Kapitel 19

Wunderwelt der Eigenschaften

Alle ActiveX-Komponenten können Eigenschaften haben. Aber was macht die Eigenschaften von ActiveX-Steuerelementen so besonders?

Der große Unterschied ist die Persistenz. Die Eigenschaften eines Steuerelements können zur Entwurfszeit eines Containers gesetzt und die Werte zusammen mit einer Applikation gespeichert werden. Das bedeutet aber gleichzeitig, daß Sie genau auf die Unterschiede zwischen Entwurfszeit- und Laufzeitverhalten von Eigenschaften achten müssen.

19.1 Eigenschaftsdatentypen

Sie sind vielleicht mit den normalen Datentypen wie Long, String, Variant und Object ausreichend vertraut. Bei ActiveX-Steuerelementen kann der Eigenschaftstyp wesentlichen Einfluß darauf haben, wie der Container zur Entwurfszeit mit dem Steuerelement interagiert. Das Steuerelement `ch19ctla` im Projekt `ch19ctls.vbp` demonstriert einige wichtige Aspekte für Eigenschaftsdatentypen. Es gibt mehrere Eigenschaftstypen, die hier genauer beschrieben werden sollen.

19.1.1 Variants

Eine Eigenschaftsprozedur, die Sie einem Steuerelement mit Hilfe des Befehls EXTRAS/PROZEDUR zuordnen, erhält automatisch den Eigenschaftstyp `Variant`. Das ist übel, weil ein `Variant` der womöglich schlechteste Typ für eine ActiveX-Steuerelementeigenschaft ist.

Ein robustes ActiveX-Steuerelement sollte in der Lage sein, alle Werte zu verarbeiten, die der Container im Entwurfs- und im Laufzeitmodus setzt. Weil ein `Variant` beliebige Datentypen aufnehmen kann, bedeutet die Verwendung von Variants als Steuerelementdatentypen eine wesentliche Mehrarbeit für den Autor des Steuerelements, sowohl bei der Codierung als auch beim Testen.

Die Verwendung von Variants als Eigenschaften kann auch im Entwurfszeit-Eigenschaftsfenster Verwirrung stiften. Enthält der `Variant` einen Wert, den es in einen String umwandeln kann, erscheint die Eigenschaft im Eigenschaftenfenster (vorausgesetzt, die anderen Anforderungen, zu denen wir gleich kommen, sind ebenfalls erfüllt). Enthält der `Variant` jedoch einen Objektverweis, wenn das Steuerelement geladen wird, erscheint die Eigenschaft nicht. Handelt es sich bei der Eigenschaft zunächst um einen String, und sie wird später so gesetzt, daß sie einen Objektverweis enthält, kann es sein, daß im Eigenschaftenfenster eine leere Zeile erscheint (abhängig davon, wann der Container sein Eigenschaftenfenster aktualisiert), und es deutet nichts darauf hin, daß die Eigenschaft auf einen zulässigen Wert gesetzt ist.

Das Eigenschaftenfenster von Visual Basic setzt einen `Variant` immer auf den String-Datentyp, wenn der Entwickler die Eigenschaft bearbeitet.

Wenn Sie all diese Dinge der langen Liste der Nachteile hinzufügen, die für Variants bekannt sind, und die in Kapitel 10 beschrieben wurden, wird eines klar:

Wenn Sie keinen wirklich guten Grund haben, sollten Sie keine Variants als Eigenschaften verwenden. Aber zumindest sollten Sie sie nicht für Eigenschaften verwenden, die zur Entwurfszeit sichtbar sind.

Und nun bleiben Sie ruhig, denn gleich werden Sie engere Bekanntschaft mit einem dieser guten Gründe machen, weshalb doch Variants verwendet werden sollten.

Typische Eigenschaftsprozeduren für eine `Variant`-Eigenschaft sehen Sie hier:

```
Private m_Variant1 As Variant

    Public Property Get Variant1() As Variant
       Variant1 = m_Variant1
    End Property

    Public Property Let Variant1(ByVal vNewValue As Variant)
       m_Variant1 = vNewValue
       PropertyChanged "Variant1"
       Debug.Print "Variant type: " & VarType(m_Variant1)
    End Property
```

19.1.2 OLE_COLOR

Der Typ `OLE_COLOR` kann etwas irreführend sein. Sie glauben vielleicht, es handle sich dabei um einen speziellen Objekttyp, wie etwa eine Schrift oder ein Bild (worauf wir gleich noch zu sprechen kommen), aber eine `OLE_COLOR`-Variable ist einfach nur ein `Long` Integer mit 32 Bit. Es handelt sich nicht um ein Objekt, und er hat deshalb auch keine Eigenschaften oder Methoden. Es ist einfach nur ein anderer Name für eine `Long`-Variable; er kann unter Visual Basic `Long`-Variablen zugewiesen werden, und ihm können `Long`-Variablen zugewiesen werden.

Warum sollten Sie diesen Variablentyp dann überhaupt verwenden? Weil Visual Basic intelligent genug ist, um zu erkennen, daß dieser Variablentyp eine spezielle Behandlung im VB-Eigenschaftenfenster erfahren soll, ebenso wie im Hinblick auf Eigenschaftenseiten. Wenn eine Eigenschaft mit dem Typ `OLE_COLOR` definiert ist, fügt Visual Basic dem VB-Eigenschaftenfenster ein Popup-Menü zur Farbauswahl hinzu. Außerdem sorgt es für die Verbindung der Eigenschaft mit der Standard-Eigenschaftsseite für die Farbauswahl.

Hier eine typische Menge von Eigenschaftenprozeduren für eine `OLE_COLOR`-Eigenschaft:

```
Private m_Color1 As OLE_COLOR

    Public Property Get Color1() As OLE_COLOR
       Color1 = m_Color1
    End Property
```

```
Public Property Let Color1(ByVal vNewValue As OLE_COLOR)
   m_Color1 = vNewValue
   PropertyChanged "Color1"
End Property
```

19.1.3 OLE_TRISTATE

Der Datentyp `OLE_TRISTATE` ist ein Aufzählungswert und wird, wie jeder Aufzählungswert in Visual Basic, auch durch einen 32-Bit-Long-Wert dargestellt. Wenn Visual Basic diesen Datentyp erkennt, stellt es im Eigenschaftenfenster ein Dropdown-Listenfeld bereit, das die drei möglichen Werte für diesen Datentyp enthält: 0 – Nicht markiert, 1 – Markiert und 2 – Grau.

Beachten Sie, daß Visual Basic keine Beschränkung für die möglichen Werte der Eigenschaften eines Aufzählungstyps vorgibt. Das bedeutet, Sie müssen Ihren eigenen Code einfügen, der sicherstellt, daß die Eigenschaft nicht auf einen ungültigen Wert gesetzt wird.

Der folgende Code zeigt typische Eigenschaftsprozeduren für eine `OLE_TRISTATE`-Eigenschaft. Beachten Sie die Fehlerauswertung in der Anweisung `Property Let`.

```
Private m_TriState1 As OLE_TRISTATE

   Public Property Get MyTriState() As OLE_TRISTATE
      MyTriState = m_TriState1
   End Property

   Public Property Let MyTriState(ByVal vNewValue As OLE_TRISTATE)
      If vNewValue > 2 or vNewValue <0 Then
         Err.Raise 380
      End If
      m_TriState1 = vNewValue
      PropertyChanged "MyTriState"
   End Property
```

19.1.4 OLE_OPTEXCLUSIVE

Dieser Datentyp ist äquivalent zum Datentyp Boolean von Visual Basic, d.h. er kann nur die Werte `True` oder `False` annehmen. Sie können einer Variablen dieses Typs beliebige Werte zuweisen, sie wird jedoch automatisch auf -1 gesetzt, wenn der Wert ungleich 0 ist.

Wenn die Standardeigenschaft für ein Steuerelement diesen Datentyp hat, nimmt Visual Basic an, daß dieses sich wie ein Optionsfeld verhalten soll. Die Standardeigenschaft ist zu jedem Zeitpunkt jeweils nur für ein Steuerelement, das `OLE_OPTEXCLUSIVE` verwendet, auf True gesetzt. Wenn Sie eines auf `True` setzen, wird die Standardeigenschaft dieses Typs für alle anderen Steuerelemente auf `False` gesetzt.

Es gibt einige Dinge zu diesem Datentyp, die aus der Dokumentation nicht ganz klar werden:

- Im Gegensatz zu dem, was die Dokumentation sagt, hat die Verwendung des Datentyps `OLE_OPTEXCLUSIVE` nichts mit der Value-Eigenschaft zu tun, auch wenn sie in der Regel für diese Eigenschaft verwendet wird. Was zählt, ist, daß die zu setzende Eigenschaft im Dialogfeld EXTRAS/PROZEDURATTRIBUTE als Standardeigenschaft festgelegt wird. Dazu setzen Sie die Prozedur-ID im Dialogfeld auf (Voreinstellung). Ein Beispiel dafür sehen Sie im Steuerelement `ch19ctla`, wo die Standardeigenschaft statt Value den Namen `xValue` hat.
- Alle Steuerelemente mit einer Eigenschaft vom Typ `OLE_OPTEXCLUSIVE`, die einen bestimmten Container gemeinsam nutzen, werden vom Container als einzelne Gruppe betrachtet, unabhängig vom Typ des Steuerelements. Sie sehen das im Formular `ch19tst1`. Wenn Sie auf das Optionsfeld klicken, wird die `xValue`-Eigenschaft für beide `ch19ctla`-Steuerelemente auf `False` gesetzt.
- Dieses Verhalten ist völlig vom Container abhängig. Sie können nicht voraussetzen, daß diese Art von Optionsausschluß auf jedem Container implementiert ist. Die meisten Container scheinen sie jedoch korrekt zu unterstützen.

Typische Eigenschaftsprozeduren für eine `OLE_OPTEXCLUSIVE`-Eigenschaft sehen Sie im folgenden Code. Die Anweisung `Debug.Print` beobachtet den aktuellen Status der Variablen, so daß Sie sehen, wenn die Eigenschaft auf `False` gesetzt wird, nachdem die `OLE_OPTEXCLUSIVE`-Standardeigenschaft eines anderen Steuerelements auf `True` gesetzt wurde.

```
Private m_Value As OLE_OPTEXCLUSIVE

    Public Property Get xValue() As OLE_OPTEXCLUSIVE
       xValue = m_Value
    End Property

    Public Property Let xValue(ByVal vNewValue As OLE_OPTEXCLUSIVE)
       m_Value = vNewValue
       Debug.Print Ambient.DisplayName & vNewValue
       PropertyChanged "xValue"
    End Property
```

19.1.5 Aufzählungstypen

Wenn Sie einer Steuerelementeigenschaft einen Aufzählungstyp zuweisen, macht Visual Basic aus der Aufzählungsliste eine Dropdown-Liste möglicher Eigenschaftswerte für das Eigenschaftenfenster. Das sehen Sie im Steuerelement `ch19ctlA` bei der Eigenschaft EnumProp, die wie folgt implementiert ist:

```
Enum TestEnum
   dwFirstVal = 0
   dwSecondVal = 1
   dwThirdVal = 2
End Enum
Private m_Enum As TestEnum

' Aufzählungs-Eigenschaft
Public Property Get EnumProp() As TestEnum
   EnumProp = m_Enum
End Property

Public Property Let EnumProp(ByVal vNewValue As TestEnum)
   m_Enum = vNewValue
   PropertyChanged "EnumProp"
End Property
```

Das stellt eine recht seltsame Einschränkung für die Autoren von Steuerelementen dar, die unter Visual Basic arbeiten. Es ist sinnvoll, Aufzählungsnamen mit einem Präfix wie etwa »`dw`« zu versehen, um Konflikte mit anderen Aufzählungen zu vermeiden, die vielleicht einen ähnlichen Namen verwenden. Es ist aber auch wünschenswert, einen leicht verständlichen Namen in der Dropdown-Liste des Eigenschaftenfensters von Visual Basic zu haben. Visual Basic bietet keine Lösung für dieses Problem, aber in SpyWorks von Desaware gibt es einen Mechanismus, die Dropdown-Liste im Eigenschaftenfenster von Visual Basic anzupassen, so daß dort nicht die Standard-Aufzählungsliste verwendet wird. Außerdem ist es damit möglich, die im Eigenschaftenfenster angezeigten Werte zu überschreiben.

Aufzählungskonflikte können zu undurchschaubaren Fehlern führen, wenn Sie zwei verschiedene Komponenten einsetzen, die denselben Aufzählungsnamen für zwei unterschiedliche Werte verwenden. Solange Sie sich im selben Projekt befinden, warnt Visual Basic Sie, falls mehrdeutige Namen auftreten, wenn Sie versuchen, die Applikation auszuführen oder zu kompilieren. Andernfalls verwendet Visual Basic einfach den ersten Namen in der Verweisreihenfolge (die Verweisreihenfolge wurde in Kapitel 9 bereits beschrieben). Widerstehen Sie der Versuchung, Aufzählungsnamen wie »Erster«, Farbnamen oder Datumswerte zu verwenden, die mit ziemlicher Sicherheit nicht eindeutig sind.

Beachten Sie, daß Visual Basic keine Bereichsprüfung für die Werte ausführt, die einer Prozedur für Aufzählungseigenschaften übergeben werden. Sie sollten sicherstellen, daß der Prozedur `Property Let` korrekte Werte übergeben werden, und andernfalls einen Fehler aufwerfen.

19.1.6 Bilder und Schriften

Bilder und Schriften sind in Standard-OLE-Objekte eingekapselt, die die Schnittstellen `IPictureDisp` und `IFontDisp` unterstützen. Visual Basic erkennt Objekte dieser Typen und läßt ihnen eine besondere Behandlung zukommen. Das Eigenschaftenfenster von Visual Basic zeigt einen Überblick, der das Objekt beschreibt, sowie eine Schaltfläche, die beim Anklicken ein Standarddialogfeld oder eine Eigenschaftsseite anzeigt, mit deren Hilfe das Objekt bearbeitet wird.

Schrifteigenschaften müssen besonders behandelt werden. Statt direkt mit dem `UserControl`-Schriftobjekt zu arbeiten, sollten Sie eine interne Schriftvariable des Typs `StdFont` erzeugen:

```
Private WithEvents m_Font1 As StdFont
```

Ein `StdFont`-Objekt ist eine Automatisierungskomponente, die Schriftinformationen enthält und die `IFontDisp`-Schnittstelle bereitstellt, so daß Sie mit der Schrift arbeiten können. Aus unserer Sicht ist noch viel wichtiger, daß dieses Objekt Ereignisse aufwerfen kann, sobald sich Eigenschaften der Schrift ändern. Das `m_Font1`-Objekt hat ein `FontChanged`-Ereignis, das den folgenden Code nutzen könnte, um die Schrift des `UserControl`-Objekts zu aktualisieren:

```
Private Sub m_Font1_FontChanged(ByVal PropertyName As String)
   Set UserControl.Font = m_Font1
End Sub
```

Das UserControl wird außerdem im `Initialize`-Ereignis so initialisiert, daß es auf das interne `m_Font1`-Objekt verweist:

```
Set m_Font1 = New StdFont
Set UserControl.Font = m_Font1
```

Sowohl das `UserControl` als auch die `m_Font1`-Variable verweisen auf das `StdFont`-Objekt, das während des `Initialize`-Ereignisses erzeugt wird. Wenn Sie die Zuweisung `UserControl.Font = m_Font1` im `FontChange`-Ereignis verwenden, ändern Sie nicht wirklich den Verweis, aber das `UserControl` interpretiert die Eigenschaftszuweisung nichtsdestotrotz als Benachrichtigung, seine Schriftinformation zu aktualisieren.

