Kapitel 18

Extender- und Ambient-Objekte

Wenn ich alles richtig gemacht habe, dann sollten Sie jetzt schon erkennen können, wie sich ActiveX-Steuerelemente zusammensetzen. Sie sollten verstehen, in welchen verschiedenen Statuszuständen sich ein Steuerelement befinden kann. Sie sollten mit den wichtigsten Ereignissen im Leben eines Steuerelements vertraut sein und außerdem einige der gebräuchlichsten Methoden und Eigenschaften kennen.

Kapitel 18 beschäftigt sich hauptsächlich damit, wie man Eigenschaften und Ereignisse in einem Steuerelement definiert, was Sie für die Implementierung Ihrer Steuerelemente notwendig brauchen. Sie können sich dieses Kapitel auch wie eine ausführliche Einführung für das nächste vorstellen. Es handelt sich dabei um ein Kapitel, in dem wir die restlichen Bausteine für ein Steuerelement vorstellen. Denn wenn das `UserControl`-Objekt den Kern Ihres ActiveX-Steuerelements bildet, dann definieren die `Extender`- und `Ambient`-Objekte zweifellos die Umgebung, in der es lebt.

18.1 Das Extender-Objekt

Die meisten Eigenschaften des `Extender`-Objekts sind Ihnen bereits vertraut. Sie finden sie in fast jedem Formular oder Steuerelement von Visual Basic. Dieser Abschnitt soll Ihnen nicht erklären, wie diese Eigenschaften aus der Sicht des Entwicklers verwendet werden, sondern wie Sie als Autor damit umgehen.

Versuchen Sie, ein neues Steuerelement anzulegen, schließen Sie den Designer und fügen Sie es auf einem neuen Formular ein. Definieren Sie noch keine Eigenschaften – dies ist ein völlig leeres Steuerelement.

Wenn Sie die Eigenschaften betrachten, die das Steuerelement zur Entwurfszeit des Containers abhängig von den Eigenschaftswerten des `UserControl`-Objekts hat, sehen Sie die in der folgenden Tabelle gezeigten Einstellungen:

Extender-Eigenschaften für ein leeres ActiveX-Steuerelement	**Kommentar**
Name	Standard
Align	Erscheint nur, wenn die `Alignable`-Eigenschaft des Steuerelements `True` ist
Cancel	Standard – erscheint nur, wenn die `DefaultCancel`-Eigenschaft des Steuerelement `True` ist
CausesValidation	Standard

Extender-Eigenschaften für ein leeres ActiveX-Steuerelement	**Kommentar**
DataBinding DataBindings DataChanged DataField DataFormat DataMember DataSource	Erscheinen nur, wenn das Steuerelement eine Eigenschaft hat, die an eine Datenbank gebunden ist (siehe Kapitel 19)
Default	Standard – Erscheint nur, wenn die `DefaultCancel`-Eigenschaft des Steuerelements `True` ist
DragIcon	
DragMode	
Enabled	Erscheint nur, wenn das Steuerelement eine `Enabled`-Eigenschaft hat – mehr darüber später
Height	
HelpContextId	Erscheint nur, wenn die `CanGetFocus`-Eigenschaft des Steuerelements `True` ist
Index	
Left	Standard
Negotiate	Erscheint nur, wenn die `Alignable`-Eigenschaft des Steuerelements `True` ist
TabIndex	Erscheint nur, wenn die `CanGetFocus`-Eigenschaft des Steuerelements `True` ist
Tabstop	Erscheint nur, wenn die `CanGetFocus`-Eigenschaft des Steuerelements `True` ist
Tag	
ToolTipText	
Top	
Visible	Standard
WhatsThisHelpID	
Width	Standard

Beachten Sie zu dieser Tabelle die folgenden Punkte:

- Die Eigenschaften, die mit dem Kommentar »Standard« versehen sind, werden von den meisten Containern unterstützt.

- Wenn die `InvisibleAtRuntime`-Eigenschaft des Steuerelements `True` ist, stehen nur die `Extender`-Eigenschaften `Name`, `Index`, `Left`, `Top` und `Tag` zur Verfügung.
- Visual Basic kümmert sich automatisch um die Persistenz dieser Entwurfszeit-Eigenschaften (sie werden zusammen mit Ihrem Projekt gespeichert).

