äußerst wichtig, dass der Accessor lediglich die gewünschten Werte zurückgibt, ansonsten aber keinerlei Seiteneffekte hervorruft. Zustandsänderungen jeglicher Art sind hier somit tabu.

Besitzt die Eigenschaft auch einen *set*-Accessor, lässt sich ihr Wert im Eigenschaftenfenster auch ändern. Zur Ausnahme von der Regel weiter unten mehr. Ansonsten ist sie auch im Eigenschaftenfenster schreibgeschützt, wie Sie es an der grauen Schrift der Eigenschaft *Radius* in Abbildung 2 sehen.

Änderungen von Eigenschaften, die die Darstellung des Steuerelementes beeinflussen, sollten im *set*-Accessor das Neuzeichnen anstoßen, um so dem Anwender ein visuelles Feedback zu geben.

Layout über Attribute

Als Beispiel zur Demonstration soll ein einfaches Steuerelement zur Darstellung eines Vielecks dienen. Über die Eigenschaften *AnzahlEcken* und *Startwinkel* werden die Geometrieparameter festgelegt, über *Farbe1* und *Farbe2* die Farben für eine Verlaufsfüllung. Die Eckpunkte des Vielecks liegen auf einem Kreis um den Mittelpunkt des Steuerelements, dessen Radius automatisch berechnet wird. Der Radius wird ebenfalls als Eigenschaft angezeigt, kann allerdings nicht geändert werden. Alle genannten Eigenschaften werden mit Kurzbeschreibungen versehen und in der Kategorie *Darstellung* angezeigt (Abbildung 2).

Viele Darstellungen im Eigenschaftenfenster werden über Attribute gesteuert (Listing 1). Den im Attribut *Description* angegebenen Text zeigt Visual Studio bei Auswahl der Eigenschaft im Beschreibungsfeld des Eigenschaftenfensters an.

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/



dotnetpro.code
A0411Property



Sprachen C#

Technik User Controls

Voraussetzungen Visual Studio .NET 2003

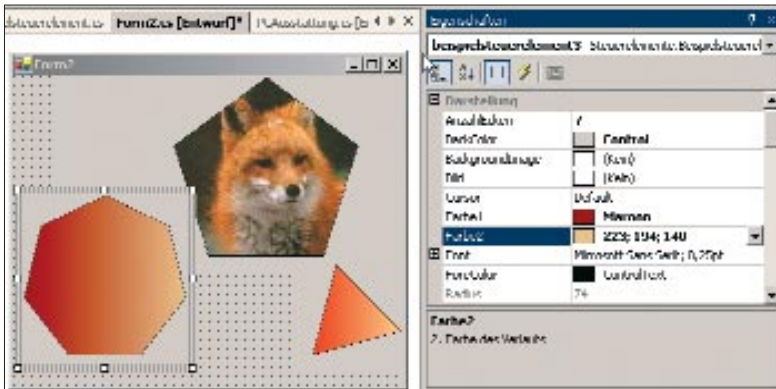


Abbildung 2 Die verbesserte Anzeige der Eigenschaften.

Beim Sortieren der Eigenschaften nach Kategorien ist das Attribut *Category* ausschlaggebend. Sie können eigene Kategorien definieren oder vorhandene verwenden. Möchten Sie die Eigenschaft sprachunabhängig in einer der Standardkategorien unterbringen, verwenden Sie den englischen Namen. Für die Kategorie *Darstellung* wäre das *Appearance*.