Was ist, wenn Ihr Objekt die Umgebungsschrift verwenden soll? Sie können `m_Font1` während des `Initialize`-Ereignisses oder des InitProperties-Ereignisses mit den Werten der Umgebungsschrift initialisieren. Falls Sie die Zuweisung während `InitProperties` vornehmen, verwendet die Schrift standardmäßig alle Eigenschaften der Umgebungsschrift, wenn das Steuerelement auf einem Formular angelegt wird. Falls Sie sie während des `Initialize`-Ereignisses vornehmen, beginnt das Steuerelement immer mit den Werten der Umgebungsschrift, es sei denn, Sie bearbeiten sie mit Hilfe von `ReadProperties` und `WriteProperties`. In diesem Beispiel wird die Schrift während des `InitProperties`-Ereignisses mit der Umgebungsschrift initialisiert:

```
Private Sub UserControl_InitProperties()
   ' Mit der Umgebungsschrift initialisieren
   Set Font = Ambient.Font
End Sub
```

Auf den ersten Blick scheint es, als würden wir die Variable `m_Font1` neu zuweisen, aber das stimmt nicht. Wir verwenden die Prozedur `Property Set`, die die Werte von der Schrift in die Variable `m_Font1` kopiert:

```
Public Property Set Font1(ByVal vNewValue As Font)
   With m_Font1
      .Bold = vNewValue.Bold
      .Charset = vNewValue.Charset
      .Italic = vNewValue.Italic
      .Name = vNewValue.Name
      .Size = vNewValue.Size
      .Strikethrough = vNewValue.Strikethrough
      .Underline = vNewValue.Underline
      .Weight = vNewValue.Weight
   End With
   PropertyChanged "Font1"
End Property

Public Property Get Font1() As Font
   Set Font1 = m_Font1
End Property
```

Die Funktion `Property Get` gibt einen Verweis auf das `m_Font1`-Objekt zurück, was dem Container ermöglicht, die internen Daten des Objekts zu verändern, indem er die verschiedenen Objektmethoden aufruft. Woher weiß das Steuerelement, daß der Container eine seiner Schrifteigenschaften geändert hat? Hier kommt das `FontChanged`-Ereignis ins Spiel.

Stellen Sie sicher, daß die Anweisung `Property Get` für eine Schrift immer ein gültiges `Font`-Objekt zurückgibt. Wenn sie nichts zurückgibt, kann der Container die Schrift möglicherweise nicht korrekt setzen. Eine Möglichkeit dazu ist, die private Schriftvariable während des `InitProperties`-Ereignisses von `UserControl` auf die Umgebungsschrift zu setzen. Die Anweisung `Property Get` für ein Bild kann »nichts« zurückgeben, um anzuzeigen, daß kein Bild angegeben wurde.

Sie können Ihre eigenen Objekte erzeugen, die ähnlich wie `Picture`- oder `Font`-Objekte behandelt werden, aber es gibt einige Besonderheiten zu beachten. Dieses Thema wird später im Kapitel noch behandelt.

Die `Font`-Eigenschaft bietet eine interessante Möglichkeit. Sie können ein einziges Font-Objekt implementieren, oder separate Eigenschaften für `FontName`, `FontSize`, `FontBold`, `FontItalic` und `FontStrikethru`. Sie können sogar beides implementieren, wie Sie im Banner-Projekt in Kapitel 21 noch sehen werden.

Unabhängig davon, welche Eigenschaften Sie bereitstellen, haben Sie auch die Möglichkeit, das `Font`-Objekt mit einem einzigen Aufruf von `PropBag.WriteProperties` festzuschreiben, oder mit separaten Aufrufen für jede einzelne Schrifteigenschaft.

Auf den ersten Blick scheint die Verwendung eines einzigen `Font`-Objekts sowohl als Eigenschaft als auch für die Persistenz am sinnvollsten zu sein, aber das stimmt nicht immer. Wenn Sie möchten, daß Ihr Steuerelement auf Web-Seiten verwendet wird, könnten Sie dem einzelnen `Font`-Objekt die Bereitstellung von einzelnen `Font`-Eigenschaften vorziehen, weil einzelne `Font`-Eigenschaften auf der Web-Seite in einer vom Menschen lesbaren Form erscheinen. Wenn Sie das `Font`-Objekt festschreiben, erhalten Sie in der Regel ein Unterobjekt auf der Web-Seite, vorausgesetzt, Ihr Web-Entwicklungswerkzeug weiß überhaupt, wie das Objekt verarbeitet wird.

Diese Technik, Schrifteigenschaften zu verarbeiten, stellt übrigens eine wesentliche Änderung gegenüber Visual Basic 5 dar. Ich empfehle Ihnen, die neue Implementierung zu verwenden.

19.2 Eigenschaftsprozeduren

Für die Definition von Eigenschaften für ActiveX-Steuerelemente verwenden Sie immer Eigenschaftsprozeduren. Wie Sie sie definieren und was Sie darin tun, ist wichtiger Teil jedes Steuerelements.

19.2.1 ByVal in Property Let-Funktionen

`Property Let`-Prozeduren können größtenteils mit oder ohne ein `ByVal` definiert werden. Sie erzielen einen Performancegewinn für `String`-Variablen, wenn Sie den Parameter als Referenz übergeben. Um jedoch eine maximale Kompatibilität für unterschiedliche Container zu erzielen, sollten Parameter immer als Wert übergeben werden. Einige Container können nämlich bei der Referenzübergabe nicht alle Variablentypen korrekt verarbeiten. Beispiel: Visual Basic 4.0 kann als Referenz übergebene Boolean-Variablen nicht korrekt verarbeiten.

19.2.2 Fehler aufwerfen

Kapitel 9 hat die Konzepte für die OLE-Fehlerverarbeitung vorgestellt, wobei jeder 32-Bit-Fehlercode in drei Komponenten zerlegt wird: den Ergebniscode, den Subsystem-Code und die Fehlernummer. Bevor Sie weiterlesen, sollten Sie schnell den Abschnitt über die OLE-Fehlerverarbeitung noch einmal durchlesen.

Wie Sie gesehen haben, verwendet die Konstante `vbObjectError` einen Subsystem-Wert von 4, es handelt sich also um einen Schnittstellenfehler, wobei die ersten 512 Fehlernummern von Visual Basic vordefiniert sind. Bei der Arbeit mit ActiveX-Steuerelementen werfen Sie möglicherweise selbst gelegentlich

vbObjectError-Werte auf, wenn es sich um benutzerdefinierte Fehler handelt. Aber OLE definiert den Subsystem-Code &H0A, der insbesondere für ActiveX-Steuerelemente vorgesehen ist.

Seltsamerweise ist dies derselbe Subsystem-Code, den Visual Basic scheinbar intern für auffangbare Fehler verwendet. Beispielsweise ist der Fehler *Unzulässiger Eigenschaftswert* der auffangbare Fehler #380. Wenn Sie die folgende Zeile ausführen, sehen Sie die Fehlermeldung Unzulässiger Eigenschaftswert:

```
Err.Raise 380
```

Wenn Sie den Subsystem-Code jedoch explizit wie folgt setzen, sehen Sie denselben Fehler, und das Err-Objekt gibt die Fehlernummer 380 zurück:

```
Err.Raise &H800A0000 Or 380
```

Wenn Sie die Liste der auffangbaren Fehler in der Online-Hilfe von Visual Basic betrachten, sehen Sie, daß viele davon gerade für ActiveX-Steuerelemente praktisch sind. Sollten Sie sie in Ihrem Steuerelement verwenden? Natürlich!

Sie sehen, die für die Subsystem-Codes des ActiveX-Steuerelements definierten Fehlerwerte sind nicht nur Standard für Visual Basic; sie sind Standard für alle ActiveX-Steuerelemente. Der Fehlercode 380 wird von jedem Steuerelement aufgeworfen, wenn Sie versuchen, einen ungültigen Eigenschaftswert zu setzen. Das bedeutet, Ihr Visual-Basic-Programm kann alle Fehler von ActiveX-Steuerelementen verarbeiten, unabhängig davon, wer das Steuerelement entwickelt hat, und in welcher Sprache es geschrieben wurde. Es bedeutet außerdem, daß Sie diese Fehlercodes in Ihren Steuerelementen verwenden können (und sollten), und daß sie von allen ActiveX-Containern korrekt unterstützt werden.

Das Projekt ch19Tst2 soll Ihnen helfen, mit den von ActiveX-Steuerelementen aufgeworfenen Fehlern sowohl zur Entwurfszeit als auch zur Laufzeit zu experimentieren. Abbildung 19.1 zeigt das Steuerelement ch19chtl2A, das in diesem Projekt verwendet wird.

Das Steuerelement enthält ein Standard-Textfeld, in das Sie die Nummer des Fehlers eingeben, der aufgeworfen werden soll. Die Einstellungen der Kontrollkästchen GET und SET bestimmen, ob der Fehler aufgeworfen werden soll, wenn die Eigenschaft gelesen wird, oder wenn sie gesetzt wird. Der Code für das Steuerelement ist ganz einfach, wie Sie hier sehen:

```
' Guide to the Perplexed
    ' Chapter 19 - ch19chtl2a
    ' Copyright (c) 1997-1998 by Desaware Inc. All Rights Reserved
    Option Explicit

    Private m_Facility As Long

    Private Const vbCtlError = &HA0000 Or &H80000000
```

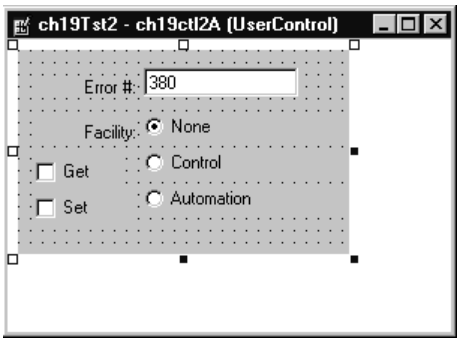

Abb. 19.1: Das Steuerelement ch19chtl2A

```
    Private Sub optFacility_Click(Index As Integer)
       Select Case Index
          Case 0
             m_Facility = 0
          Case 1
             m_Facility = vbCtlError
          Case 2
             m_Facility = vbObjectError
       End Select
    End Sub

    ' Die Kontrollkästchen Get und Let bestimmen, wann der Fehler aus-
gelöst wird
    Public Property Get TriggerError() As String
       Dim errcode&
       errcode = Val(txtErrNum.Text) Or m_Facility
       Debug.Print Hex$(errcode)
       If chkGet.Value Then Err.Raise errcode
    End Property

    Public Property Let TriggerError(ByVal vNewValue As String)
       Dim errcode&
       errcode = Val(txtErrNum.Text) Or m_Facility
       Debug.Print Hex$(errcode)
       If chkLet.Value Then Err.Raise errcode
    End Property
```

Das Testformular, `ch19frm.frm`, enthält eine einzige Instanz des Steuerelements sowie zwei Schaltflächen. Die Schaltfläche GET bewirkt, daß die `TriggerError`-Eigenschaft des Steuerelements gelesen wird. Die Schaltfläche SET bewirkt, daß sie auf einen Zufallswert gesetzt wird.

Hier folgt eine Frage, die Sie schnell beantworten können. Es sollte klar sein, wie das Steuerelement zur Laufzeit getestet werden kann – alle Standardelemente auf dem Steuerelement `ch19tl2a` sind aktiv, Sie ändern also einfach die Fehlernummer im Textfeld, wählen die gewünschten Optionen aus und klicken auf die Schaltflächen GET oder SET. Aber wie führen Sie denselben Text zur Entwurfszeit des Containers aus?

Wenn Sie die Eigenschaftseinstellungen für das Steuerelement betrachten (die Eigenschaften des `UserControl`-Objekts für das Steuerelement), sehen Sie, daß die `EditAtDesignTime`-Eigenschaft auf `True` gesetzt ist. Nachdem Sie den Designer geschlossen und das Steuerelement auf das Formular gezeichnet haben, rufen Sie im Kontextmenü (das ist das Menü, das erscheint, wenn Sie mit der rechten Maustaste auf das Steuerelement klicken) den Befehl BEARBEITEN auf. Damit wird das Steuerelement aktiv, und Sie können das Textfeld bearbeiten und die anderen Optionen für das Steuerelement setzen. Klicken Sie auf das Formular. Das Steuerelement verläßt den Bearbeitungsmodus. Wenn Sie erneut auf das Steuerelement klicken, versucht Visual Basic, die `TriggerError`-Eigenschaft zu lesen, um das Eigenschaftenfenster zu laden, so daß Sie das Lesen der Eigenschaft testen können. Um das Setzen der Eigenschaft zu testen, bearbeiten Sie den `TriggerError`-String im Eigenschaftenfenster.

Warum beschränkt sich dieses Beispielprogramm auf den Test von Fehlern, die beim Zugriff auf Eigenschaften aufgeworfen werden? Ganz allgemein kann man sagen, der einzige Zeitpunkt, zu dem Ihr Steuerelement Fehler aufwerfen sollte, ist beim Zugriff auf Eigenschaften oder bei Aufrufen von Methoden. Sie sollten vermeiden, Fehler aufzuwerfen, während interne Ereignisse verarbeitet werden. Der Container ist nämlich vielleicht nicht darauf ausgelegt, Fehler zu verarbeiten, die zu beliebigen Zeitpunkten aufgeworfen werden. Wenn ein Fehler zu anderen Zeitpunkten auftritt, sollten Sie ihn auffangen und ihn dann aufwerfen, wenn in einer Methode oder Eigenschaft wieder ein Zugriff auf das Steuerelement erfolgt.

Häufig verwendete Fehlercodes. Eine vollständige Liste der auffangbaren Fehler finden Sie in der Online-Hilfe von Visual Basic (suchen Sie im Index nach »Fehler, auffangbare«). Die meisten dieser Fehler sind selbsterklärend. Tabelle 19.1 listet die von Steuerelement-Autoren am häufigsten verwendeten Fehler auf.

Fehlercode	Beschreibung
7	Nicht genügend Speicher – Wird normalerweise aufgeworfen, wenn in Ihrem Steuerelement eine Speicherreservierung fehlschlägt.
380	Unzulässiger Eigenschaftswert – Werfen Sie diesen Fehler auf, wenn der Benutzer versucht, die Eigenschaft auf einen unzulässigen Wert zu setzen.

Tab. 19.1: Von Steuerelement-Autoren häufig verwendete Fehler

Fehlercode	Beschreibung
381	Ungültiger Index des Eigenschaftsfelds – Werfen Sie diesen Fehler auf, wenn für eine parametrisierte Eigenschaft (Eigenschaftsfeld) ein ungültiger Indexwert angegeben wird.
382	Set wird zur Laufzeit nicht unterstützt – Werfen Sie diesen Fehler auf, wenn versucht wird, eine Eigenschaft zu setzen, von der Sie wollen, daß eine Eigenschaft nur zur Entwurfszeit gesetzt werden soll. Werfen Sie diesen Fehler nur auf, wenn die Umgebungseigenschaft `UserMode` auf `True` gesetzt ist.
383	Set wird nicht unterstützt (schreibgeschützte Eigenschaft) – Werfen Sie diesen Fehler auf, wenn versucht wird, eine Eigenschaft zu setzen, die nur gelesen werden soll.
387	Set nicht zulässig – Werfen Sie diesen Fehler auf, wenn versucht wird, eine Eigenschaft zu setzen, die vorübergehend als nur lesbar konfiguriert ist.
393	Get wird zur Laufzeit nicht unterstützt – Werfen Sie diesen Fehler auf, wenn versucht wird, eine Eigenschaft zu lesen, von der Sie wollen, daß sie zur Laufzeit nur geschrieben werden kann. Dieser Eigenschaftstyp wird in der Regel verwendet, um eine Aktion im Steuerelement auszulösen, und sollte besser als Methode implementiert werden. Werfen Sie diesen Fehler nur auf, wenn die Umgebungseigenschaft `UserMode` auf `True` gesetzt ist.
394	Get wird nicht unterstützt (schreibgeschützte Eigenschaft) – Werfen Sie diesen Fehler auf, wenn versucht wird, eine Eigenschaft zu setzen, von der Sie wollen, daß sie nur geschrieben werden kann. Dieser Eigenschaftstyp sollte besser als Methode implementiert werden.