Neben diesen Eigenschaften stellt Visual Basic die folgenden `Extender`-Eigenschaften bereit, auf die der Entwickler zur Laufzeit zugreifen kann:

Extender-Eigenschaften, Methoden und Ereignisse, die nur zur Laufzeit zur Verfügung stehen	**Kommentare**
`Container`-Eigenschaft	Der Container (in der Regel ein Formular), in dem das Steuerelement enthalten ist
`Object`-Eigenschaft	Bezieht sich auf das zugrundeliegende Objekt, nicht auf den Extender
`Parent`-Eigenschaft	Standard – der Container des Steuerelements. Das `UserControl`-Objekt bietet außerdem eine `Parent`-Eigenschaft, die immer zur Verfügung steht, unabhängig vom Container.
`Drag`-Methode	
`Move`-Methode	
`SetFocus`-Methode	
`ShowWhatsThis`-Methode	
`Zorder`-Methode	
`DragDrop`-Ereignis	
`GotFocus`-Ereignis	Erscheint nur, wenn die `CanGetFocus`-Eigenschaft des Steuerelements `True` ist
`LostFocus`-Ereignis	Erscheint nur, wenn die `CanGetFocus`-Eigenschaft des Steuerelements `True` ist
`DragOver`-Ereignis	
`ObjectEvent`-Ereignis	Teil eines `VBControlExtender`-Objekts (siehe Kapitel 11)
`Validate`-Ereignis	

Aus der Sicht es Entwicklers (das ist die Person, die Ihr Steuerelement einsetzt) besteht kein Unterschied zwischen Steuerelementeigenschaften und `Extender`-Eigenschaften. Sie erscheinen als einzige Oberfläche. Wie ist das möglich? Wenn Sie als Entwickler auf ein Steuerelement zugreifen, dann greifen Sie eigentlich

auf den `Extender` des Steuerelements zu. Das `Extender`-Objekt reflektiert die Eigenschaften und Methoden des Objekts selbst. Diesen Sachverhalt sehen Sie in Abbildung 18.1.

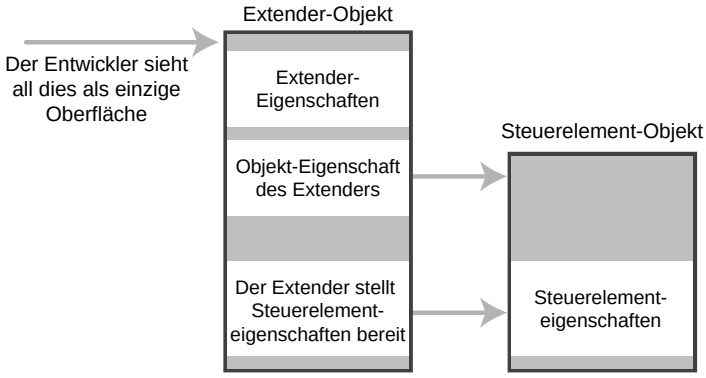

Abb. 18.1: Die Beziehung zwischen Steuerelement und Extender.

Für das `Extender`-Objekt sind einige Dinge zu beachten:

- Eigenschaften, Methoden und Ereignisse des `Extender`-Objekts werden vom Container bereitgestellt. Sie können nicht voraussetzen, daß jede Eigenschaft, Methode oder jedes Ereignis für jedes `Extender`-Objekt vorhanden sind, weil das vom jeweiligen Container abhängig ist.
- Weil das `Extender`-Objekt sich abhängig vom Container ändern kann, erfolgt der gesamte Zugriff auf `Extender`-Eigenschaften und Methoden von Ihrem Steuerelement aus spät gebunden.
- Wenn ein Entwickler auf Ihr Steuerelement verweist, verweist er eigentlich auf das `Extender`-Objekt für Ihr Steuerelement. Über die `Object`-Eigenschaft (falls diese existiert) kann er direkt auf das Steuerelement-Objekt zugreifen.

Die Programmgruppe `ch18tst1.vbg` enthält zwei Projekte: Das Projekt `ch18ctls.vbp` demonstriert anhand mehrerer Steuerelemente die in diesem Kapitel beschriebenen Techniken, und das Projekt `ch18tst1.vbp` ist ein Testprogramm, das diese Steuerelemente einsetzt.