Schreibschutz und Sichtbarkeit steuern

Um eine Eigenschaft, die sowohl über einen *get*- als auch einen *set*-Accessor verfügt, vor Änderungen im Eigenschaftfenster zu schützen, können Sie das *ReadOnly*-Attribut verwenden:

```
[... ReadOnly(true)]
public int Radius {...}
```

So lässt sich der Eigenschaft im Code ein Wert zuweisen, nicht jedoch in der Entwurfsansicht. Um eine Eigenschaft in

der IDE vollständig zu verbergen, verwenden Sie das *Browsable*-Attribut:

```
[Browsable(false)]
public int UnsichtbareEigenschaft {...}
```

Unzulässige Werte und Standardwerte

Die Eigenschaft *AnzahlEcken* soll definitionsgemäß auf die Werte von drei bis zehn beschränkt werden. Im *set*-Accessor der Eigenschaft wird daher der Wert geprüft und bei Überschreiten des Definitionsbereichs eine Ausnahme ausgelöst (Listing 2). Bei der Eingabe eines ungültigen Wertes weist das Eigenschaftfenster mit einer Dialogbox darauf hin (Abbildung 3).

Zur Definition von Standardwerten sind zwei Schritte erforderlich. Zum einen muss der Wert über das Attribut *De-*

Listing 2

Werte auf Gültigkeit prüfen und Standardwerte definieren.

```
// Öffentliche Eigenschaft
[Description("Anzahl der Ecken des zu zeichnenden Vielecks (3-10)"),
Category("Appearance"),
DefaultValue(3)]
public int AnzahlEcken {
    get { return anzahlEcken; }
    set {
        // Wertebereich prüfen
        if (value < 3 || value > 10)
            throw new ArgumentOutOfRangeException(
                "Wert muss zwischen 3 und 10 liegen");
        // Wert speichern und Control neu zeichnen
        anzahlEcken = value;
        PunkteBerechnen();
        this.Refresh();
    }
}
```

faultValue festgelegt werden, zum anderen sollte im Code dafür gesorgt werden, dass die Eigenschaft nach der Erstellung des Steuerelements auch diesen Wert besitzt. Vom Standardwert abweichende Werte stellt das Eigenschaftfenster mit fester Schrift dar.

Außerdem hat das Attribut Auswirkungen auf die Code-Generierung. Nur wenn der Wert nicht dem Standardwert entspricht, erzeugt die Entwicklungsumgebung Code für die Zuweisung in *InitializeComponent*. Ansonsten entfällt der betreffende Befehl. Für *AnzahlEcken = 7* wird diese Zeile erzeugt, während sie für *AnzahlEcken = 3* unterdrückt wird:

```
this.beispielsteuerelement1.AnzahlEcken = 7;
```

Attributwert und Initialwert müssen also übereinstimmen, da sonst unter Umständen beim Programmstart die Eigenschaft nicht den richtigen Wert besitzt. Hierfür ist der Programmierer selbst verantwortlich, denn die IDE prüft dies nicht.

Standardwerte für Datentypen, die keine Konstanten kennen

Das *DefaultValue*-Attribut erlaubt die Angabe von Konstanten für die gängigen Datentypen. Andere Typen, zum Beispiel die Struktur *Color*, besitzen aber keine Konstanten. Hier muss man sich eine Überladung des Attribut-Konstruktors zu Nutze machen, die den Wert als String sowie den erforderlichen Datentyp übernimmt. Dieser Konstruktor benutzt einen

Listing 1

Attribute geben Kurzbeschreibung und Kategorie vor.

```
// Öffentliche Eigenschaft
[Description("Startwinkel zum Zeichnen des Vielecks"),
Category("Appearance")]
public int Startwinkel {
    get { return startwinkel; }
    set {
        // Wert speichern und Control neu zeichnen
        startwinkel = value;
        PunkteBerechnen();
        Refresh();
    }
}
```

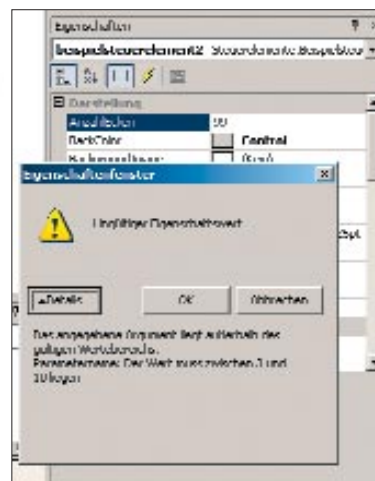


Abbildung 3 Durch Auslösen von Exceptions lassen sich fehlerhafte Eingaben zurückweisen.