Tab. 19.1: Von Steuerelement-Autoren häufig verwendete Fehler

Die Konzepte der Fehlerverarbeitung, die in Kapitel 9 für ActiveX-Komponenten beschrieben wurden, gelten auch für ActiveX-Steuerelemente. Wenn eine Komponente, die Ihr Steuerelement verwendet (ein Standardelement oder ein Objekt, wie beispielsweise das `UserControl`-Objekt für das Steuerelement), einen Fehler aufwirft, sollten Sie diesen innerhalb Ihres Steuerelements verarbeiten, und ihn dann für den Container Ihres Steuerelements aufwerfen, falls das sinnvoll ist.

Um den Entwicklern, die Ihr Steuerelement einsetzen, das Leben zu erleichtern, sollten Sie die Fehler dokumentieren, die Ihr Steuerelement aufwerfen kann. Das ist insbesondere für benutzerdefinierte Fehler wichtig.

19.2.3 Laufzeit und Entwurfszeit

Als Steuerelement-Autor legen Sie fest, wann es erlaubt sein soll, Eigenschaften zu lesen oder zu schreiben. Es gibt 16 mögliche Permutationen für die Lese/Schreib-Berechtigungen, wie in Tabelle 19.2 gezeigt. »Lesen« gibt an, wann die

Eigenschaft gelesen werden kann; »Schreiben« gibt an, daß sie gesetzt werden kann. Das Verhalten des VB-Eigenschaftsfensters bezieht sich auf numerische oder String-Eigenschaftstypen.

Lesen Laufzeit	Lesen Entwurfszeit	Schreiben Laufzeit	Schreiben Entwurfszeit	Beschreibung
Ja	Ja	Ja	Ja	Jederzeit voller Zugriff – eine der gebräuchlichsten Konfigurationen.
Ja	Ja	Ja	Nein	Siehe Fußnote 1.
Ja	Ja	Nein	Ja	Nur Schreiben zur Entwurfszeit. Diese Konfiguration wird häufig für Eigenschaften verwendet, die nur zur Entwurfszeit gesetzt werden können.
Ja	Ja	Nein	Nein	Welchen Nutzen hat eine Eigenschaft, die nicht gesetzt werden kann? Sie kann genutzt werden, um Informationen aus dem Steuerelement zu ermitteln. Siehe Fußnote 1.
Ja	Nein	Ja	Ja	Siehe Fußnote 2.
Ja	Nein	Ja	Nein	Zugriff nur zur Laufzeit. Wird häufig für Eigenschaften verwendet, die nur zur Laufzeit zur Verfügung stehen.
Ja	Nein	Nein	Ja	Siehe Fußnote 2.
Ja	Nein	Nein	Nein	Zur Laufzeit nur lesbar. Wird häufig verwendet, um Informationen aus einem Steuerelement zu ermitteln. Kann nicht persistent gemacht werden.

Tab. 19.2: Eigenschaftsberechtigungen; Laufzeit UserMode = True, Entwurfszeit UserMode = False

Lesen Laufzeit	Lesen Entwurfszeit	Schreiben Laufzeit	Schreiben Entwurfszeit	Beschreibung
Nein	Ja	Ja	Ja	Kann zur Entwurfszeit gesetzt werden, aber zur Laufzeit nur geschrieben. Eine sehr ungebräuchliche Konfiguration, aber sicher in der Verwendung.
Nein	Ja	Ja	Nein	Siehe Fußnote 1.
Nein	Nein	Ja	Ja	Könnte verwendet werden, um eine Operation in einem Steuerelement auszulösen. Siehe Fußnote 2.
Nein	Nein	Ja	Nein	Wird manchmal genutzt, um eine Operation in einem Steuerelement auszulösen. Dieser Ansatz war in der VBX-Technologie sehr gebräuchlich, wo keine benutzerdefinierten Methoden erlaubt waren. Versuchen Sie, statt dessen eine Methode zu verwenden, um diese Aufgabe zu bewerkstelligen.
Nein	Nein	Nein	Ja	Siehe Fußnote 2.
Nein	Nein	Nein	Nein	Wenn Sie einen Verwendungszweck für diese Kombination finden, dann lassen Sie es mich bitte wissen!

Tab. 19.2: Eigenschaftsberechtigungen; Laufzeit UserMode = True, Entwurfszeit UserMode = False

Fußnote 1: Die Eigenschaft wird im VB-Eigenschaftenfenster angezeigt, kann aber nicht gesetzt werden. Diese Konfiguration ist möglicherweise sinnvoll für die Anzeige einer Eigenschaft im VB-Eigenschaftenfenster, aber sie ist nicht-intuitiv und führt möglicherweise zu Beschwerden von Benutzern Ihres Steuerelements, die sich fragen, warum jeder Versuch, die Eigenschaft zu bearbeiten, zu einem Fehler führt (oder einfach nicht funktioniert). Wenn Sie eine Eigenschaft im VB-Eigenschaftenfenster anzeigen möchten, die nicht gesetzt werden kann, sollten Sie wenigstens vermeiden, einen Fehler dafür aufzuwerfen.

Fußnote 2: Eine Eigenschaft, die zur Entwurfszeit nicht gelesen werden kann, wird im VB-Eigenschaftenfenster nicht angezeigt. Ein solches Szenario ist möglich, wenn Sie einen Mechanismus besitzen, mit dem Sie den Eigenschaftswert intern oder über eine Eigenschaftsseite setzen.

Sie sind dafür verantwortlich, den Zugriff auf Ihre Eigenschaften gegebenenfalls zu verhindern. Das ist auf zweierlei Arten möglich. Sie können einen Fehler aufwerfen oder die Operation einfach ignorieren.

Die Fehler 382, 383, 392 und 394, die Sie in Tabelle 19.1 gesehen haben, können beim Versuch aufgeworfen werden, auf die Eigenschaft zuzugreifen. Um beispielsweise eine Eigenschaft zur Laufzeit nur lesbar zu machen, könnten Sie den folgenden Code verwenden:

```
Public Property Let MyProp(ByVal vNewValue As Variant)
   If Ambient.UserMode Then
      Err.Raise 382
   End If
End Property
```

Jeder Versuch, die Eigenschaft zur Laufzeit zu setzen, wirft den Fehler »Set wird zur Laufzeit nicht unterstützt« auf.

Bei ActiveX-Codekomponenten ist es gebräuchlich, nur lesbare Eigenschaften zu erzeugen, indem man einfach die `Property Let`-Prozedur wegläßt (und umgekehrt für Eigenschaften, die nur geschrieben werden können). Sie sollten diesen Ansatz nicht für ActiveX-Steuerelemente verfolgen. Er funktioniert vielleicht, aber die Fehlermeldungen für den Versuch, die unerlaubte Operation auszuführen (Meldungen wie »Objekt erforderlich«), sind für den Endbenutzer nicht besonders aussagekräftig. Besser implementieren Sie die Prozeduren und werfen den geeigneten Fehler auf.

Wenn Sie möchten, daß eine Eigenschaft im Visual-Basic-Eigenschaftenfenster erscheint, aber zur Entwurfszeit nur gelesen wird, sollten Sie keinen Fehler aufwerfen, wenn der Benutzer versucht, die Eigenschaft zu setzen. Ignorieren Sie einfach den Versuch, die Eigenschaft zu setzen, wie im folgenden Code gezeigt:

```
Public Property Let MyProp(ByVal vNewValue As Variant)
   If Not Ambient.UserMode Then   ' Design time
     Exit Property
   End If
   ' Hier erfolgt die Operation zum Setzen der Eigenschaft zur Laufzeit
End Property
```

19.3 Steuerelement-Prozedurattribute

Das Dialogfeld Extras, Prozedurattribute bietet mehrere Einstellungen, die für ActiveX-Steuerelemente extrem wichtig sind. Kapitel 10 hat die Attribute vorgestellt, die auf jede ActiveX-Komponente angewendet werden können. Mehrere zusätzliche Attribute beziehen sich speziell auf ActiveX-Steuerelemente. Die Optionen für die Datenbindung werden später in diesem Kapitel erklärt.

19.3.1 Prozedur-ID

Sie haben bereits gesehen, daß die Prozedur-ID der Dispatch-ID (DispID) einer ActiveX-Automatisierungsschnittstelle entspricht. Sie wissen, daß OLE zahlreiche Standard-Prozedur-IDs definiert, die Sie größtenteils im Dropdown-Kombinationsfeld im Dialogfeld Prozedurattribute auswählen können.

Beachten Sie, daß die Prozedur-ID keinen Einfluß auf die eigentliche Eigenschaft hat, sondern nur darauf, wie der Container mit der Eigenschaft umgeht. Die Eigenschaft oder Methode muß nicht denselben Namen wie die Prozedur-ID haben. Der Container geht nicht nach dem Eigenschaftsnamen, sondern nur nach der Prozedur-ID.

Tabelle 19.3 listet die Prozedur-IDs auf, die Einfluß auf das Verhalten von Visual Basic haben. Sie sollten den Standardeigenschaften, die Sie in Ihrem Steuerelement implementieren, immer die Standard-Prozedur-ID zuweisen. Damit stellen Sie sicher, daß Ihr Steuerelement auch auf anderen Containern korrekt funktioniert, die eine spezielle Verarbeitung für Standardeigenschaften bereitstellen.

IDs	Beschreibungen
(Keine)	Nicht-Standard-Dispatch-ID, die der Eigenschaft zugeordnet wurde.
(Vorgabe)	Die Eigenschaft, auf die zugegriffen wird, wenn Sie auf das Objekt verweisen, ohne daß Sie einen Eigenschaftsnamen angeben. Wählen Sie die Standardeigenschaft sorgfältig aus (oder vermeiden Sie ihre Verwendung ganz), um die Entwickler nicht zu verwirren, die Ihr Steuerelement einsetzen.
AboutBox	Wenn einer Methode diese Prozedur-ID hinzugefügt ist, ruft der Container die Methode auf, um ein Info-Feld anzuzeigen. Visual Basic fügt dem VB-Eigenschaftenfenster einen Infofeld-Eintrag hinzu.

Tab. 19.3: Standard-Prozedur-IDs von Visual Basic

IDs	Beschreibungen
Caption oder Text	Wenn eine Eigenschaft eine dieser Prozedur-IDs erhält, aktualisiert Visual Basic sie automatisch, wenn Tastencodes im VB-Eigenschaftenfenster erkannt werden. Das ist das Standardverhalten von Text- und Caption-Eigenschaften für viele Steuerelemente. Beachten Sie, daß Sie diese Prozedur-IDS für alle Eigenschaften verwenden können, auch wenn sie nichts mit einer typischen Text- oder Caption-Eigenschaft zu tun haben.
Enabled	Durch Einstellung dieser Prozedur-ID für eine Eigenschaft ist es Visual Basic möglich, das Standard-Enabled-Verhalten für Ihr Steuerelement zu implementieren, wie in Kapitel 18 beschrieben.

Tab. 19.3: Standard-Prozedur-IDs von Visual Basic

19.3.2 Eigenschaftenkatalog-Seite

ActiveX-Steuerelemente können Eigenschaftsseiten implementieren, um das Verhalten und die Eigenschaften Ihres Steuerelements zur Entwurfszeit manipulierbar zu machen. Dieses Kombinationsfeld enthält eine Liste aller aktuell definierter Eigenschaftsseiten. Wenn Sie einer Eigenschaft im VB-Eigenschaftenfenster eine Eigenschaftsseite zuweisen, zeigt Visual Basic eine Schaltfläche an, über die Sie zu der betreffenden Eigenschaftsseite gelangen.

Die Zuweisung einer Eigenschaftsseite für eine Eigenschaft auf diese Weise überschreibt das existierende Verhalten des Eintrags für diese Eigenschaft im Eigenschaftenfenster. Eine Aufzählungseigenschaft beispielsweise hat in der Regel ein Dropdown-Kombinationsfeld im Eigenschaftenfenster, das die möglichen Werte für die Eigenschaft auflistet. Wenn Sie der Eigenschaft eine Eigenschaftsseite zuweisen, erscheint statt dessen die Schaltfläche für den Eigenschaftsseitendialog.

Eigenschaftsseiten werden in diesem und in Kapitel 20 noch genauer beschrieben.

19.3.3 Kategorie

Das Kombinationsfeld Kategorie ermöglicht Ihnen, einer Eigenschaft eine Kategorie zuzuordnen. Damit ist es möglich, Eigenschaften so zu gruppieren, wie sie zusammengehören, und wie sie angezeigt werden sollen, wenn Sie im VB-Eigenschaftenfenster auf die Registerkarte NACH KATEGORIE klicken. Das Kombinationsfeld enthält eine Liste der Standardkategorien, Sie können aber auch eine eigene hinzufügen, indem Sie sie in das Bearbeitungsfeld des Kombinationsfelds eingeben.

Diese Einstellung dient nur der Bequemlichkeit der Entwickler, die Ihr Steuerelement einsetzen, und hat keinerlei Einfluß auf die Funktionalität des Steuerelements.

19.3.4 Im Eigenschaftenkatalog nicht anzeigen

Diese Option verhindert, daß eine Eigenschaft im Eigenschaftenfenster von Visual Basic erscheint. Andere Container sollten diesem Verhalten ebenfalls folgen. Die Eigenschaft kann im Objektkatalog von Visual Basic weiterhin angesehen werden und wird in der Typbibliothek für das Steuerelement nicht als verborgen markiert. Verwenden Sie diese Option für alle Eigenschaften, auf die nur zur Laufzeit ein Zugriff möglich sein soll.

19.3.5 Voreinstellung für die Benutzeroberfläche

Es ist möglich, eine Eigenschaft und ein Ereignis als Standard für die Benutzeroberfläche voreinzustellen. Diese Voreinstellung wird im VB-Eigenschaftenfenster angezeigt, wenn der Entwickler dieses aufruft. Die Voreinstellung für das Ereignis bewirkt, daß dieses ausgewählt wird, wenn der Entwickler auf das Codefenster doppelklickt oder Ihre Komponente auswählt, um den Ereignissen für Ihr Steuerelement Code hinzuzufügen. Diese Einstellung dient nur der Bequemlichkeit für die Entwickler, die Ihr Steuerelement nutzen, und hat keinen Einfluß auf die Funktionalität des Steuerelements.

19.4 Benutzerdefinierte Objekte

Sie können Ihre eigenen Objekte erzeugen, die ähnlich wie Picture- und Font-Objekte verarbeitet werden, wobei es jedoch einige Feinheiten zu beachten gibt. Überlegen Sie kurz, was Sie bereits über die Arbeit mit Eigenschaften und ihr Verhalten im VB-Eigenschaftenfenster wissen:

- Eine `Property Get`-Funktion muß einen Wert zurückgeben, der als String angezeigt werden kann, damit die Eigenschaft im VB-Eigenschaftenfenster erscheint.
- Eine `Property Get`-Funktion muß einen Objektverweis zurückgeben, wenn Sie möchten, daß es sich um eine echte `Object`-Eigenschaft handelt.
- Eine `Object`-Eigenschaft wird immer mit Hilfe einer separaten Dialog- oder Eigenschaftsseite bearbeitet. Sie sollten also eine Dialogschaltfläche im VB-Eigenschaftenfenster für die `Object`-Eigenschaft anzeigen.
- Eine `Property Set`-Prozedur muß erlauben, einen Objektverweis zu setzen, wenn Sie möchten, daß es sich um eine echte `Object`-Eigenschaft handelt.