Das Steuerelement `ch18ctla` ist völlig leer, bis auf zwei Funktionen, die Ihnen den Zugriff auf die internen `Extender`- und `Me`-Objekte des Steuerelements erlauben. Sie verwenden den folgenden Code:

```
Public Function InternalExtender() As Object
   Set InternalExtender = UserControl.Extender
End Function
```

```
Public Function InternalMe() As Object
   Set InternalMe = Me
End Function
```

Sie sollten die Objekte in der Regel nicht auf diese Weise bereitstellen, aber für Testzwecke ist das ganz praktisch. Legen Sie das Steuerelement auf einem Formular an, fügen Sie eine Schaltfläche ein, und geben Sie für Ihr `Click`-Ereignis den folgenden Code ein:

```
Private Sub Command1_Click()
   If ch18CtlA1.Container Is ch18CtlA1.Parent Then
      Debug.Print "Parent und Container sind gleich"
   Else
      Debug.Print "Parent und Container sind nicht gleich"
   End If
   If ch18CtlA1 Is ch18CtlA1.Object Then
      Debug.Print "Steuerelement und Steuerelement-Objekt sind gleich"
   Else
      Debug.Print "Steuerelement und Steuerelement-Objekt sind nicht gleich"
   End If
   If ch18CtlA1.Object Is ch18CtlA1.InternalMe Then
      Debug.Print "Object-Eigenschaft des Extenders ist das Me des _
Steuerelements"
   Else
      Debug.Print " Object-Eigenschaft des Extenders ist nicht das Me des _
Steuerelements "
   End If

   If ch18CtlA1 Is ch18CtlA1.InternalExtender Then
      Debug.Print "Steuerelement ist der Extender des Steuerelements"
   Else
      Debug.Print " Steuerelement ist nicht der Extender des Steuerelements "
   End If
End Sub
```

Daraus entstehen die folgenden Ergebnisse:

```
Parent und Container sind gleich
Steuerelement und Steuerelement-Objekt sind nicht gleich
Object-Eigenschaft des Extenders ist das Me des Steuerelements
Steuerelement ist der Extender des Steuerelements
```

Daraus gelangen wir zu den in Tabelle 18.1 gezeigten Ergebnissen. Sie zeigen, wie der Autor oder der Entwickler auf die verschiedenen Eigenschaften des Steuerelements zugreift.

Objekt	Zugriff als Entwickler	Zugriff als Steuerelement-Autor
Steuerelement `myctl` Extender	myctl	`UserControl.Extender`
Steuerelement `myctl` Objekt selbst (nicht der Extender)	`myctl.Object`	`Me`

Container für das Steuerelement »myctls”	`myctl.Parent`
	`myctl.Container`
		`UserControl.Extender.Container`
		`UserControl.Extender.Parent`
		`UserControl.Parent`

Tab. 18.1: Navigation zwischen Steuerelement, Container und Extender

18.1.1 Der Einfluß der Regeln für den Gültigkeitsbereich

Mit den allgemeinen Regeln für den Gültigkeitsbereich von Variablen, Eigenschaften und Methoden sind Sie bereits vertraut. Wenn Sie ein Element mit demselben Namen wie ein Element auf einer höheren Ebene anlegen, wird dieses andere Element dadurch überdeckt. Wenn Sie also eine globale Variable `x` haben und in einer Prozedur `x` definieren, wird die globale Variable dadurch überdeckt, bis die Prozedur beendet wird.

Dasselbe gilt für Steuerelementeigenschaften in bezug auf `Extender`-Eigenschaften. Angenommen, Sie fügen Ihrem Steuerelement eine `Tag`-Eigenschaft hinzu. Der VB-Extender verwaltet standardmäßig bereits eine `Tag`-Eigenschaft. Weil der Entwickler auf den `Extender` zugreift, hat diese benutzerdefinierte Eigenschaft Vorrang. Diesen Sachverhalt sehen Sie im Projekt `ch18tst1.vbp` mit dem Steuerelement `ch18ctlb.ctl`. Dieses Steuerelement implementiert seine eigene `Tag`-Eigenschaft und Elementvariable unter Verwendung des folgenden Codes:

```
Public Property Get Tag() As String
        Tag = m_Tag
    End Property

    Public Property Let Tag(ByVal vNewValue As String)
       m_Tag = vNewValue
       Debug.Print "Internes Tag setzen"
       Report
    End Property

    Public Sub Report()
       Debug.Print "Interne Tag-Variable setzt" & m_Tag
       Debug.Print "Interne Tag-Eigenschaft setzt" & Tag
       Debug.Print "Extender Tag setzt " & Extender.Tag
    End Sub
```