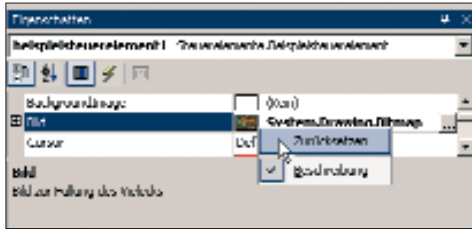


Abbildung 4 Das Löschen einer Auswahl für Eigenschaften vom Typ Bitmap ist in diesem Fall nur möglich, wenn als Standardwert null definiert wurde.

TypeConverter zur Umwandlung des Strings in den Zieltyp. Attribute für Eigenschaften vom Typ *Color* können für eine benannte Farbe so aussehen:

```
[DefaultValue(typeof(Color), "Red")] ...
```

Für einen RGB-Farbwert dagegen heißt das Attribut:

```
[DefaultValue(typeof(Color), "100, 50, 250")]
```

Die vollständige Implementierung der Farbeigenschaften sehen Sie in Listing 3. Der Code zum Zeichnen des Steuerelements ist hier nicht von Bedeutung. Sie finden ihn aber wie alle Listings auf der Heft-CD. Wichtig ist die Definition des Standardattributs bei Referenztypen, die auch *null*-Werte annehmen. Beispielsweise stellt das Eigenschaftenfenster für Eigenschaften vom Typ *Bitmap* den Datei-Öffnen-Dialog für unterstützte Bilddateien zur Verfügung. Um eine Auswahl rückgängig machen zu können, muss *null* als Standardwert vorgegeben werden. Nur dann lässt sich der Eintrag wieder löschen (Abbildung 4):

```
protected Bitmap bild = null;
```

```
[... DefaultValue(typeof(Object), null)]
public Bitmap Bild {...}
```

Dialog zum Öffnen beliebiger Dateien

Soll der Anwender in einer Eigenschaft einen beliebigen Dateipfad mithilfe eines Datei-Öffnen-Dialogs auswählen können (Abbildung 5), dann muss der Eigenschaft ein *UITypeEditor* zugewiesen werden. Ein solcher Editor dient dazu, ein Dialogfenster zur Definition des Eigenschaftswertes anzuzeigen und kann dadurch weit mehr Komfort bieten als die direkte Eingabe im *PropertyGrid*.

Für viele Zwecke gibt es bereits vorgefertigte *UITypeEditors* im Framework. Sie müssen die entsprechende Klasse für die Dateiauswahl nicht neu erfinden, es gibt sie bereits:

```
System.Windows.Forms.Design.FileNameEditor
```

Allerdings kann diese Klasse nicht direkt verwendet werden, da ihr Konstruk-

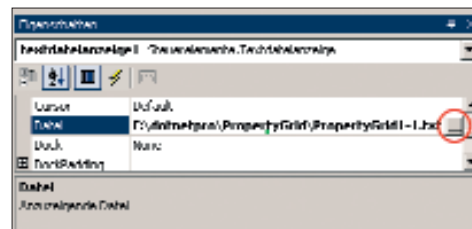


Abbildung 5 Der Wert der Eigenschaft Datei lässt sich mithilfe eines Dateidialogs auswählen.

tor keine Einschränkung für die auswählbaren Dateitypen vorsieht. Stattdessen wird eine zusätzliche Klasse *TextfilenameEditor* von *FileNameEditor* abgeleitet (siehe Listing 4).

Klickt ein Anwender auf die Dialogschaltfläche mit den drei Punkten, dann wird die Instanzmethode *InitializeDialog* aufgerufen. Um einen Dateifilter zu installieren genügt es, diese Methode zu überschreiben und die Eigenschaft *Filter* auf den gewünschten Wert zu setzen.