Offensichtlich erfüllen sowohl die `Picture`- als auch die `Font`-Objekte alle diese Anforderungen. Auf den ersten Blick scheinen diese etwas widersprüchlich zu

sein. Wie kann die Property Get-Funktion sowohl einen String als auch ein Objekt zurückgeben? Oh je – ich glaube, wir müssen doch Variants verwenden.

19.4.1 Die Bruch-Objekte clsMyObject/clsMyObject2

clsMyObject und clsMyObject2 im Projekt ch19ctls sind zwei extrem einfache Objekte, die zwei Zahlen speichern: Einen Zähler und einen Nenner für einen Bruch. Das ist gar nicht so abwegig, wie Sie vielleicht denken, weil es viele Brüche gibt, die durch Fließkommazahlen nicht exakt dargestellt werden können.

Zuerst betrachten wir das Objekt clsMyObject. Das Steuerelement ch1OctlA hat die Eigenschaft MyObject1, die Ihnen ermöglicht, ein Objekt dieses Typs zu setzen und zu laden. Der Eigenschaftswert kann zur Entwurfszeit über eine Eigenschaftsseite gesetzt und dann festgeschrieben (mit dem Projekt gespeichert) werden. Ich weiß, daß die Eigenschaftsseiten noch nicht besprochen wurden. Eine kurze Einführung folgt später in diesem Abschnitt, und im nächsten Kapitel finden Sie detaillierte Erklärungen.

Die Persistenzaspekte werden wir später betrachten. Hier wollen wir uns auf das Objekt konzentrieren.

Listing 19.1 zeigt den Code für das Objekt clsMyObject. Die Nenner- (Denominator) und Zähler- (Numerator) Eigenschaften sind ganz einfach. Sie sehen, daß der Nenner mit 1 initialisiert wird und nicht auf 0 gesetzt werden kann. Die Result-Eigenschaft (die nur gelesen werden kann) könnte einen Überlauffehler verursachen, aber es wird keine Fehlerüberprüfung ausgeführt, weil diese eben maximal denselben Überlauffehler verursachen würde.

```
' Guide to the Perplexed
' Beispiele für Kapitel 19
' Copyright (c) 1997-1998 by Desaware Inc. All Rights Reserved

Option Explicit

Private m_Numerator As Double
Private m_Denominator As Double

Public Property Get Numerator() As Double
   Numerator = m_Numerator
End Property

Public Property Let Numerator(ByVal vNewValue As Double)
   m_Numerator = vNewValue
End Property

Public Property Get Denominator() As Double
   Denominator = m_Denominator
End Property
```

```
Public Property Let Denominator(ByVal vNewValue As Double)
   If vNewValue = 0 Then Exit Property
   m_Denominator = vNewValue
End Property

Private Sub Class_Initialize()
   m_Denominator = 1
End Sub

Public Property Get Result() As Double
   Result = m_Numerator / m_Denominator
End Property

Public Property Get DisplayName() As String
   DisplayName = Str$(m_Numerator) & "/" & _
LTrim$(Str$(m_Denominator))
End Property
```

Listing. 19.1: Das Objekt clsMyObject

19.4.2 Die Eigenschaftsprozeduren

In diesem Beispiel ist die Variable `m_MyObject1`, die die aktuelle Einstellung der `MyObject1`-Eigenschaft enthält, auf einen zulässigen Wert gesetzt. Sie ist also wie folgt definiert und initialisiert:

```
Private m_MyObject1 As clsMyObject

Private Sub UserControl_Initialize()
   Set m_MyObject1 = New clsMyObject
End Sub
```

Bevor Sie den restlichen Code für die Implementierung ansehen, wollen wir betrachten, was nötig ist, um die zuvor beschriebenen Bedingungen zu erfüllen. Zur Entwurfszeit:

- Die `Property Get`-Anweisung muß einen deskriptiven String zurückgeben, der im VB-Eigenschaftenfenster angezeigt werden kann.
- Er muß alle Änderungen ignorieren, die der Benutzer im VB-Eigenschaftenfenster vornimmt.
- Er muß eine Schaltfläche realisieren, um eine Eigenschaftsseite anzuzeigen, die die Bearbeitung des Objekts erlaubt.
- Er muß eine Möglichkeit für die Eigenschaftsseite bieten, um das Objekt der Eigenschaft zu lesen und zu schreiben.

Zur Laufzeit darf die Eigenschaft sowohl für das Lesen als auch für das Schreiben nur Verweise auf Objekte vom Typ `clsMyObject` entgegennehmen.

Zunächst betrachten wir die Get-Seite der Gleichung. Die `Property Get`-Prozedur ist wie folgt definiert:

```
Public Property Get MyObject1() As Variant
   If Ambient.UserMode Then
      Set MyObject1 = m_MyObject1
   Else
      MyObject1 = m_MyObject1.DisplayName
   End If
End Property
```

Zur Laufzeit gibt die `Property Get`-Anweisung immer einen Verweis auf das Objekt zurück. Zur Entwurfszeit gibt sie immer einen deskriptiven String für das Objekt zurück. Dieser String erscheint im VB-Eigenschaftenfenster für die Eigenschaft.

Aber wie kann eine Eigenschaftsseite auf das Objekt der Eigenschaft zur Entwurfszeit zugreifen, wenn die `Property Get`-Anweisung einen String zurückgibt? Sie verwendet eine separate Prozedur dafür, wie hier gezeigt:

```
Friend Property Get InternalMyObject1() As clsMyObject
   Set InternalMyObject1 = m_MyObject1
End Property
```

Beachten Sie, daß diese Prozedur eine `Friend`-Funktion ist, und nur andere Komponenten desselben Projekts darauf Zugriff haben. Die Eigenschaftsseite kann sie ebenfalls benutzen, aber sie bleibt den Entwicklern, die Ihr Steuerelement einsetzen, verborgen.

Auf der Zuweisungsseite haben Sie es sowohl mit der `Let`- als auch mit der `Set`-Prozedur zu tun. Auf der Let-Seite können Sie einfach feststellen, ob der `Variant` ein Objekt des richtigen Typs enthält. Ist das der Fall, wird der internen Variablen der neue Wert zugewiesen. Andernfalls wird zur Laufzeit der Fehler Unzulässiger Eigenschaftstyp aufgeworfen. Zur Entwurfszeit werden Fehler einfach ignoriert; die Anzeige im VB-Eigenschaftenfenster bleibt unverändert. Sie brauchen sich nicht darum zu kümmern, ob unzulässige Wert von der Eigenschaftsseite gesetzt werden, weil Sie der Autor der Eigenschaftsseite sind und sicherstellen können, daß sie beim Setzen der Eigenschaft immer gültige Werte verwendet.

Eine separate `Property Set`-Prozedur übernimmt das direkte Setzen des Objekts mit Hilfe der `Set`-Anweisung anstelle der `Variant`-Zuweisung.

```
Public Property Let MyObject1(ByVal vNewValue As Variant)
   If TypeOf vNewValue Is clsMyObject Then
      Set m_MyObject1 = vNewValue
   Else
```

```
      If Ambient.UserMode Then
         Err.Raise 380
      End If
      ' Zur Entwurfszeit keinen Fehler aufwerfen
   End If
   PropertyChanged "MyObject1"
End Property

Public Property Set MyObject1(ByVal vNewValue As clsMyObject)
   Set m_MyObject1 = vNewValue
   PropertyChanged "MyObject1"
End Property
```

Mit diesem Code können Sie die Eigenschaft auf zweierlei Arten zuweisen, wie im folgenden Code aus dem Projekt `ch19tst1` gezeigt:

```
Private Sub Command1_Click()
   Dim myobj As clsMyObject
   Set myobj = ch19ctlA1.MyObject1
   myobj.Numerator = 2
   myobj.Denominator = 3
   ch19ctlA1.MyObject1 = myobj
   Debug.Print ch19ctlA1.MyObject1.DisplayName
   myobj.Numerator = 1
   myobj.Denominator = 3
   Set ch19ctlA1.MyObject1 = myobj
   Debug.Print ch19ctlA1.MyObject1.DisplayName
End Sub
```

Im ersten Fall verwenden Sie eine direkte Zuweisung ohne Set-Anweisung. Das funktioniert, weil das Objekt automatisch vorübergehend in einen `Variant` umgewandelt wird, der dann der `Property Let`-Prozedur übergeben wird. Die Prozedur erkennt, daß das Objekt den korrekten Typ hat, und nimmt die Zuweisung vor. Im zweiten Fall wird das Objekt direkt der `Property Set`-Prozedur übergeben. Dieser Ansatz ist viel effizienter.

Sie könnten die Möglichkeit verwerfen, daß in der `Property Let`-Anweisung `clsMyObject`-Objekte zugewiesen werden (Variablenzuweisung ohne das Set-Schlüsselwort), indem Sie einfach immer einen Fehler aufwerfen, wenn sie zur Laufzeit aufgerufen wird. Damit wären die Entwickler gezwungen, immer die Set-Befehlssyntax zu verwenden. Sie sollten immer einen Fehler aufwerfen, wenn Sie diesen Ansatz verwenden. Andernfalls können die Entwickler nicht erkennen, was falsch war, wenn sie versehentlich das Schlüsselwort Set weglassen (und das passiert ganz häufig).

Noch eine letzte Anmerkung zum obigen Code: Mit wie vielen Objekten haben Sie es letztlich zu tun? Wenn Sie sie sorgfältig durchgehen, sehen Sie, daß es sich nur um ein einziges Objekt handelt! Durch die Zuweisung der Variablen `myobj`

zur `MyObject1`-Eigenschaft des Steuerelements wird die Eigenschaft demselben Objekt zugewiesen, auf das sie bereits verweist. Sie sehen dieselben Einzelschrittergebnisse, als ob Sie die Zuweisungen vorgenommen hätten. Das ist für dieses Beispiel nicht wichtig, weil hier nur bewiesen werden sollte, daß die Zuweisung wirklich funktioniert.

19.4.3 Eigenschaftsseiten – Grundlagen

Wir werden die Eigenschaftsseiten im nächsten Kapitel genauer betrachten. Hier werden wir nur so viel erklären, wie wir brauchen, um ein benutzerdefiniertes Objekt zu implementieren. Die Eigenschaftsseite für das Objekt `clsMyObject` sehen Sie in Abbildung 19.2.

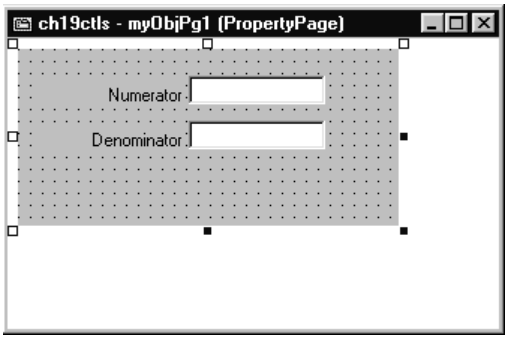

Abb. 19.2: Entwurfsansicht der clsMyObject-Eigenschaftsseite, myObjPg1.

Es gibt ein Textfeld für den Zähler und eines für den Nenner. Visual Basic fügt der Seite bei der Anzeige automatisch die Schaltflächen OK, ABBRECHEN und ÜBERNEHMEN hinzu.

Nachdem Sie die Eigenschaftsseite erzeugt haben, können Sie sie der Eigenschaft zuordnen. Dazu verwenden Sie das Dialogfeld EXTRAS/PROZEDURATTRIBUTE. Alle definierten Eigenschaftsseiten erscheinen im Kombinationsfeld EIGENSCHAFTENKATALOG-Seite.

Jede Eigenschaftsseite hat ein `PropertyPage`-Objekt, ganz ähnlich wie jedes VB-Steuerelement ein `UserControl`-Objekt hat. Das `PropertyPage`-Objekt hat die Eigenschaft `Changed`, die angibt, ob eine der Eigenschaften für die Seite geändert wurde. Sie wird immer gesetzt, wenn Sie eines der Textfelder ändern, das die Objekteigenschaften darstellt, wie im folgenden Code gezeigt:

```
Private Sub txtDenominator_Change()
   Changed = True
End Sub

Private Sub txtNumerator_Change()
```

```
    Changed = True
End Sub
```

Das `SelectionChanged`-Ereignis von `PropertyPage` tritt auf, wenn die Seite zum ersten Mal einem Objekt zugeordnet wird. Zu diesem Zeitpunkt verwendet das Programm die `InternalMyObject1`-Eigenschaft des Steuerelements, um das Objekt zu laden, das bearbeitet werden soll. Anschließend initialisiert es die Textfelder abhängig von diesem Objekt. Schließlich setzt es die `Changed`-Eigenschaft auf False zurück (weil sie auf `True` gesetzt wurde, als die Textfelder initialisiert wurden, und wir eigentlich noch gar keine Änderungen vorgenommen haben). Machen Sie sich hier noch keine Gedanken über das Feld `SelectedControls()`. Gehen Sie einfach davon aus, daß es eine Möglichkeit bietet, auf das zu bearbeitende Steuerelement zuzugreifen.

```
Private Sub PropertyPage_SelectionChanged()
    Dim myobject As ch19ctlA
    Dim refobject As clsMyObject
    ' Frühe Bindung, um Zugriff auf die Friend-Prozedur zu haben
    Set myobject = SelectedControls(0)
    Set refobject = myobject.InternalMyObject1
    txtNumerator.Text = refobject.Numerator
    txtDenominator.Text = refobject.Denominator
    ' Initialisierung abgeschlossen - die geänderten
    ' Werte sollen überschrieben werden
    Changed = False
End Sub
```

Wenn Sie diese Änderungen auf die Eigenschaft anwenden oder auf die OK-Schaltfläche der Seite klicken, wird das `ApplyChanges`-Ereignis von `PropertyPage` ausgelöst. Das Programm erzeugt jetzt ein neues Objekt, lädt die Eigenschaften des Objekts abhängig von den Textfeldern und weist sie dem Steuerelement über die `Property Set`-Anweisung des Steuerelements zu, wie der folgende Code zeigt:

```
Private Sub PropertyPage_ApplyChanges()
    Dim refobject As clsMyObject
    Set refobject = New clsMyObject
    refobject.Numerator = txtNumerator.Text
    refobject.Denominator = txtDenominator.Text
    Set SelectedControls(0).MyObject1 = refobject
End Sub
```

Es gibt noch andere Ansätze, die Sie für die Implementierung der `ApplyChanges`-Anweisung verwenden könnten, wie Sie im nächsten Kapitel sehen werden.

19.5 Persistenz

Einer der wichtigsten Unterschiede zwischen ActiveX-Codekomponenten und ActiveX-Steuerelementen ist, daß die Steuerelemente in der Lage sind, Informationen über ihren Status zu speichern, der von einem Entwickler zur Entwurfszeit gesetzt wird. Kapitel 17 hat die Ereignisse vorgestellt, während derer Sie die Eigenschaftsinformation speichern sollten. Nun wollen wir betrachten, wie das realisiert wird.

Den Hauptteil der Arbeit übernimmt das `PropBag`-Objekt, das vom `UserControl`-Objekt während der Ereignisse `ReadProperties` und `WriteProperties` bereitgestellt wird. Das `PropBag`-Objekt hat zwei Methoden. Die `Read`-Methode nimmt einen Eigenschaftsnamen und den Standardwert als Vorgabe entgegen, und gibt den gespeicherten Eigenschaftswert zurück. Die Funktion gibt einen Variant zurück und kann damit jeden Eigenschaftstyp verarbeiten. Die `Write`-Methode nimmt einen Eigenschaftsnamen, einen Wert und einen Standardwert als Parameter entgegen. Die Bedeutung von solchen Vorgabewerten ist in Kapitel 17 genauer beschrieben.