Der Textcode im Formular frmch18test1 wird einer zweiten Schaltfläche zugeordnet:

```
Private Sub Command2_Click()
   ch18CtlB1.Tag = "Tag-Eigenschaft "
   ch18CtlB1.Object.Tag = "Objekt-Tag-Eigenschaft "
End Sub
```

Daraus entstehen die folgenden Ergebnisse:

```
Internes Tag setzen
Interne Tag-Variable setzt Objekt-Tag-Eigenschaft
Interne Tag-Eigenschaft setzt Objekt-Tag-Eigenschaft
Extender Tag setzt Tag-Eigenschaft
```

Daraus können wir schließen, daß es zwei separate Tag-Variablen gibt – eine wird vom Steuerelement implementiert, die andere vom Container. Der Zugriff auf diese Variablen erfolgt wie in Tabelle 18.2 gezeigt.

Tag, auf das der Zugriff erfolgt	Zugriff als Entwickler	Zugriff als Autor
Extender-Tag-Eigenschaft, die von Visual Basic verwaltet wird	ch18Ctlb1.Tag	UserControl.Extender.Tag
Interne Tag-Eigenschaft, die von Ihrem Steuerelement verwaltet wird	ch18CtlB1.Object.Tag	Tag Me.Tag

Tab. 18.2: Beispiel für den Gültigkeitsbereich der Tag-Eigenschaft für ch18CtlB1

Warum sollten Sie eine Eigenschaft wie Tag definieren, die bereits vom Container implementiert wird? Weil Sie sicherstellen möchten, daß alle, die Ihr Steuerelement verwenden, Zugriff auf eine Tag-Eigenschaft für das Steuerelement haben, und Sie nicht davon ausgehen können, daß jeder Container, in dem das Steuerelement ausgeführt wird, eine Tag-Eigenschaft für seinen Extender unterstützt.

18.1.2 Zugriff auf Extender-Eigenschaften

Der größte Haken bei der Verwendung von Extender-Eigenschaften ist, daß Sie nie sicher sein können, ob sie wirklich zur Verfügung stehen. Wenn Sie nicht möchten, daß Ihr Steuerelement auf Visual Basic oder ganz bestimmte andere Container eingeschränkt ist, müssen Sie also Vorsichtsmaßnahmen ergreifen, wenn Sie auf Extender-Eigenschaften zugreifen, so daß Ihr Steuerelement sich korrekt verhält. Es gibt zwei Ansätze dafür:

- Sie setzen für jeden Zugriff auf eine `Extender`-Eigenschaft oder Methode eine Fehlerverarbeitung ein.
- Sie prüfen explizit, ob eine `Extender`-Eigenschaft vorhanden ist, bevor Sie sie aufrufen.

Der Ansatz mit der Fehlerverarbeitung ist klar. Der zweite Ansatz kann auf zweierlei Arten realisiert werden. Sie können eine Routine anlegen, die versucht, auf eine Eigenschaft zuzugreifen, und die die Ergebnisse aufzeichnet. Der folgende Code aus dem Steuerelement `ch18ctlb.ctl` zeigt, wie das bewerkstelligt wird:

```
Private TagIsPresent As Boolean
Private xyzIsPresent As Boolean
Private zzzispresent As Boolean
Private Sub CheckExtender()
   Dim testnumber As Integer
   Dim v As Variant
   TagIsPresent = True
   xyzIsPresent = True
   zzzispresent = True
   On Error GoTo itp1
   testnumber = 0
   v = Extender.Tag
   testnumber = 1
   v = Extender.xyz
   testnumber = 2
   v = Extender.zzz
   Exit Sub
itp1:
   Select Case testnumber
      Case 0
         TagIsPresent = False
      Case 1
         xyzIsPresent = False
      Case 2
         zzzispresent = False
   End Select
   Resume Next
End Sub
```

Die Routine prüft einfach nacheinander die gewünschten Eigenschaften und setzt auf Modulebene ein Flag für diejenigen, die nicht vorhanden sind. Eine einfachere Version dieser Routine könnte ein Boolesches Feld verwenden, wobei jeder Eintrag einer bestimmten `Extender`-Eigenschaft entspricht.