Standardeigenschaft und Standardereignis

Zunächst sei noch erwähnt, dass die Attribute *Description* und *Category* auch für die Definition von Ereignissen eingesetzt werden können. In der Darstellung der Ereignisliste, wie sie im Eigenschaftenfenster bei der C#-Programmierung (leider nicht bei VB.NET) vorgenommen wird, dienen sie hier ebenso der Erläuterung und Strukturierung wie bei den Eigenschaften.

Für eine Steuerelementklasse kann aber zusätzlich mithilfe der Attribute *DefaultProperty* und *DefaultEvent* eine Eigenschaft oder ein Ereignis als Standard festgelegt werden. Während das Attribut für Eigenschaften nur wenig Bedeutung hat – nämlich die automatische Auswahl

Listing 3

Default-Werte für Color-Strukturen definieren.

```
protected Color farbe1 = Color.Red;
[Description("1. Farbe des Verlaufs"),
 Category("Appearance"),
 DefaultValue(typeof(Color), "Red")]
public Color Farbe1
{
    get {return farbe1;}
    set
    {
        farbe1 = value;
        Refresh();
    }
}
```

```
protected Color farbe2 = Color.FromArgb(100, 50, 250);
[Description("2. Farbe des Verlaufs"),
 Category("Appearance"),
 DefaultValue(typeof(Color), "100, 50, 250")]
public Color Farbe2
{
    get {return farbe2;}
    set
    {
        farbe2 = value;
        Refresh();
    }
}
```

Listing 4

Ein eigener UITypeEditor für die Auswahl von Textdateien.

```
// Verweis auf System.Design.dll hinzufügen
public class Textdateianzeige : System.Windows.Forms.UserControl
{
    ...

    // Die geschützte Member-Variablen für den Pfad
    protected string datei = "";

    // Für die öffentliche Eigenschaft wird der unten
    // definierte UITypeEditor festgelegt
    [System.ComponentModel.Editor(typeof(TextfilenameEditor),
        typeof(System.Drawing.Design.UITypeEditor)),
        Description("Anzuzeigende Datei"),
        DefaultValue("")]
    public string Datei
    {
        get { return datei; }
        set
        {
            datei = value;
            if (datei != "")
            {
                System.IO.StreamReader sr = new System.IO.StreamReader(datei);
                LBLText.Text = sr.ReadToEnd();
                sr.Close();
            }
            else
            {
                LBLText.Text = "Kein Text geladen";
            }
        }
    }

    // UITypeEditor für die Auswahl von Textdateien
    public class TextfilenameEditor
        : System.Windows.Forms.Design.FileNameEditor
    {
        protected override void InitializeDialog(
            OpenFileDialog openFileDialog)
        {
            openFileDialog.Filter = "Text-Dateien (*.txt)|*.txt";
        }
    }
}
```

dieser Eigenschaft, wenn zuvor keine andere ausgewählt war –, ist das *Default-Event*-Attribut besonders wichtig. Es gibt vor, welche Ereignisprozedur anzulegen und anzuzeigen ist, wenn der Anwender im Entwurfsmodus einen Doppelklick auf das Steuerelement ausführt. Tabelle 1 zeigt noch einmal die bislang beschriebenen Attribute und ihre Bedeutungen.

Enumerationen

Waren Enumerationen in früheren Programmiersprachen lediglich benannte Integer-Konstanten, so bietet .NET wesentliche Erweiterungen. Über die Klasse *Enum* lassen sich beispielsweise Werte und Namen zur Laufzeit abrufen. Kein

Wunder also, dass auch das PropertyGrid-Control bereits entsprechende Funktionalität zum Anzeigen und Ändern von Eigenschaften, die auf einem Enum-Typ basieren, an Bord hat. Definiert man beispielsweise eine Eigenschaft folgenden Typs

```
public enum Betriebssystemversionen
{
    Windows95,
    Windows98,
    WindowsME,
    Windows2000,
    WindowsXP
}
```

dann lässt sich im Eigenschaftfenster die gewünschte Auswahl treffen (Abbildung 6).