Visual Basic will effizient sein und speichert nur Eigenschaften, die geändert wurden. Sie müssen also unbedingt die `PropertyChanged`-Methode des `UserControl`-Objekts aufrufen, wenn eine persistente Eigenschaft geändert wurde. Wenn Sie das nicht tun, werden die Änderungen, die Sie zur Entwurfszeit an den Eigenschaften vornehmen, nicht gespeichert, und das VB-Eigenschaftsfenster wird nicht aktualisiert, um den aktuellen Eigenschaftswert anzuzeigen. Es passiert nichts, wenn diese Methode zur Laufzeit aufgerufen wird, Sie brauchen also im Code, der Ihre Eigenschaft setzt, nicht zwischen Laufzeit und Entwurfszeit unterscheiden.

In früheren Codebeispielen haben Sie bereits einige einfache Beispiele für die Persistenz kennengelernt – ohne sie wäre es fast unmöglich, mit einem ActiveX-Steuerelement etwas Sinnvolles zu erledigen. Leider können die einfachen Beispiele, die Sie so häufig sehen, zu einer etwas eingeschränkten Sicht auf die Persistenz der Eigenschaften führen. Erlauben Sie mir also eine schärfere Aussage: Das `PropBag`-Objekt und seine Methoden sowie die Ereignisse `ReadProperties` und `WriteProperties` haben nichts mit den Eigenschaften Ihres Steuerelements zu tun.

OK, vielleicht war das etwas drastisch. Schließlich verwenden Sie diese Elemente von Visual Basic oft, um die Eigenschaftswerte Ihres Steuerelements persistent zu machen. Aber ich stehe auf folgendem Standpunkt: Die Ereignisse `ReadProperties` und `WriteProperties` sollen den Zeitpunkt signalisieren, zu dem die persistenten Daten Ihres Steuerelements gespeichert oder geladen werden sollen. Das `PropBag`-Objekt wird genutzt, um die Speicher- und Ladeoperationen auszuführen.

Aber ob die persistenten Daten Ihres Steuerelements in irgendeiner Beziehung zu den Eigenschaften Ihres Steuerelements stehen, bleibt völlig Ihnen überlassen. Mit anderen Worten:

- Der Eigenschaftsname, unter dem Sie Daten unter Verwendung des `PropBag`-Objekts speichern, muß nicht mit dem Namen einer Eigenschaft in Ihrem Steuerelement übereinstimmen.
- Mit Hilfe des `PropBag`-Objekts können Sie Daten speichern, die keiner Steuerelement-Eigenschaft entsprechen.
- Sie können mehrere `PropBag`-Eigenschaften verwenden, um Informationen zu einer Steuerelementeigenschaft zu speichern.

Der Eigenschaftsname, unter dem Sie Daten speichern, ist nur ein Bezeichner – er hat überhaupt nichts mit den Eigenschaften in Ihrem Steuerelement zu tun. Damit ist es offensichtlich, daß Sie bei der Verwendung des `PropBag`-Objekts zum Speichern von Steuerelementeigenschaften die Daten unter dem Eigenschaftsnamen ablegen sollten, um die Leute nicht zu verwirren, die Formulardateien oder HTML-Seiten lesen, die Ihr Steuerelement verwenden. Aber das muß nicht unbedingt so sein.

Das Steuerelement `ch19ctlB` im Projekt `ch19ctls` demonstriert den Unterschied zwischen persistenten Daten und Steuerelementeigenschaften. Das Steuerelement implementiert die nur lesbare Eigenschaft `CreationTime`, die immer das Datum und die Zeit enthält, wann einem Container die Steuerelementinstanz hinzugefügt wurde.

Die Zeit wird in der Datumsvariablen `m_CreationTime` abgelegt. Diese Variable wird während des `InitProperties`-Ereignisses von `UserControl` initialisiert. Dieses Ereignis wird nur aufgerufen, wenn eine Steuerelementinstanz erzeugt wird, wie im folgenden Code gezeigt:

```
Private m_CreationTime As Date

Private Sub UserControl_InitProperties()
   m_CreationTime = Now()
   BackColor = Ambient.BackColor
   Set Font = Ambient.Font
   Label3.Caption = m_CreationTime
   PropertyChanged "CreationTime"
End Sub
```

Das `Label3`-Standardelement zeigt die Erstellungszeit an. Die `PropertyChanged`-Methode wird aufgerufen, um sicherzustellen, daß die Änderung gespeichert wurde.

Die Erstellungszeit wird während des `WriteProperties`-Ereignisses gespeichert:

```
Call PropBag.WriteProperty("ControlCreationTime", m_CreationTime)
```

und während des ReadProperties-Ereignisses gelesen:

```
m_CreationTime = PropBag.ReadProperty("ControlCreationTime",
m_CreationTime)
     Label3.Caption = m_CreationTime
```

Die Eigenschaft wird mit Hilfe der Property Get-Prozedur gelesen:

```
     Public Property Get CreationTime() As Date
        CreationTime = m_CreationTime
     End Property

     Public Property Let CreationTime(ByVal vNewValue As Date)
        Err.Raise 383
     End Property
```

Der Name, unter dem die Daten abgelegt werden (ControlCreationTime), wurde intern gewählt, um sich von dem Eigenschaftsnamen zu unterscheiden, und so zu betonen, daß die beiden nicht in direktem Zusammenhang stehen. Die Property Let-Prozedur wirft immer den Fehler »Set nicht unterstützt« auf. Die Eigenschaft wird im Dialog EXTRAS/PROZEDURATTRIBUTE mit der Option IM EIGENSCHAFTENKATALOG NICHT ANZEIGEN markiert, so daß sie nicht im Eigenschaftenkatalog (VB-Eigenschaftsfenster) erscheint.

Weil es keine Möglichkeit gibt, die Eigenschaft CreationTime zur Entwurfszeit oder zur Laufzeit zu setzen, bleibt sie immer auf dem Wert, der beim ersten Plazieren im Container gesetzt wurde.

19.5.1 Eigenschaften zuordnen

Die Tatsache, daß persistent gemachte Daten keine direkte Beziehung zu einer bestimmten Eigenschaft oder Variablen in einem Steuerelement haben, bietet eine enorme Flexibilität für die Nutzung dieser Daten. Sie haben bereits mehrere Situationen kennengelernt, wo einzelne Datenelemente mehr als einer Eigenschaft oder einer Variablen zugeordnet werden können. Das wird im Steuerelement ch10ctlB mit den Eigenschaften BackColor und Font demonstriert.

Die BackColor-Eigenschaft wird genutzt, um die Background-Eigenschaft für das Steuerelement und die beiden Bildfelder zu setzen. Die Font-Eigenschaft wird genutzt, um die Schriften für die drei Bezeichnungsfelder zu setzen. Die Eigenschaftsprozeduren für diese Eigenschaften sehen wie folgt aus:

```
Public Property Get BackColor() As OLE_COLOR
        BackColor = UserControl.BackColor
     End Property

     Public Property Let BackColor(ByVal New_BackColor As OLE_COLOR)
        UserControl.BackColor() = New_BackColor
        Picture1.BackColor = New_BackColor
```

```
        Picture2.BackColor = New_BackColor
        PropertyChanged "BackColor"
     End Property

Public Property Get Font() As Font
   Set Font = m_Font1
End Property

Public Property Set Font(ByVal New_Font As Font)
   Dim lbl As Object
   With m_Font1
      .Bold = New_Font.Bold
      .Charset = New_Font.Charset
      .Italic = New_Font.Italic
      .Name = New_Font.Name
      .Size = New_Font.Size
      .Strikethrough = New_Font.Strikethrough
      .Underline = New_Font.Underline
      .Weight = New_Font.Weight
   End With

   For Each lbl In UserControl.Controls
      If TypeOf lbl Is Label Then Set lbl.Font = m_Font1
   Next
   PropertyChanged "Font"
End Property
```

Die `BackColor`-Eigenschaft demonstriert eine Möglichkeit, eine einzige Eigenschaft mehreren Objekten zuzuordnen. Die `Font`-Eigenschaft demonstriert, wie Sie die Steuerelementauflistung nutzen können, um dasselbe zu tun. Die Ereignisse `ReadProperties` und `WriteProperties` werden wie folgt implementiert:

```
     'Eigenschaftswerte aus dem Speicher laden
     Private Sub UserControl_ReadProperties(PropBag As PropertyBag)
        ' Durch Verwendung der öffentlichen Eigenschaft setzen wir alle
auf 3,
        ' rufen aber auch unnötigerweise PropertyChanged auf
        BackColor = PropBag.ReadProperty("BackColor", &H8000000F)
        Set Font = PropBag.ReadProperty("Font", Nothing)
     End Sub

     'Eigenschaftswerte in den Speicher schreiben
     Private Sub UserControl_WriteProperties(PropBag As PropertyBag)
        Call PropBag.WriteProperty("BackColor", UserControl.BackColor,
&H8000000F)
        Call PropBag.WriteProperty("Font", Font, Nothing)
 End Sub
```

Während des ReadProperties-Ereignisses können Sie die Object-Eigenschaften direkt laden oder die Property Get-Prozedur dafür verwenden, wie hier gezeigt.

19.5.2 Selbst-persistente Objekte

Früher in diesem Kapitel haben Sie gesehen, wie es möglich war, eine benutzerdefinierte Object-Eigenschaft zu erzeugen, die zur Entwurfszeit unter Verwendung einer Eigenschaftsseite bearbeitet werden kann. Das Steuerelement ch19ctlA demonstriert eine Möglichkeit, ein solches Objekt persistent zu machen. Die Klasse clsMyObject enthält die beiden folgenden Funktionen, die vom Steuerelement aufgerufen werden:

```
' Eine Methode, Eigenschaften zu speichern
Friend Sub ReadProperties(PropBag As PropertyBag, propname$)
   m_Numerator = PropBag.ReadProperty(propname$ & "Numerator", 0)
   m_Denominator = PropBag.ReadProperty(propname$ & "Denominator", 1)
End Sub

Friend Sub WriteProperties(PropBag As PropertyBag, propname$)
   PropBag.WriteProperty propname$ & "Numerator", m_Numerator, 0
   PropBag.WriteProperty propname$ & "Denominator", m_Denominator, 1
End Sub
```

Die Funktionen sind als Friend-Funktionen definiert, weil es keinen besonderen Grund gibt, sie den Endbenutzern zur Verfügung zu stellen. Sie müßten sie öffentlich machen, wenn die Klasse in ihrer eigenen ActiveX-DLL definiert würde. Die Funktionen werden in den ReadProperties- und WriteProperties-Ereignissen von ch10ctlA wie folgt aufgerufen:

Im ReadProperties-Ereignis:

```
Call m_MyObject1.ReadProperties(PropBag, "MyObject1")
```

Im WriteProperties-Ereignis:

```
m_MyObject1.WriteProperties PropBag, "MyObject1"
```

In diesem Beispiel speichert das Objekt seine internen Daten in mehreren Eigenschaften. Der Name des Objekts muß der Komponente übergeben werden, so daß sie einen eindeutigen Eigenschaftsnamen erzeugen kann (falls es mehrere Eigenschaften gibt, die dieses Objekt nutzen).

Beachten Sie, daß das PropBag-Objekt auch binäre Daten (Byte-Felder) verarbeiten kann. Wenn Sie also ein komplexes Objekt mit vielen internen Variablen haben, können Sie diese immer in ein einzelnes Feld laden und sie in einem Datenblock speichern.

Vielleicht bemerken Sie, daß die `Font`- und `Picture`-Eigenschaften durch das `PropBag`-Objekt völlig selbst-persistent sind. Visual Basic bietet einen Mechanismus, das auch so zu realisieren, wie in der Klasse `clsMyObject2` gezeigt. Diese Klasse ist identisch mit der Klasse `clsMyObject1`, außer für die Persistenz. Das Steuerelement macht das Objekt unter Verwendung desselben Codetyps persistent, der von den `Font`- und `Picture`-Eigenschaften verwendet wird. Im `ReadProperties`-Ereignis sehen Sie den folgenden Code:

```
Set m_MyObject2 = PropBag.ReadProperty("MyObject2", m_MyObject2)
```

Und im `WriteProperties`-Ereignis sehen Sie den folgenden Code:

```
PropBag.WriteProperty "MyObject2", m_MyObject2, Nothing
```

Damit diese Form der Selbst-Persistenz funktioniert, muß die `Persistable`-Eigenschaft des Klassenmoduls auf `True` gesetzt werden. Damit werden dem Klassenmodul die Ereignisse `ReadProperties` und `WriteProperties` hinzugefügt, das den folgenden Code enthält, um die einzelnen Elementvariablen persistent zu machen:

```
Private Sub Class_ReadProperties(PropBag As PropertyBag)
   m_Numerator = PropBag.ReadProperty("Numerator", 0)
   m_Denominator = PropBag.ReadProperty("Denominator", 1)
End Sub

Private Sub Class_WriteProperties(PropBag As PropertyBag)
   PropBag.WriteProperty "Numerator", m_Numerator, 0
   PropBag.WriteProperty "Denominator", m_Denominator, 1
End Sub
```

Betrachten Sie den Unterschied zwischen den Methoden, wie die Formulare im Formular-Header persistent gemacht werden:

```
Begin gtpch19ctls.ch19ctlA ch19ctlA2
      Height          =   915
      Left            =   450
      TabIndex        =   2
      Top             =   1530
      Width           =   2085
      _ExtentX        =   3678
      _ExtentY        =   1614
      Variant1        =   1
      BeginProperty Font1 {0BE3520-8F91-11CE-9DE3-00AA004BB851}
         Name            =   "MS Sans Serif"
         Size            =   8.25
         Charset         =   0
         Weight          =   400
         Underline       =   0   'False
         Italic          =   0   'False
         Strikethrough   =   0   'False
```

```
EndProperty
MyObject1Denominator=   2
BeginProperty MyObject2 {2A82BDB8-21C0-11D2-A750-00001C1AD1F8}
   Denominator     =   2
EndProperty
```

Das Objekt `MyObject1` wird auf derselben Ebene wie das Steuerelement persistent gemacht, während der Eigenschaftsname zwischen Objekten unterscheidet. Das `MyObject2`-Objekt wird als eigenes Unterobjekt persistent gemacht, wobei die CLSID das persistent zu machende Objekt angibt.

Die Klassen-Eigenschaft `Instancing` muß ebenfalls auf `MultiUse` gesetzt werden. Warum? Weil der CLSID-Bezeichner im Formular gespeichert wird.

Nachdem Visual Basic selbst-persistente Objekte unterstützt, könnten Sie versucht sein, diesen Ansatz immer zu verwenden. Aber glauben Sie es oder nicht, es gibt Situationen, wo der für den `MyObject1`-Fall verwendete Ansatz besser ist, auch wenn er weniger elegant sein mag. Warum?

- Der selbst-persistente Ansatz macht es erforderlich, daß das Objekt öffentlich ist, so daß das `ReadProperties`-Ereignis es von Grund auf neu erzeugen kann. Sie möchten vielleicht nicht, daß Ihr Sub-Objekt öffentlich erzeugt werden kann.
- Der selbst-persistente Ansatz ist wesentlich langsamer, weil er bedingt, daß in der Registrierung nachgeschlagen und der Mechanismus zum Erzeugen des Standard-COM-Objekts aufgerufen wird.

Es gibt zwei weitere Aspekte, die berücksichtigt werden sollten, wenn selbst-persistente Klassen erzeugt werden, indem die `Persistable`-Eigenschaft der Klasse auf `True` gesetzt wird.