Ein noch einfacherer Ansatz verwendet die DLL `apigid32.dll`, die Sie zusammen mit diesem Buch erhalten. Diese DLL wurde ursprünglich für *The Visual Basic Programmer's Guide to the Win32 API* geschrieben und enthält zahlreiche sehr praktische Low-Level-Funktionen, unter anderem agIsValidName. Die Funktion ist wie folgt deklariert:

```
Declare Function agIsValidName& Lib "apigid32.dll" (ByVal o As Object,
ByVal_ lpname$)
```

Ihre Verwendung sehen Sie im folgenden Code aus dem Steuerelement `ch18ctlb.ctl`:

```
    Debug.Print "Tag property present: " & agIsValidName(Extender,
"Tag")
    Debug.Print "xyz property present: " & agIsValidName(Extender,
"xyz")
```

Diese Funktion ist intern ganz einfach. Sie ruft nur die Methode `GetIDsOfNames` auf der Dispatch-Schnittstelle des Objekts auf, um zu prüfen, ob sie einen zulässigen Rückgabewert erhält. Wie Sie aus Teil I dieses Buchs wissen, ist es unter Visual Basic erforderlich, daß alle Schnittstellen entweder Dispatch-Schnittstellen sind, oder duale Schnittstellen, die die Dispatch-Schnittstellen-Funktionen zusammen mit den von Ihnen definierten enthalten.

Welchen Ansatz verwenden Sie am besten? Das bleibt völlig Ihnen überlassen. Ich bevorzuge den zuletzt beschriebenen, weil die meisten Steuerelemente, die ich entwickle, die Werkzeuge von Desaware nutzen (was gar nicht erstaunlich ist, weil diese Werkzeuge hauptsächlich für mich und mein Team entwickelt wurden, um leistungsfähigere Steuerelemente erzeugen zu können). Weil wir die `apigid32.dll` (oder einen ihrer großen Brüder, die auch die Funktion `IsValidName` enthalten) bereitstellen, entstehen keine zusätzlichen Kosten in Hinblick auf die Ressourcen oder die Performance, wenn die DLL-Funktion genutzt wird.

18.1.3 Abhängigkeiten des Steuerelements

Die Extender-Eigenschaften, die einem bestimmten Steuerelement zur Verfügung gestellt werden, sind möglicherweise von den Einstellungen dieses Steuerelements abhängig. Diese Abhängigkeiten entstehen ausschließlich durch den Container. Im folgenden werden die wichtigsten dieser Abhängigkeiten beschrieben, die Sie als Visual-Basic-Programmierer kennen sollten.

`Enabled`. Die `Enabled`-Eigenschaft ist etwas seltsam. Um zu sehen, warum das so ist, laden Sie das Projekt `EnTest1.vbp` aus dem Beispielverzeichnis für Kapitel 18 auf der CD-ROM zum Buch. Dieses Projekt enthält ein einziges Steuerelement, `EnCtl1.ctl`, das eine Schaltfläche und eine öffentliche `Enabled`-Eigenschaft verwendet. Der Code für dieses Steuerelement sieht wie folgt aus:

```
Option Explicit

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

'Eigenschaftenwerte einlesen
Private Sub UserControl_ReadProperties(PropBag As PropertyBag)
    UserControl.Enabled = PropBag.ReadProperty("Enabled", True)
End Sub

'Eigenschaftenwerte abspeichern
Private Sub UserControl_WriteProperties(PropBag As PropertyBag)
    Call PropBag.WriteProperty("Enabled", UserControl.Enabled, True)
End Sub
```

Dieser Code sollte Ihnen nun schon vertraut sein – so wird eine öffentliche Eigenschaft implementiert und persistent gemacht. Die interne `Enabled`-Eigenschaft des `UserControl`-Objekts wird abhängig von der öffentlichen Variablen gesetzt. Dem Steuerelement wurde eine `Report`-Funktion hinzugefügt. Sie gibt den Status der `Enabled`-Eigenschaft, die `Enabled`-Eigenschaft von `UserControl` und die `Enabled`-Eigenschaft von `Command1` in das Direktfenster aus:

```
Public Sub Report()
    Debug.Print "Im Steuerelement-Code - "
    Debug.Print "Enabled-Eigenschaft: " & Enabled
    Debug.Print "UserControl.Enabled-Eigenschaft: " & _
    UserControl.Enabled
    Debug.Print "Command1.Enabled: " & Command1.Enabled
    Debug.Print "Extended Enabled: " & Extender.Enabled
    Debug.Print "Ende Steuerelement-Code."
End Sub
```