Sollen mehrere Werte gleichzeitig auswählbar sein, müssen sie zwei Bedingungen erfüllen:

- Jeder Wert muss ein Bit darstellen, um Oder-Verknüpfungen zuzulassen
- Damit das Eigenschaftfenster die Bitverknüpfungen korrekt anzeigt, muss der Enumerationstyp mit dem *Flags*-Attribut versehen werden.

Als Beispiel dient eine Aufzählung möglicher Software-Komponenten eines PCs. Die Zahlenwerte der Elemente sind 2er-Potenzen (Listing 5). Im *set*-Accessor der Eigenschaft *Ausstattung* werden alle Werte der Enumeration *Komponenten* durchlaufen und jeweils geprüft, ob das betreffende Bit gesetzt ist. Falls ja, wird der Text in der ListBox aufgenommen. Abbildung 6 zeigt am Beispiel von *Ausstattung*, wie das Eigenschaftfenster Bitkombinationen darstellt: als kommaseparierte Liste. Der Designer erzeugt automatisch den Code für die benötigte Oder-Verknüpfung:

```
this.pcAusstattung1.Ausstattung =
    ((Steuerelemente.Komponenten)
    ((Steuerelemente.Komponenten.Textverarbeitung
    | Steuerelemente.Komponenten.Browser)));
```

Leider ist die Eingabe im Eigenschaftfenster etwas holpriger. Denn auswählen lassen sich nur einzelne Elemente aus der Liste. Kombinationen muss man im Klartext eingeben.

Tabelle 1

Typische Attribute zur Steuerung der Darstellung im Eigenschaftfenster.

Attribut	Bedeutung im Eigenschaftfenster
Description	Textbeschreibung.
Category	Kategorie, in der die Eigenschaft oder das Ereignis angezeigt werden soll.
DefaultValue	Standardwert der Eigenschaft.
Browsable	Darstellung ein- oder ausschalten.
ReadOnly	Schreibschutz.
DefaultProperty	Standardeigenschaft eines Controls.
DefaultEvent	Standardereignis eines Controls.
Editor	Auswahl eines speziellen Editors.

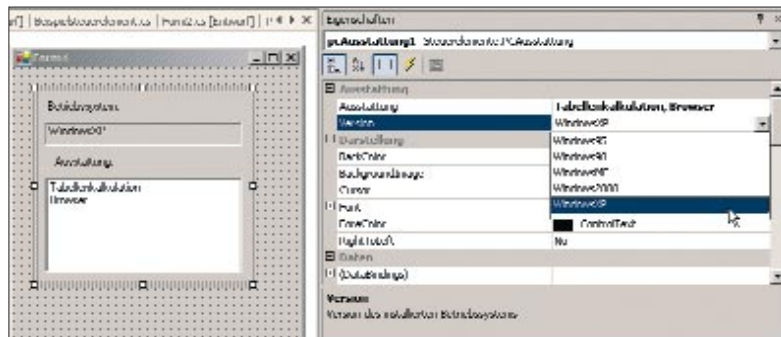


Abbildung 6 Auch Enumerationskonstanten lassen sich zur direkten Auswahl anzeigen.

Kommunikation zwischen Steuerelement und Designer

Um dem Anwender etwas mehr Komfort bieten zu können, sollte es möglich sein, bereits zur Entwurfszeit das Steuerelement zu bedienen.

In Abbildung 7 sehen Sie ein Steuerelement, das basierend auf dem oben beschriebenen Enumerationstyp eine Reihe von Kontrollkästchen anzeigt. Der Pfeil als Mauszeiger deutet schon an, dass an der betreffenden Stelle mit der Maus die Kontrollkästchen gesetzt werden können und nicht, wie erwartet, das Steuerelement aktiviert und verschoben wird. So lässt sich die entsprechende Einstellung *Softwareausstattung* direkt auf dem Steuerelement zusammenstellen. Das Eigenschaftenfenster aktualisiert dabei automatisch die Anzeige.

Zur Laufzeit werden Änderungen konventionell über das Event *AusstattungGeändert* gemeldet.