Sie brauchen kein Steuerelement, das diese Klassentypen persistent macht. Sie können direkt ein `PropBag`-Objekt erzeugen und die Methoden `ReadProperties` und `WriteProperties` zu beliebigen Zeitpunkten aufrufen, um ein `PropertyBag` zu lesen oder zu schreiben. Das scheint nicht sehr sinnvoll zu sein, bis Sie erkennen, daß Sie auf den Inhalt eines `PropertyBags` in Form eines Byte-Felds zugreifen können, das wiederum als Datei oder OLE-strukturierte Speicherdatei geschrieben werden kann (mit einem Werkzeug wie beispielsweise den Storage-Tools von Desaware). Damit ist es möglich, fast alle VB-erzeugten COM-Komponenten auf einfache Weise persistent zu machen. Details, wie das funktioniert, finden Sie in der VB-Dokumentation.

Bei der Arbeit mit selbst-persistenten Klassen ist es extrem wichtig, niemals die CLSID des Objekts zu ändern. Wenn sie sich ändert, kann VB kein neues Objekt des korrekten Typs erzeugen. Das führt uns zum Thema der Versionsbildung, auf das wir später eingehen werden. Hier ist die beste Möglichkeit, die CLSID eines Objekts beizubehalten, die Komponente zu kompilieren und die Binär-Kompatibilität für die Komponente zu setzen. Wenn Sie während eines Aufrufs von `Read-`

`Properties` einen Fehler erhalten, in dem Visual Basic eine unzulässige Typzuweisung moniert, dann deshalb, weil VB versucht, ein Objekt mit einer alten CLSID und einem alten Server zu erzeugen, die nicht mit der aktuellen CLSID übereinstimmen. Wenn `ReadProperties` angibt, daß es kein Objekt mit einer bestimmten CLSID erzeugen kann, dann in der Regel deshalb, weil sich die CLSID geändert hat und es keinen alten Server mit der vorherigen CLSID gibt. VB scheint manchmal alte CLSID-Werte in Formularen aufzubewahren, Sie könnten also versuchen, ein Formular als Textdatei zu öffnen und die Zeilen zu löschen, in denen die Eigenschaft festgeschrieben wird, um ein neues Objekt zu starten, ohne das Formular neu anlegen zu müssen.

Der sicherste Ansatz beim Erzeugen selbst-persistenter Objekte ist, sie in eigenen, separaten ActiveX-DLLs zu plazieren. Damit ist es für Sie einfacher, beim Testen die Versionssteuerung zu verwalten.

Oh ja, in Version 6 können die ActiveX-Steuerelemente in Visual Basic öffentliche Klassen-Objekte bereitstellen. Ich vermute, daß das den Zweck haben sollte, diese Art selbst-persistenter Klasse zu erlauben, obwohl man ja nie weiß. Dies war eine der Funktionen, die am lautesten gefordert wurden.

19.5.3 Besuchen wir den Assistenten!

Visual Basic beschäftigt den Schnittstellen-Assistenten für ActiveX-Steuerelemente. Er hilft Ihnen, Ihre ActiveX-Steuerelemente zu erzeugen und zu verwalten.

Wie Sie vielleicht schon bemerkt haben, bin ich immer etwas skeptisch, was die Verwendung von Assistenten betrifft. Sie können zur Krücke werden, die das eigene Verständnis und eigene Entwürfe völlig verkümmern lassen, und Sie das Ganze einfach durch die Fähigkeiten eines anderen Programmierers ersetzen, der vielleicht gar nicht so intelligent ist wie Sie und auch nicht weiß, welche Anforderungen Ihre Applikation stellt. Weil Sie den zugrundeliegenden Code nicht selbst erstellt haben, sind Sie vielleicht auch nicht auf Möglichkeiten aufmerksam geworden, die vom Assistenten nicht implementiert wurden, die aber für die Lösung Ihres jeweiligen Problems sehr nützlich sein könnten.

Nachdem ich das nun gesagt habe: Es gibt zwei Situationen, in denen Assistenten extrem nützlich sein können. Erstens erledigen sie die müßige Arbeit, Routinen zu schreiben, die Sie selbst auch entwickeln könnten. Und zweitens helfen sie, die richtige Methode zu erlernen, bestimmte Softwareprobleme zu lösen.

Dieses Buch hat Ihnen gezeigt, wie man ActiveX-Komponenten ohne die Hilfe eines Assistenten erzeugt. Nachdem Sie die dabei verwendeten Techniken zur Genüge kennen, sollten Sie die von Microsoft bereitgestellten Assistenten betrachten. Insbesondere der Schnittstellen-Assistent für ActiveX-Steuerelemente ist sehr praktisch. Listing 19.2 zeigt den Code, den dieser Assistent erzeugt, mit mehreren Standardeigenschaften und Eigenschaften, die dem UserControl-Objekt zugeordnet werden.

```
Option Explicit
'Ereignis-Deklarationen:
Event KeyDown(KeyCode As Integer, Shift As Integer)
' MappingInfo=UserControl,UserControl,-1,KeyDown
Event KeyPress(KeyAscii As Integer)
' MappingInfo=UserControl,UserControl,-1,KeyPress
Event KeyUp(KeyCode As Integer, Shift As Integer)
' MappingInfo=UserControl,UserControl,-1,KeyUp
Event MouseDown(Button As Integer, Shift As Integer, X As Single,_
 Y As Single) 'MappingInfo=UserControl,UserControl,-1,MouseDown
Event MouseMove(Button As Integer, Shift As Integer, X As Single,_
 Y As Single) 'MappingInfo=UserControl,UserControl,-1,MouseMove
Event MouseUp(Button As Integer, Shift As Integer, X As Single,_
 Y As Single)
'MappingInfo=UserControl,UserControl,-1,MouseUp
Event Click() 'MappingInfo=UserControl,UserControl,-1,Click
Event DblClick() 'MappingInfo=UserControl,UserControl,-1,DblClick


Private Sub UserControl_KeyDown(KeyCode As Integer, Shift As Integer)
   RaiseEvent KeyDown(KeyCode, Shift)
End Sub

Private Sub UserControl_KeyPress(KeyAscii As Integer)
   RaiseEvent KeyPress(KeyAscii)
End Sub

Private Sub UserControl_KeyUp(KeyCode As Integer, Shift As Integer)
   RaiseEvent KeyUp(KeyCode, Shift)
End Sub

Private Sub UserControl_MouseDown(Button As Integer, Shift As Integer, _
X As Single, Y As Single)
   RaiseEvent MouseDown(Button, Shift, X, Y)
End Sub

Private Sub UserControl_MouseMove(Button As Integer, Shift As Integer, _
X As Single, Y As Single)
   RaiseEvent MouseMove(Button, Shift, X, Y)
End Sub

Private Sub UserControl_MouseUp(Button As Integer, Shift As Integer, X As _
Single, Y As Single)
   RaiseEvent MouseUp(Button, Shift, X, Y)
End Sub

'ACHTUNG: DIE FOLGENDEN AUSKOMMENTIERTEN ZEILEN NICHT ENTFERNEN ODER ÄNDERN!
```

```
'MappingInfo=UserControl,UserControl,-1,BackColor
Public Property Get BackColor() As OLE_COLOR
   BackColor = UserControl.BackColor
End Property

Public Property Let BackColor(ByVal New_BackColor As OLE_COLOR)
   UserControl.BackColor() = New_BackColor
   PropertyChanged "BackColor"
End Property

'ACHTUNG: DIE FOLGENDEN AUSKOMMENTIERTEN ZEILEN NICHT ENTFERNEN ODER ÄNDERN!
'MappingInfo=UserControl,UserControl,-1,ForeColor
Public Property Get ForeColor() As OLE_COLOR
   ForeColor = UserControl.ForeColor
End Property

Public Property Let ForeColor(ByVal New_ForeColor As OLE_COLOR)
   UserControl.ForeColor() = New_ForeColor
   PropertyChanged "ForeColor"
End Property

'ACHTUNG: DIE FOLGENDEN AUSKOMMENTIERTEN ZEILEN NICHT ENTFERNEN ODER ÄNDERN!
'MappingInfo=UserControl,UserControl,-1,Enabled
Public Property Get Enabled() As Boolean
   Enabled = UserControl.Enabled
End Property

Public Property Let Enabled(ByVal New_Enabled As Boolean)
   UserControl.Enabled() = New_Enabled
   PropertyChanged "Enabled"
End Property

'ACHTUNG: DIE FOLGENDEN AUSKOMMENTIERTEN ZEILEN NICHT ENTFERNEN ODER ÄNDERN!
'MappingInfo=UserControl,UserControl,-1,Font
Public Property Get Font() As Font
   Set Font = UserControl.Font
End Property

Public Property Set Font(ByVal New_Font As Font)
   Set UserControl.Font = New_Font
   PropertyChanged "Font"
End Property

'ACHTUNG: DIE FOLGENDEN AUSKOMMENTIERTEN ZEILEN NICHT ENTFERNEN ODER ÄNDERN!
'MappingInfo=UserControl,UserControl,-1,BackStyle
Public Property Get BackStyle() As Integer
   BackStyle = UserControl.BackStyle
End Property
```

```
Public Property Let BackStyle(ByVal New_BackStyle As Integer)
   UserControl.BackStyle() = New_BackStyle
   PropertyChanged "BackStyle"
End Property

'ACHTUNG: DIE FOLGENDEN AUSKOMMENTIERTEN ZEILEN NICHT ENTFERNEN ODER ÄNDERN!
'MappingInfo=UserControl,UserControl,-1,BorderStyle
Public Property Get BorderStyle() As Integer
   BorderStyle = UserControl.BorderStyle
End Property

Public Property Let BorderStyle(ByVal New_BorderStyle As Integer)
   UserControl.BorderStyle() = New_BorderStyle
   PropertyChanged "BorderStyle"
End Property

'ACHTUNG: DIE FOLGENDEN AUSKOMMENTIERTEN ZEILEN NICHT ENTFERNEN ODER ÄNDERN!
'MappingInfo=UserControl,UserControl,-1,Refresh
Public Sub Refresh()
   UserControl.Refresh
End Sub

Private Sub UserControl_Click()
   RaiseEvent Click
End Sub

Private Sub UserControl_DblClick()
   RaiseEvent DblClick
End Sub

'Initialize Properties for User Control
Private Sub UserControl_InitProperties()
   Set Font = Ambient.Font
End Sub

'Eigenschaftswerte aus dem Speicher laden
Private Sub UserControl_ReadProperties(PropBag As PropertyBag)

   UserControl.BackColor = PropBag.ReadProperty("BackColor", &H8000000F)
   UserControl.ForeColor = PropBag.ReadProperty("ForeColor", &H80000012)
   UserControl.Enabled = PropBag.ReadProperty("Enabled", True)
   Set Font = PropBag.ReadProperty("Font", Ambient.Font)
   UserControl.BackStyle = PropBag.ReadProperty("BackStyle", 1)
   UserControl.BorderStyle = PropBag.ReadProperty("BorderStyle", 0)
End Sub
```

```
'Eigenschaftswerte in den Speicher schreiben
Private Sub UserControl_WriteProperties(PropBag As PropertyBag)

   Call PropBag.WriteProperty("BackColor", UserControl.BackColor, &H8000000F)
   Call PropBag.WriteProperty("ForeColor", UserControl.ForeColor, &H80000012)
   Call PropBag.WriteProperty("Enabled", UserControl.Enabled, True)
   Call PropBag.WriteProperty("Font", Font, Ambient.Font)
   Call PropBag.WriteProperty("BackStyle", UserControl.BackStyle, 1)
   Call PropBag.WriteProperty("BorderStyle", UserControl.BorderStyle, 0)
End Sub
```

Listing. 19.2: Code, den der Schnittstellen-Assistent für ActiveX-Steuerelemente erzeugt

Bei der Verwendung des Assistenten sollten Sie mehrere Dinge beachten:

- Prüfen Sie die Prozedur-IDs der Standard-Eigenschaften und Ereignisse und stellen Sie sicher, daß sie auf die richtigen Standard-Prozedur-IDs gesetzt sind. Das ist insbesondere für die `Enabled`-Eigenschaft wichtig, wenn diese korrekt funktionieren soll.
- Der Assistent setzt keine Eigenschaften auf Aufzählungstypen. Sie sollten den Eigenschaftstyp ändern, wenn Sie möchten, daß das Eigenschaftsfenster von Visual Basic eine aufzählende Liste aller für die Eigenschaft möglichen Werte anzeigt.
- Wenn Sie die Standard-Eigenschaft des `UserControl`-Objekts im Eigenschaftenfenster von Visual Basic ändern, während Sie das Steuerelement entwikkeln, sollten Sie darauf achten, die Vorgabewerte im Aufruf von `PropBag.ReadProperty` und `PropBag.WriteProperty` für diese Eigenschaft zu ändern. Dies ist im Beispiel mit der `BackStyle`-Eigenschaft demonstriert.
- Der für die `Font`-Eigenschaft erzeugte Assistentencode stimmt nicht mit der aktuell empfohlenen Methode für die Verarbeitung von Schrifteigenschaften überein.

19.5.4 Asynchrone Persistenz

Sie haben gesehen, wie ein Container ein `PropertyBag` nutzen kann, um Eigenschaften persistent zu machen. Wenn Daten in ein `PropertyBag` geschrieben werden, kann der Container zahlreiche Dinge damit tun, abhängig vom Datentyp. Ist der Datentyp ein String, eine Zahl oder ein anderer Typ, der als Text dargestellt werden kann, kann der Container die Information in Textform ablegen. Wenn Sie mit Hilfe eines Texteditors eine Formulardatei betrachten, sehen Sie die Eigenschaften als Strings. Die Eigenschaften für eine typische Schaltfläche können wie folgt dargestellt werden:

```
Begin VB.CommandButton cmdSet
   Caption         =   "Set"
   Height          =   465
   Left            =   3420
   TabIndex        =   2
   Top             =   810
   Width           =   1005
End
```

Aber was ist mit den Daten, die nicht als Text dargestellt werden können? Bilder und andere binäre Datentypen müssen in separaten Dateien abgelegt werden. Eine Bild-Eigenschaft könnte damit wie folgt gespeichert werden:

```
Picture         =   "Form1.frx":0000
```

Die Formulardatei enthält einen Verweis auf eine Datendatei (mit der Erweiterung `.FRX`), die die eigentlichen Daten enthält.

Jetzt wollen wir einen Moment lang überlegen, was es bedeutet, ein ActiveX-Steuerelement auf einer Web-Seite zu plazieren. Eigenschaften, die als Text dargestellt werden können, sind ganz einfach zu verarbeiten – sie können direkt auf der Web-Seite angezeigt werden. HTML bietet ebenfalls eine Methode, die binären Daten auf die Seite aufzunehmen, aber wer möchte schon jemanden zwingen, Hunderte von Kilobyte, wenn nicht gar Megabyte binärer Daten herunterzuladen, um eine Web-Seite zu sehen?

Offensichtlich braucht man hier eine Methode, ein Steuerelement anzuweisen, Daten von einer anderen Position zu laden. Und um eine vernünftige Performance zu gewährleisten, braucht man eine Möglichkeit, Daten asynchron im Hintergrund herunterzuladen, wenn das Steuerelement so schnell wie möglich angezeigt werden oder eine erste Funktionalität bereitstellen soll. Beispielsweise könnten Sie veranlassen, daß die `Picture`-Eigenschaft eines Steuerelements im Hintergrund geladen wird, so daß das Steuerelement sofort angezeigt und unmittelbar gestartet wird, während das richtige Bild erst noch aufgebaut wird.