Das Projekt enthält ein einziges Formular, `enForm1`, das einen Timer und das Steuerelement `EnCtl1` verwendet. Während des `Timer`-Ereignisses wird die folgende Routine aufgerufen:

```
Private Sub Timer1_Timer()
    Debug.Print "Enabled-Eigenschaft: " & Enabled
    Debug.Print "Enabled-Eigenschaft des Steuerelements: " & _
    EnableTestControl.Enabled
    Debug.Print "Report: ";
    EnableTestControl.Report
End Sub
```

Für das Steuerelement `EnCtl1` sind im VB-Eigenschaftenfenster des Formulars zwei Eigenschaften gesetzt: Die `Enabled`-Eigenschaft ist auf `True` gesetzt, und die `Name`-Eigenschaft ist auf `EnableTestControl` gesetzt. Die `Enabled`-Eigenschaft des Formulars ist auf `False` gesetzt.

Bevor Sie dieses Beispiel ausführen, wollen wir überlegen, mit welchen verschiedenen `Enabled`-Eigenschaften wir es zu tun bekommen. Im Steuerelement (aus Perspektive des Steuerelement-Autors) sind das:

Syntax	Worauf Sie zugreifen
`Enabled`	Die öffentliche `Enabled`-Eigenschaft für das Steuerelement.
`UserControl.Enabled`	Die `Enabled`-Eigenschaft des `UserControl`-Objekts
`Command1.Enabled`	Die `Enabled`-Eigenschaft der Standard-Schaltfläche im Steuerelement.
`Extender.Enabled`	Die `Enabled`-Eigenschaft, die der Container als Teil des `Extender`-Objekts bereitstellt. Aufgrund der zuvor beschriebenen Regeln für den Gültigkeitsbereich, sieht der Entwickler, der dieses Steuerelement einsetzt, diese Eigenschaft, es sei denn, er greift explizit auf die `Enabled`-Eigenschaft des Steuerelements zu, unter Verwendung der Object-Eigenschaft und der Syntax `EnableTestControl.Object.Enabled`.

Für den Container (Perspektive des Anwendungsentwicklers) sehen die Eigenschaften wie folgt aus:

Syntax	Worauf Sie zugreifen
`Enabled`	Die `Enabled`-Eigenschaft für das Formular (eigentlich die `Enabled`-Eigenschaft für das `Extender`-Objekt des Formulars).
`EnableTestControl.Enabled`	Die vom Container als Teil des `Extender`-Objekts für das Steuerelement bereitgestellte `Enabled`-Eigenschaft.
`EnableTestControl.Object.Enabled`	Die öffentliche `Enabled`-Eigenschaft für das Steuerelement. (Entwickler sollten auf diese Eigenschaft nie zugreifen.)

Bei der Ausführung dieser Applikation sehen Sie die folgende Anzeige im Direkt-Fenster:

```
Enabled-Eigenschaft: False
Enabled-Eigenschaft des Steuerelements: False
Report: Im Steuerelement-Code -
Enabled-Eigenschaft: True
UserControl.Enabled-Eigenschaft: True
Command1.Enabled: True
Extended Enabled: False
Ende Steuerelement-Code.
```

Die `Enabled`-Eigenschaft für das Formular ist deaktiviert. Das ist sinnvoll, weil sie im VB-Eigenschaftenfenster für das Formular gesetzt wurde.

Die `Enabled`-Eigenschaft des Steuerelements, das sich auf dem `Extender`-Objekt befindet, ist `False`. Beachten Sie, daß sie zur Entwurfszeit auf `True` gesetzt wurde. Aber die öffentliche `Enabled`-Eigenschaft für das Steuerelement sowie die des `UserControl`-Objekts im Steuerelement sind alle `True`! Was ist da los? Warum ist die `Enabled`-Eigenschaft des Extenders `False`, wenn doch alle `Control`-Eigenschaften aktiviert sind?