Wie Sie Listing 6 entnehmen können, ist die Implementierung hier nicht mehr mit ein paar einfachen Attributen zu erledigen. Damit das Steuerelement überhaupt zur Entwurfszeit Mausereignisse entgegennehmen kann, muss der Steuerelement-Klasse per Attribut eine *ControlDesigner*-Klasse zugewiesen werden. In der Überschreibung der Methode *GetHitTest* wird berechnet, ob sich der Mauszeiger über einem zu bedienenden Steuerelement (hier den Kontrollkästchen) befindet. Die Rückgabe von *true* bewirkt dann, dass die Mausereignisse in diesem Bereich automatisch an das betreffende Steuerelement weitergeleitet werden, so dass beispielsweise ein Klick auf ein Kontrollkästchen die Ereignismethode *cb_Click* aufruft. In ihr wird der sich ergebende neue Eigenschaftswert ermittelt.

Ein Problem besteht allerdings noch darin, dem Eigenschaftenfenster die Änderung der Eigenschaft mitzuteilen. Denn automatisch erfährt es ja nicht, wenn sich

der Wert einer angezeigten Eigenschaft geändert hat. Es muss also ein Weg gefunden werden, vom Code des Steuerelements heraus einem Designer, sofern vorhanden, Informationen zukommen zu lassen.

Nun gibt es verschiedene Möglichkeiten, um herauszufinden, welcher Designer wie benachrichtigt werden muss. Im Beispiel bietet sich eine einfache Lösung an. Die als Attribut angegebene Designer-Klasse muss von der Entwicklungsumgebung instanziiert werden. Danach wird die virtuelle Methode *Initialize* aufgerufen. Durch Überschreiben dieser Methode kann dem angeschlossenen Steuerelement die Referenz des Designer-Objektes bekannt gemacht werden. Über diese Referenz kann dann im Code des Steuerelements die Methode *DesignerInformieren* der Klasse *ZubehörControlDesigner* aufgerufen werden. Diese gibt die Änderungsinformationen mithilfe der Methode *RaiseComponentChanged* weiter.

Im gemeinsamen Klick-Ereignis *cb_Click* der Kontrollkästchen wird also geprüft, ob die Referenz des Designers ungleich *null* ist und dann über diese Referenz indirekt *RaiseComponentChanged* aufgerufen. So wird das Eigenschaftenfenster von Änderungen, die durch Anklicken der Kontrollkästchen hervorgerufen worden sind, in Kenntnis gesetzt.

Verben zur freien Programmierung nutzen

Gibt es für die Eigenschaften eines Steuerelements sinnvolle Voreinstellungen, dann kann man dem Anwender die Arbeit

Listing 5

Einen Enum-Typ mit kombinierbaren Bitwerten definieren und für Eigenschaften einsetzen.

```
// Aufzählung der möglichen Komponenten
[Flags]
public enum Komponenten
{
    Textverarbeitung = 0x01,
    Tabellenkalkulation = 0x02,
    Bildbearbeitung = 0x04,
    Browser = 0x08,
    Videoschnitt = 0x10
}

...

protected Komponenten ausstattung;

[Description("Installierte Anwendungssoftware"),
Category("Ausstattung"),
DefaultValue(Komponenten.Textverarbeitung)]
public Komponenten Ausstattung
{
    get { return ausstattung; }
    set
    {
        ausstattung = value;

        // Listbox löschen
        LBAusstattung.Items.Clear();

        // Ausgewählte Elemente in Listbox aufnehmen
        foreach (Komponenten k in Enum.GetValues(typeof(Komponenten)))
            if ((ausstattung & k) != 0)
                LBAusstattung.Items.Add(Enum.GetName(typeof(Komponenten), k));
    }
}
```