Die ActiveX-Steuerelemente von Visual Basic unterstützen diese Art des asynchronen Downloads. Das Steuerelement `AsyncCtl.ctl` demonstriert einen Ansatz, ein Bild asynchron zu laden. Das Programm funktioniert in jedem Container, aber der Download ist nur in Containern wie dem Microsoft Explorer asynchron, wo diese Funktion unterstützt wird. Den relevanten Code sehen Sie in Listing 19.3.

```
Const m_def_PictureName = ""
'Eigenschaftsvariablen:
Dim m_PictureName As String

Public Property Get Picture() As Picture
   Set Picture = UserControl.Picture
End Property
```

```
    Public Property Set Picture(ByVal vNewValue As Picture)
       Set UserControl.Picture = vNewValue
    End Property

    Public Property Get PictureName() As String
       PictureName = m_PictureName
    End Property

    Public Property Let PictureName(ByVal New_PictureName As String)
       m_PictureName = New_PictureName
       UserControl.AsyncRead m_PictureName, vbAsyncTypePicture, "Picture"
       PropertyChanged "PictureName"
    End Property

    Private Sub UserControl_ReadProperties(PropBag As PropertyBag)
       PictureName = PropBag.ReadProperty("PictureName", m_def_PictureName)
    End Sub

    Private Sub UserControl_WriteProperties(PropBag As PropertyBag)
       Call PropBag.WriteProperty("PictureName", m_PictureName,_
m_def_PictureName)
    End Sub

    Private Sub UserControl_AsyncReadComplete(AsyncProp As AsyncProperty)
       On Error GoTo AsyncErr
       Select Case AsyncProp.PropertyName
          Case "Picture"
              Debug.Print "Picture arrived"
              Set Picture = AsyncProp.Value
       End Select
       Exit Sub
    AsyncErr:
       ' Wenn Sie versuchen, auf den Wert zuzugreifen, werden Fehler
       ' für asynchrone Operationen aufgeworfen
    End Sub

Private Sub UserControl_AsyncReadProgress(AsyncProp As AsyncProperty)
   Debug.Print AsyncProp.BytesRead
End Sub
```

Listing. 19.3: Der Code von AsynchCtl.ctl für die asynchrone Persistenz

In diesem Fall wird die `Picture`-Eigenschaft im Dialogfeld EXTRAS/PROZEDURATTRIBUTE als nicht sichtbar im Eigenschaftskatalog markiert. Wir müssen nämlich nicht die `Picture`-Eigenschaft persistent machen. Statt dessen machen wir die Eigenschaft `PictureName` persistent, die im Eigenschaftenfenster von Visual Basic erscheint und mit Hilfe der Ereignisse `ReadProperties` und `Write-`

Properties persistent gemacht wird, wie bereits gezeigt wurde. Diese Eigenschaft enthält die Position, von der das Steuerelement das Bild laden soll. Diese Position kann als Netzwerk-URL oder als Pfad zu einer lokalen Datei angegeben werden. Wenn diese Eigenschaft gesetzt wird, entweder über das ReadProperties-Ereignis oder über die Property Let-Prozedur, ruft sie die AsyncRead-Methode des UserControl-Objekts auf.

Die AsyncRead-Methode nimmt drei Parameter entgegen. Der erste ist der Pfad oder die URL (Web-Adresse), unter dem bzw. der sich die gewünschte Datei befindet. Der zweite Parameter gibt den Datentyp an, Bild, Datei oder Binär. Der letzte Parameter ist ein Bezeichner, der angibt, welche Daten geladen werden sollen. Sie setzen beliebig viele Eigenschaften, die im Hintergrund geladen werden sollen, und mit dem Bezeichner werden diese einzeln identifiziert. Dieser Bezeichner wird in der Regel auf den Namen der Eigenschaft gesetzt, was aber gänzlich Ihnen überlassen bleibt.

Nachdem die Daten geladen sind, wird das AsyncReadComplete-Ereignis ausgelöst. Dieses Ereignis verwendet als einzigen Parameter einen Verweis auf ein AsyncProperty-Objekt. Dieses Objekt erlaubt Ihnen, den Datentyp zu ermitteln, ebenso wie den Bezeichner, der während des Aufrufs von AsyncRead angegeben wurde. Außerdem enthält es eine Value-Eigenschaft, einen Variant, der die zurückgegebenen Daten aufnimmt. Value enthält ein Bild, falls Sie einen Bildtyp angegeben haben, den Namen einer temporären Datei, falls Sie einen Dateityp angegeben haben, oder ein Byte-Feld, falls Sie einen binären Typ angegeben haben.

Wenn während der Leseoperation ein Fehler aufgetreten ist, wird ein Fehler aufgeworfen, sobald Sie auf die Value-Eigenschaft zugreifen. Es ist also wichtig, während des AsyncReadComplete-Ereignisses immer eine Fehlerverarbeitung bereitzustellen. Und Sie sollten nie einen Fehler während dieses Ereignisses aufwerfen. Dieses Ereignis kann jederzeit auftreten, und die Container können nicht immer Fehler verarbeiten, die Sie aufwerfen, während es verarbeitet wird.

Asynchrone Lesevorgänge können auf jedem Container problemlos für lokale Dateien ausgeführt werden. Nur Browser-fähige Container unterstützen URL-Pfade. Ob die Operation wirklich asynchron stattfindet oder nicht, ist ebenfalls von dem Container abhängig, und ob der Zugriff über eine URL oder einen Pfad erfolgt. Sie sehen das Verhalten des asynchronen Ladens, indem Sie das Steuerelement in der VB-Umgebung ausführen. Wenn Sie ein Steuerelement auf diese Weise ausführen, erzeugt Visual Basic eine temporäre Web-Seite, etwa wie diese:

```
<HTML>
<BODY>
<OBJECT classid="clsid:5E08955D-8E90-11D0-91BB-00AA0036005A">
</OBJECT>
</BODY>
</HTML>
```

Die classid ist die aktuelle CLSID des Steuerelements. Visual Basic startet Ihren Browser (den Internet Explorer, weil dies der einzige Browser ist, der zuverlässig ActiveX-Steuerelemente anzeigt). Das Steuerelement wird geladen und auf einer leeren Seite des Browsers angezeigt.

Das Steuerelement hat zwei Schaltflächen, LOCAL und WEB. Die Schaltfläche LOCAL lädt ein großes Bild, das sich im Projekt-Verzeichnis befindet. Die Web-Schaltfläche lädt ein Bild von der Web-Site von Desaware. Während das Hintergrundbild geladen wird, wird immer wieder das `AsyncReadProgress`-Ereignis aufgerufen. Während des Ereignisses können Sie das `AsyncProp`-Objekt lesen, um weitere Informationen über den Fortschritt der Übertragung zu erhalten.

19.5.5 Steuerelemente aktualisieren

Was passiert, wenn Sie irgendwann eine persistente Eigenschaft (oder andere Daten, die mit `ReadProperties` oder `WriteProperties` persistent gemacht wurden) Ihres Steuerelements hinzufügen oder entfernen? Wenn Sie einem Steuerelement eine persistente Eigenschaft hinzufügen, sollten Sie einen Vorgabewert bereitstellen. Andernfalls kann ein Fehler auftreten, wenn das Steuerelement versucht, ein Projekt zu lesen, das mit einem älteren Steuerelement gespeichert wurde. Ist ein Vorgabewert angegeben, wird er in die Variable geladen, falls die Eigenschaft in der Datei nicht gefunden wurde. Persistente Daten können problemlos hinzugefügt werden.

Eine vollständigere Beschreibung der Themen zur Aktualisierung von Steuerelementen finden Sie in Kapitel 25.

19.6 Datenbindung

Die Datenbindung bezieht sich auf den Prozeß, eine oder mehrere Eigenschaften eines ActiveX-Steuerelements zu den Feldern einer Datenbank zu binden. Das Problem bei der Beschreibung eines solchen Themas ist, daß es schnell so umfangreich werden kann, nicht nur das restliche Buch, sondern auch zahlreiche weitere Bände davon zu füllen.

Dieser Abschnitt setzt voraus, daß Sie wenigstens ein Grundwissen über die Datenbindung mitbringen, wie sie für Standard-Steuerelemente implementiert wird, beispielsweise Textfelder oder Bezeichnungsfelder. Das sollte reichen, damit Sie die »einfache« Methode verstehen, Steuerelemente zu binden, was nur kurz erklärt wird, weil Sie in der Visual-Basic-Dokumentation relativ viele Informationen darüber finden.

Anschließend werden wir die »harten Bandagen« betrachten, die Datenbindung auszuführen, wobei Sie von Visual Basic übernehmen und es selbst machen. Dieses Thema wird ebenfalls nur kurz gestreift, weil es einfach zu umfangreich ist, um hier in Gänze erklärt werden zu können. Ich versuche nur, Ihnen die richtige Richtung zu weisen. Wenn Sie DAO-Experte (Data Access Objects) sind, sollten

Sie keine Probleme damit haben, das hier vorgestellte einfache Beispiel zu einem komplexen Datenverwaltungssystem zu machen – falls Sie das möchten. Wenn Sie kein DAO-Experte sind, werden Sie mir irgendwann nicht mehr folgen können, weil ich mich nicht bemüht habe, die Feinheiten des Datenzugriffs unter Visual Basic zu erklären. Das ist eine andere Geschichte.

Visual Basic 6 erlaubt Ihnen, eigene Datenquellen zu erzeugen – und damit letztlich, Ihre eigenen Daten-Steuerelemente zu entwickeln. Das ist ebenfalls ein sehr umfangreiches Thema, das den Rahmen dieses Buchs sprengen würde. Hoffentlich schreibt irgend jemand ein gutes Buch, das sich insbesondere mit dem Datenzugriff mit Visual Basic beschäftigt, und widmet diesem Thema die Ausführlichkeit, die es verdient.

19.6.1 Die einfache Methode

Die einfache Methode, die Datenbindung für eine Eigenschaft in einem Steuerelement vorzunehmen, ist, es Visual Basic zu überlassen. Dazu definieren Sie eine Eigenschaft und setzen ihre Attribute im Dialogfeld EXTRAS/PROZEDURATTRIBUTE. Es gibt vier Möglichkeiten für die Datenbindung:

- DATENGEBUNDENE EIGENSCHAFT. Markieren Sie dieses Kontrollkästchen, um die Eigenschaft zu binden.
- AN DATEN-FELD GEBUNDEN. Diese Eigenschaft wird zum Standard für die gebundene Eigenschaft, falls das Steuerelement mehrere gebundene Eigenschaften hat.
- ZUR ENTWURFSZEIT IN DATABINDINGS-AUFLISTUNG ANZEIGEN. Erlaubt dem Entwickler, der Ihr Steuerelement verwendet, die Datenbindung zur Entwurfszeit zu bearbeiten.
- CANPROPERTYCHANGE VOR ÄNDERUNG AUFRUFEN. Diese Option informiert den Container darüber, daß Ihr Steuerelement `CanPropertyChange` aufruft, bevor gebundene Eigenschaftswerte geändert werden.

Um das Ganze besser zu verstehen, müssen Sie wissen, was passiert, wenn eine Eigenschaft gebunden wird. Visual Basic fügt dem `Extender`-Objekt des Steuerelements vier Eigenschaften hinzu:

- Die `DataSource`-Eigenschaft wird vom Entwickler verwendet, um ein Datensteuerelement auszuwählen, zu dem Ihr Steuerelement gebunden wird.
- Die `DataField`-Eigenschaft wird vom Entwickler genutzt, um ein Feld in der Datenbank auszuwählen, zu dem eine der Eigenschaften Ihres Steuerelements gebunden wird (diejenige, für die »AN DATEN-FELD GEBUNDEN« gesetzt ist. Sie sollten dieses Kontrollkästchen immer für mindestens eine Eigenschaft markieren).

- Die `DataBindings`-Eigenschaft erscheint im VB-Fenster, um dem Entwickler zu erlauben, andere gebundene Eigenschaften in Ihrem Steuerelement zu verschiedenen Feldern der Datenbank zu binden. Alle gebundenen Steuerelemente, deren Prozedurattribute »ZUR ENTWURFSZEIT IN DATABINDINGS-AUFLISTUNG ANZEIGEN« angeben, erscheinen im DATABINDINGS-Dialogfeld. Das `Extender`-Objekt des Steuerelements stellt außerdem eine `DataBindings`-Auflistung bereit, die ein `DataBinding`-Objekt für jede gebundene Eigenschaft enthält. Dieses Objekt enthält die Bindungsinformationen für die Eigenschaft.
- Die `DataChanged`-Eigenschaft gibt an, ob sich eine der gebundenen Eigenschaften geändert hat.

Beachten Sie unbedingt, daß die Datenbindung vom Container unterstützt wird. Die Container sind jedoch nicht gezwungen, die Datenbindung zu unterstützten, und nicht alle, die das überhaupt tun, unterstützen auch die Bindung von mehr als einer Eigenschaft in einem Steuerelement.

Abbildung 19.3 zeigt die Architektur der einfachen Datenbindung. Die `DataSource`-Eigenschaft des Steuerelements wählt ein Datensteuerelement aus, das die Datenbank angibt, die Quelltabelle oder das Dynaset, und das zur Laufzeit bestimmt, auf welche Zeile im Recordset zugegriffen wird. Das `DataBindings`-Objekt bindet einzelne Eigenschaften innerhalb des Steuerelements zu den verschiedenen Feldern des Recordsets. Sie bestimmen eine der Eigenschaft als Standard für die gebundenen Eigenschaften, indem Sie das Attribut »AN DATEN-FELD GEBUNDEN« setzen. Das Feld, zu dem sie gebunden wird, wird durch die DataField-Eigenschaft des Steuerelements angegeben.

Das Steuerelement `ch19bind.ctl` im Projekt `Binding.vbp` demonstriert die einfache Bindung von zwei Steuerelementeigenschaften zu einer Datenbank. Für dieses Beispiel wurde eine einfache Datenbank bereitgestellt, `dbdemo.mdb`. Sie enthält eine einzige Tabelle mit mehreren Feldern. Zwei davon sind hier interessant: `show`, das den Namen einer TV-Show enthält, und `rating`, das eine kurze Auswertung der Show enthält.

Das Steuerelement implementiert mehrere Standardeigenschaften, die in den folgenden Listings nicht gezeigt werden. Es enthält zwei Eigenschaften, die beide gebunden werden, eine zum Feld `show` und eine zum Feld `rating` der Tabelle. Beachten Sie, daß die Bindung zur Entwurfszeit durch den Entwickler stattfindet, und nicht durch Sie als Autor des Steuerelements. Jede dieser Eigenschaften könnte auch zu anderen Feldern in der Datenbank gebunden werden. Die Eigenschaften werden zwei Textfeldern zugewiesen, `txtShow` und `txtRating`. Der Code für ihre Eigenschaftsprozeduren sieht wie folgt aus:

```
Public Property Get ShowInfo() As String
   ShowInfo = txtShow.Text
End Property
```

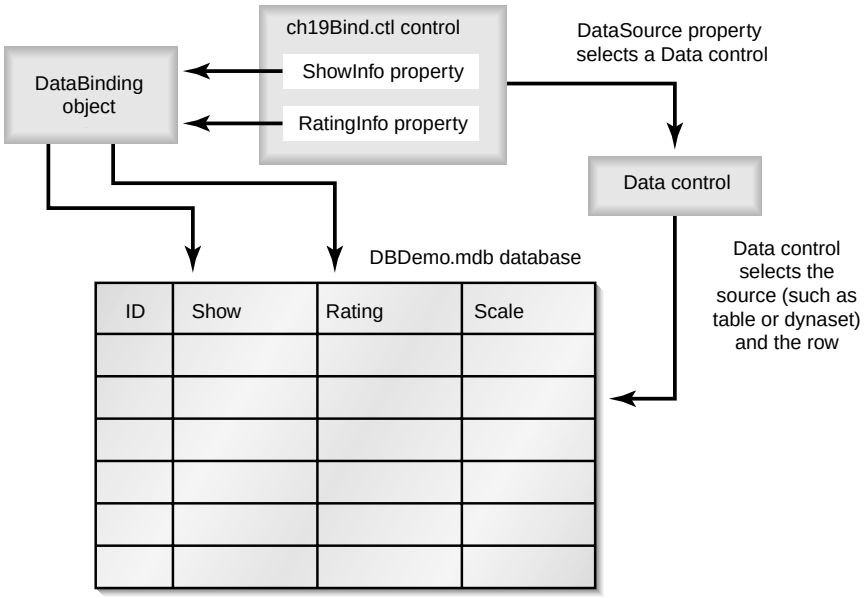

Abb. 19.3: Architektur der einfachen Datenbindung.

```
Public Property Let ShowInfo(ByVal New_ShowInfo As String)
   If UserControl.CanPropertyChange("ShowInfo") Then
      txtShow.Text() = New_ShowInfo
      PropertyChanged "ShowInfo"
   End If
End Property

Public Property Get RatingInfo() As String
   RatingInfo = txtRating.Text
End Property

Public Property Let RatingInfo(ByVal New_RatingInfo As String)
   If UserControl.CanPropertyChange("RatingInfo") Then
      txtRating.Text() = New_RatingInfo
      PropertyChanged "RatingInfo"
   End If
End Property
```

Beachten Sie, daß der Wert der Eigenschaft nur gesetzt werden kann, wenn die Funktion `CanPropertyChange` `True` zurückgibt. Das ist erforderlich, weil in den Prozedurattributen für beide Eigenschaften das Kontrollkästchen »CANPROPERTYCHANGE VOR ÄNDERUNG AUFRUFEN« markiert ist. Diese Option wird von VB zu diesem Zeitpunkt nicht verwendet – `CanPropertyChange` gibt immer `True`

zurück. Aber wenn Sie `CanPropertyChange` immer aufrufen, gehen Sie sicher, daß Ihr Steuerelement auch in zukünftigen Containern (sowohl in VB als auch in anderen) das richtige Verhalten beibehält.

Der Aufruf der Funktion `PropertyChanged` ist vor allem wichtig, wenn Sie die Datenbindung einsetzen. Sie wissen, daß `PropertyChanged` Visual Basic darüber informiert, daß eine Eigenschaft geändert wurde, so daß dieses erkennt, ob die Eigenschaft gespeichert werden soll. Außerdem wird sie genutzt, um Visual Basic darüber zu informieren, daß sich der Wert einer gebundenen Eigenschaft geändert hat, so daß es erkennt, daß dieser in die Datenbank geschrieben werden soll.

Wenn Sie möchten, daß Datenbankfelder bearbeitet werden können, müssen Sie Visual Basic darüber informieren, daß sie sich geändert haben. Das passiert wie folgt:

```
Private Sub txtRating_Change()
   PropertyChanged "RatingInfo"
End Sub

Private Sub txtShow_Change()
   PropertyChanged "ShowInfo"
End Sub
```

Wechselt das Datensteuerelement zu einer anderen Eigenschaft, greift VB auf die `DataChanged`-Eigenschaft des Steuerelements zu (oder auf die `DataChanged`-Eigenschaft für jedes `DataBinding`-Objekt). Ist diese Eigenschaft gleich `True`, versucht VB, die aktualisierten Informationen in die Datenbank zu schreiben.

Falls Sie den Schnittstellen-Assistenten für ActiveX-Steuerelemente verwendet haben, um die gebundenen Eigenschaften zu erzeugen, vergessen Sie nicht, in der `Property Let`-Prozedur auf `CanPropertyChange` zu testen, und die Eigenschaftspersistenz aus den `ReadProperties`- und `WriteProperties`-Ereignissen von `UserControl` zu entfernen. Es ist sinnlos, eine Eigenschaft persistent zu halten, die zu einer Datenbank gebunden ist.

Wenn ein Steuerelement mehrere gebundene Eigenschaften enthält, kann es Möglichkeiten dafür definieren, mit dem Steuerelement zu interagieren. Beispielsweise könnten Sie eine Eigenschaft mit einem Vorgabewert initialisieren, der von einer anderen Eigenschaft abhängig ist.

19.6.2 Die harte Methode

Was tun Sie, wenn Ihr Steuerelement eine gebundene Liste enthalten soll? Oder mehrere Listen aus verschiedenen Tabellen? Sie können den Ansatz hier erweitern, indem Sie direkt über das `Extender`-Objekt auf die Datenquelle zugreifen:

```
Private Sub UserControl_Show()
   Dim obj As Object
   If Ambient.UserMode Then
```

```
      Set m_MyData = Extender.DataSource
      Set m_MyRecordSet = m_MyData.Recordset
      LoadCombo
   End If
End Sub
```

Nachdem Sie Zugriff auf eine Datenquelle und ein Recordset haben, können Sie das Recordset ganz einfach durchlaufen und eine Zeile oder ein Feld laden. Wenn Ihr Steuerelement die komplexe Datenbindung einsetzt, können Sie Felder angeben, die unter Verwendung der `DataBindings`-Eigenschaft zu verschiedenen Quellen gebunden sind. Anschließend lesen Sie jede dieser Datenquellen im Code.

Oder Sie umgehen die Datenbindung von Visual Basic vollständig. Beachten Sie, daß Ihr ActiveX-Steuerelement vollständigen Zugriff auf alle Datenbankfunktionen von Visual Basic hat. Mit anderen Worten, es ist möglich, die Datenbindung auch selbst zu implementieren. Sie ist vielleicht nicht so sauber wie die von Visual Basic bereitgestellte, aber sie kann extrem leistungsfähig sein.

Das Projekt `Binding.vbp` enthält ein zweites Steuerelement, `ch19bnd2.ctl`, das eine eigene Datenbindung implementiert. Das hier gezeigte Beispiel demonstriert nur skizzenhaft die Implementierung und soll Ihnen nur kurz zeigen, wie die Standard-Datenbankoperationen ausgeführt werden können. Das Steuerelement enthält ein Kombinationsfeld und ein Textfeld. Im Kombinationsfeld wird die »show«-Information ausgewählt, und das Textfeld zeigt die »rating«-Information an und erlaubt ihre Bearbeitung.

Als erstes muß sich das Steuerelement darum kümmern, einen Verweis auf ein `Database`-Objekt zu erhalten. Dazu gibt es mehrere Möglichkeiten. Sie können eine Eigenschaft einfügen, die den Datenbanknamen und den Datensatzquellnamen enthält, wodurch Sie letztlich die vom Datensteuerelement verwendeten Eigenschaften kopieren, um eine Datenbank und eine Datensatzquelle auszuwählen.

Oder Sie machen es einfach und tricky. Dieses Beispiel implementiert eine `DataSource`-Eigenschaft, wobei es sich um einen String handelt, der den Namen eines Datensteuerelements auf demselben Container wie das Steuerelement angibt. Die Eigenschaft wird wie folgt implementiert:

```
Dim m_DataSource As String

Public Property Get DataSource() As String
   DataSource = m_DataSource
End Property

Public Property Let DataSource(ByVal vNewValue As String)
   m_DataSource = vNewValue
   PropertyChanged "DataSource"
End Property
```

```
Private Sub UserControl_ReadProperties(PropBag As PropertyBag)
   m_DataSource = PropBag.ReadProperty("DataSource", "")
End Sub

Private Sub UserControl_WriteProperties(PropBag As PropertyBag)
   PropBag.WriteProperty "DataSource", m_DataSource, ""
End Sub
```

Die Eigenschaft wird auf die übliche Weise mit den `ReadProperties`- und `WriteProperties`-Ereignissen des `UserControl`-Objekts persistent gemacht. Der Nachteil bei diesem Ansatz ist, daß Sie kein Dropdown-Listenfeld anzeigen können, das die Namen der bereits auf dem Container vorhandenen Datensteuerelemente anzeigt (das normale Verhalten der `DataSource`-Eigenschaft). Damit gibt es mehrere Möglichkeiten:

- Sie können es dem Entwickler überlassen, den Namen direkt einzugeben (vielleicht auch gleich mit einer Auswertung, um sicherzustellen, daß er mit einem Wert für ein Steuerelement übereinstimmt, was hier nicht gezeigt ist).
- Sie können direkte Eingaben verhindern und vielleicht eine Eigenschaftsseite verwenden, um dem Benutzer eine Auswahl aus den Steuerelementen auf dem Container zu ermöglichen.
- Sie können SpyWorks von Desaware einsetzen, um ein benutzerdefiniertes Dropdown-Listenfeld zu erzeugen, das die Namen der verfügbaren Datensteuerelemente enthält, und so das Standardverhalten der `DataSource`-Eigenschaft nachbildet.
- Sie können eine separate gebundene Eigenschaft implementieren (was bewirkt, daß die DataSource-Eigenschaft für das Steuerelement definiert wird). Anschließend können Sie diese `DataSource`-Eigenschaft zur Entwurfszeit des Containers lesen und sie in ihrer eigenen internen Variablen persistent machen. Dieser Schritt ist erforderlich, weil Sie die `DataSource`-Eigenschaft von `Extender` zur Laufzeit nicht lesen können.

Ich muß hier betonen, daß die `DataSource`-Eigenschaft, die in den ersten drei Optionen implementiert wird, dem Entwickler ähnlich wie die standardmäßige `DataSource`-Eigenschaft erscheint und auch genauso funktioniert. Aber sie wird anders implementiert. In dem früher gezeigten Beispiel `ch19Bind.ctl` ist die `DataSource`-Eigenschaft Teil des `Extender`-Objekts des Steuerelements und wird von dem Container implementiert (wie jede `Extender`-Eigenschaft). In diesem Beispiel ist die `DataSource`-Eigenschaft Teil der öffentlichen Schnittstelle Ihres Steuerelements und wird von Ihnen implementiert, dem Autor des Steuerelements.

Die `DataSource`-Eigenschaft teilt Ihnen den Namen des Datensteuerelements mit. Aber wie erhalten Sie einen Verweis auf das Steuerelement? Eine Möglichkeit sehen Sie im folgenden Code:

```
Dim m_MyRecordSet As RecordSet
Dim m_MyData As Data

Private Sub UserControl_Show()
   Dim obj As Object
   If Ambient.UserMode Then
      For Each obj In Extender.Parent.Controls
         If TypeOf obj Is Data Then
            If obj.Name = m_DataSource Then
               Set m_MyData = obj
               Set m_MyRecordSet = m_MyData.RecordSet
               LoadCombo
           End If
         End If
      Next
   End If
End Sub
```

Dieser Code zeigt, wie Sie mit Hilfe der Steuerelementauflistung des Containers, auf die über die Extender-Eigenschaft zugegriffen wird, nach einem bestimmten Steuerelement suchen.

Diese Funktion wird während des `Show`-Ereignisses aufgerufen, während dessen sicher vorausgesetzt werden kann (wenigstens bei Visual Basic), daß alle anderen Steuerelemente, die zu dem Container gehören, bereits positioniert wurden.

Damit diese Funktion für die Verwendung in einem realen Steuerelement robust wird, müssen zwei Dinge getan werden:

- Führen Sie eine Fehlerprüfung ein. Beachten Sie, daß Sie nicht voraussetzen können, daß jedes `Extender`-Objekt eine `Parent`-Eigenschaft hat, oder daß der Container eine Steuerelementeauflistung unterstützt.
- Führen Sie eine zusätzliche Logik ein, um festzulegen, ob Sie das Kombinationsfeld bei jeder Anzeige des Steuerelements neu laden wollen.

Das Kombinationsfeld kann mit Hilfe einer Standard-Datenbanktechnik geladen werden, wie folgt:

```
Public Sub LoadCombo()
   m_MyRecordSet.MoveFirst
   Do While Not m_MyRecordSet.EOF
      Combo1.AddItem m_MyRecordSet.Fields("Show")
      m_MyRecordSet.MoveNext
   Loop
   Combo1.ListIndex = 0
End Sub
```

Auch hier sollten Sie jede erforderliche Fehlerprüfung einfügen.

In diesem Beispiel ist die `DataField`-Information fest im Code vorgegeben. Sie können offensichtlich selbst eine oder mehrere `DataField`-Eigenschaften angeben, oder Ihre eigene `DataBindings`-Typauflistung implementieren, mit eigener Eigenschaftsseite, um eine komplexere Bindung zu realisieren. Das könnte dem Entwickler erlauben, andere Felder für die verschiedenen gebundenen Objekte im Steuerelement anzugeben.

Das »rating«-Textfeld wird immer dann geladen, wenn im Kombinationsfeld eine Auswahl getroffen wird. Das wird durch den folgenden Code realisiert:

```
Dim m_RatingChanged As Boolean

Private Sub Combo1_Click()
   If m_RatingChanged Then
      m_MyRecordSet.Edit
      m_MyRecordSet.Fields("rating") = Text1.Text
      m_MyRecordSet.Update
   End If
   m_MyRecordSet.FindFirst "[Show] = '" & Combo1.Text & "'"
   Text1.Text = m_MyRecordSet.Fields("rating")
   m_RatingChanged = False
End Sub
```

Auch hier ist die Feldinformation fest im Code vorgegeben. Die Variable `m_RatingChanged` ist eine private Boolesche Variable, die anzeigt, ob das Textfeld `Text1` geändert wurde. Diese Variable könnte einfach als öffentliche Eigenschaft bereitgestellt werden, um eine identische Funktionalität wie die Standardeigenschaft `DataChanged` bereitzustellen. Wenn die Auswahl im Kombinationsfeld geändert wird, wie beispielsweise durch das `Click`-Ereignis angezeigt, prüft das Steuerelement, ob sich das Textfeld geändert hat. Ist dies der Fall, aktualisiert es das Feld in der Datenbank. Anschließend sucht es den ausgewählten Datensatz und setzt den RecordSet-Zeiger mit Hilfe der `FindFirst`-Methode auf diese Position. Danach lädt es das Textfeld `Text1` mit dem »rating«-Feld für diesen Datensatz und markiert Text1 als »sauber«. Bleibt nur noch der Code, der das Textfeld `Text1` als geändert markiert:

```
Private Sub Text1_Change()
   m_RatingChanged = True
End Sub
```

Interessant bei diesem Beispiel ist, daß das Steuerelement letztlich Datenbankfelder bindet (in diesem Fall zu Standardelementen, aber es könnten auch Variablen verwendet werden), aber es bindet sie nicht zu den öffentlichen Eigenschaften des Steuerelements! Könnten Sie öffentliche Eigenschaften definieren, die als gebunden agieren? Natürlich – lassen Sie sie nur die entsprechenden Eigenschaften der Standardelemente setzen, wie Sie es im vorigen Beispiel gesehen haben.

Weil wir die Bindung von Visual Basic nicht verwenden, müssen wir uns nicht darum kümmern, `PropertyChange` oder `CanPropertyChange` für Änderungen in der Datenbank aufzurufen. Es wäre aber nicht schlimm, würden Sie gebundene öffentliche Eigenschaften auf diese Weise erzeugen.

Damit schließen wir unsere kurze Einführung in die Möglichkeiten der Do-it-yourself-Datenbindung mit Hilfe der ActiveX-Steuerelemente unter Visual Basic. Der Autor hat dadurch mehr Arbeit, aber er kann sehr viel mehr Funktionalität bereitstellen und die Abhängigkeiten vom Container reduzieren. Wenn die vom Container unterstützte Datenbindung für Sie ausreichend ist, dann sollten Sie sie auf alle Fälle nutzen.

Sie kennen nun schon fast alle Grundlagen der ActiveX-Steuerelemente. Bleibt noch ein wichtiges Thema. Die Eigenschaftsseiten wurden nur kurz vorgestellt; jetzt soll ihre Verwendung demonstriert werden.