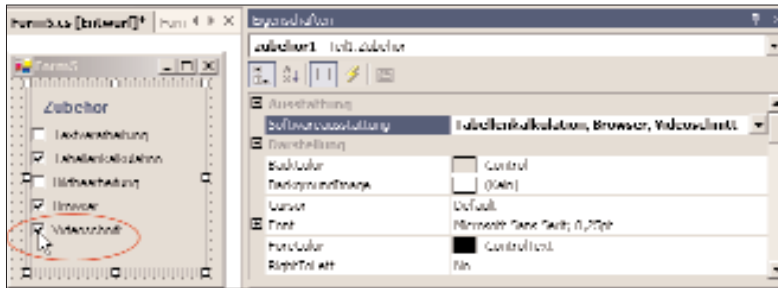


Abbildung 7 Ein Steuerelement kann auch zur Entwurfszeit bedienbar sein.

erleichtern, indem man Verben definiert, die im Eigenschaftfenster als Hyperlink-Tasten dargestellt werden (Abbildung 8). Die ausgelöste Aktion lässt sich frei programmieren. So können zum Beispiel Eigenschaften mit Standardwerten belegt oder ein spezieller Editor angezeigt werden. Steuerelemente, die Verben benutzen, sind zum Beispiel das TabControl und das DataGrid. Zur Demonstration der Vorgehensweise wird das Beispiel durch das Verb *Büroausstattung* erweitert. Bei Klick auf die Schaltfläche sollen die Kontrollkästchen für Textverarbeitung, Tabellenkalkulation und Browser gesetzt und die anderen zurückgesetzt werden.

Wurde einer Steuerelementklasse ein entsprechendes Designer-Attribut zugewiesen, dann ruft das PropertyGrid-Control die Eigenschaft *Verbs* dieses Designers ab. Diese gibt im Normalfall eine leere Liste zurück, lässt sich jedoch überschreiben. Die Implementierung des *get-Accessors* von *Verbs* gestaltet sich relativ

einfach. Für die benötigten Verben werden Instanzen angelegt, an einen Eventhandler gebunden und in einer Auflistung vom Typ *DesignerVerbCollection* aufgenommen.

Dann müssen lediglich noch die Eventhandler programmiert werden. Hier werden die geforderten Aktionen codiert. Abschließend müssen dem Eigenschaftfenster über *RaiseComponentChanged* Änderungen an den Eigenschaften mitgeteilt werden.

Hilfe, ich sehe nur eine Fehlermeldung!

Keine Panik! Auch im Entwurfsmodus können Sie den Debugger benutzen. Wie, das können Sie in [1] nachlesen.

Fazit

Pfiffige Steuerelemente zeichnen sich nicht nur durch ihre Funktionalität zur Laufzeit aus, sondern auch durch ihre

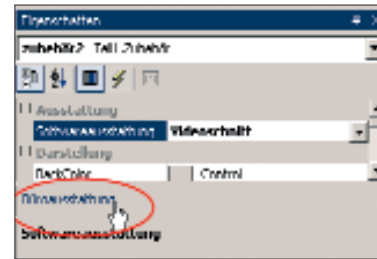


Abbildung 8 Über Verben können besondere Aktionen angeboten werden.

Stärken in der Entwicklungsumgebung. Der zusätzliche Programmieraufwand ist nicht zu unterschätzen, lohnt sich aber allemal. Aus Platzgründen konnten in diesem Artikel nur die grundlegenden Techniken erläutert werden. VB.NET-Programmierer finden in [2] ein umfangreiches Kapitel zum Thema PropertyGrid.

Übrigens: Das PropertyGrid-Control ist nicht auf den Designer der Entwicklungsumgebung beschränkt, sondern steht jeder Windows-Anwendung zur Verfügung. Sie können es in eigenen Programmen einsetzen und zur Darstellung der Eigenschaften beliebiger Objekte benutzen. ||||| >

[1] Joachim Fuchs, Designer im Debug-Streik, Windows UserControls und Forms im Designer debuggen, dotnetpro 3/2004, Seite 82 ff.

[2] Joachim Fuchs, Andreas Barchfeld, Das Visual Basic .NET Codebook, Addison-Wesley 2004, ISBN 3-8273-2007-0



Listing 6

Zur Entwurfszeit müssen dem Designer Änderungen an den Eigenschaften eines Steuerelementes mitgeteilt werden.

```
[DefaultProperty("Softwareausstattung"),
Designer(typeof(Zubehör.ZubehörControlDesigner)),
DefaultEvent("AusstattungGeändert")]
public class Zubehör : System.Windows.Forms.UserControl
{
    // Eventhandler für Ausstattungsänderungen
    public event Zubehör.EventHandler AusstattungGeändert;

    // Aufzählung der möglichen Komponenten
    [Flags]
    public enum Komponenten
    {
        Nichts           = 0x0,
        Textverarbeitung = 0x01,
        Tabellenkalkulation = 0x02,
        Bildbearbeitung  = 0x04,
        Browser          = 0x08,
        Videoschnitt    = 0x10
    }

    // Array der benötigten Kontrollkästchen
    protected CheckBox[] Checkboxes;

    // Referenz eines angekoppelten Designers
    protected ZubehörControlDesigner designer;

    ...

    public Zubehör()
    {
        // Dieser Aufruf ist für den Windows Form-Designer erforderlich.
        InitializeComponent();

        // Anlegen der benötigten Checkboxes
        Komponenten[] werte = (Komponenten[])Enum.GetValues(
            typeof(Komponenten));
        Checkboxes = new CheckBox[werte.Length-1];

        int i = 0;
        foreach (Komponenten k in werte)
        {
            if (k != 0)
            {
                Checkboxes[i] = new CheckBox();
                Checkboxes[i].Text = Enum.GetName(typeof(Komponenten), k);
                Checkboxes[i].Tag = k;
                Checkboxes[i].Location = new Point(0, 30 + i*20);
                Checkboxes[i].Size = new Size(200, 20);
                Controls.Add(Checkboxes[i]);
                Checkboxes[i].Click +=new EventHandler(cb_Click);
                i++;
            }
        }

        // Setzen des Standardwertes
        Softwareausstattung = 0;
    }

    ...

    private void cb_Click(object sender, EventArgs e)
    {
        Komponenten alt = softwareausstattung;

        // Angeklickte CheckBox ermitteln und
        // zugehöriges Bit setzen oder löschen
        CheckBox cb = (CheckBox)sender;
        Komponenten k = (Komponenten)cb.Tag;
        if (cb.Checked)
            softwareausstattung |= k;
        else
            softwareausstattung &= ~k;

        // Wenn ein Designer angemeldet ist, diesen informieren
        if (designer != null)
            designer.DesignerInformieren(alt, softwareausstattung);

        // Änderung melden (öffentliches Event)
        if (AusstattungGeändert != null)
            AusstattungGeändert(this, new ZubehörEventArgs(
                softwareausstattung));
    }

    // Implementierung der Eigenschaft

    // Handler-Delegate für das Ereignis
    public delegate void ZubehörEventHandler(object sender,
        ZubehörEventArgs e);

    // Designer-Klasse zur Unterstützung der Bedienung im Entwurfsmodus
    // System.Design.dll einbinden!
    public class ZubehörControlDesigner :
        System.Windows.Forms.Design.ControlDesigner
    {
        protected override bool GetHitTest(Point point)
        {
            // Rückgabe true, wenn Mauszeiger im Bereich der CheckBox
            Point pt = this.Control.PointToClient(point);
            return (pt.Y>30) && (pt.X < 20);
        }

        public override void Initialize(IComponent component)
        {
            base.Initialize(component);
            // Referenz der Designer-Instanz im Steuerelement speichern
            ((ZubehörControl).designer = this;
        }

        public void DesignerInformieren(Komponenten alt, Komponenten neu)
        {
            // Eigenschaftenfenster über Änderungen informieren
            PropertyDescriptor pd = TypeDescriptor.GetProperties(
                Control).Find("Softwareausstattung",true);
            RaiseComponentChanged(pd, alt, neu);
        }
    }
}
```