Das ist eine spezifische Konvention von Visual Basic. Betrachten Sie zunächst die Arbeitsweise von Windows. Wenn Sie ein Fenster deaktivieren, erscheinen alle seine Kindfenster weiterhin als aktiviert. Sie erhalten keine Tastencodes und Mauseingaben, weil ihr Eltern-Formular deaktiviert ist. Wenn Sie möchten, daß sie als deaktiviert erscheinen, müssen Sie sie einzeln deaktivieren. Das entspricht der Situation, die wir in Kapitel 17 für die Sichtbarkeit gesehen haben. Dabei konnte für die Elemente auf einem Formular die `Visible`-Eigenschaft auf `True` gesetzt sein, obwohl sie tatsächlich verborgen waren, wenn ihr Container verborgen war.

In diesem Beispiel haben wir das Formular deaktiviert, nicht aber das Steuerelement. Die Entwickler von Visual Basic beschlossen (aus Gründen, die irgendwann vergessen und nicht festgehalten wurden), daß die Steuerelemente eines deaktivierten Containers ebenfalls als deaktiviert gelten sollen (obwohl sie nicht explizit deaktiviert wurden). Sie konnten jedoch nicht einfach die Steuerelemente deaktivieren. Das hätte ihr Erscheinungsbild verändert, was nicht dem Standardverhalten von Windows-Applikationen entspricht. Deshalb blieben die Steuerelemente aktiviert (falls sie betroffen waren), aber man manipulierte das Ganze so, daß die `Enabled`-Eigenschaft im `Extender`-Objekt für Steuerelemente auf einem deaktivierten Container `False` ist.

Damit dieser Trick funktioniert, ist eine Sache zwingend erforderlich. Es gibt keine Möglichkeit, wie Visual Basic erkennen könnte, welche Eigenschaft die `Enabled`-Eigenschaft ist. Man kann nicht davon ausgehen, daß sie immer den Namen »`Enabled`« hat – möglicherweise will der Steuerelement-Autor den Eigenschaftsnamen lokalisieren oder einen völlig anderen Begriff dafür verwenden. Wie können Sie als Steuerelement-Autor Visual Basic mitteilen, welche Ihrer Eigenschaften die `Enabled`-Eigenschaft für das Steuerelement ist?

Die Lösung liegt in der `IDispatch`-Schnittstelle, die in Teil I dieses Buchs vorgestellt wurde. Sie wissen, daß jede Eigenschaft auf der Schnittstelle eine eindeutige Dispatch-ID hat. Außerdem wissen Sie, daß das COM negative Dispatch-IDs definiert, die bestimmte Standardeigenschaften bezeichnen. Diese Werte haben keinen Einfluß auf das Verhalten der Schnittstelle oder der eigentlichen Eigenschaft, können aber vom Container genutzt werden, um diese Eigenschaften gegebenenfalls gesondert zu behandeln. Die Dispatch-ID -514 bezeichnet die `Enabled`-Eigenschaft.

Mit dem Befehl EXTRAS/PROZEDURATTRIBUTE zeigen Sie das Dialogfeld PROZEDURATTRIBUTE an. Wählen Sie die Enabled-Eigenschaft aus, und klicken Sie auf die Schaltfläche WEITERE. Im Kombinationsfeld Prozedur-ID sehen Sie, daß die Prozedur-ID von Enabled ausgewählt ist. Visual Basic setzt die Dispatch-ID für diese Eigenschaft automatisch auf -514.

Wenn Sie ein leeres Steuerelement ohne Eigenschaften anlegen, erzeugt Visual Basic standardmäßig keine Enabled-Eigenschaft für das Extender-Objekt. Sobald Sie jedoch einer der Steuerelement-Eigenschaften die Enabled-Prozedur-ID zuweisen, fügt Visual Basic dem Extender für das Steuerelement eine Enabled-Eigenschaft hinzu. Wenn Sie die Enabled-Eigenschaft im Code oder im VB-Eigenschaftenfenster setzen, setzt Visual Basic auch die Eigenschaft, die Sie mit dieser ID angegeben haben, unabhängig von dem eigentlichen Namen der Eigenschaft.

Was passiert, wenn Sie dem Steuerelement eine Enabled-Eigenschaft hinzufügen, aber die Prozedur-ID für die Eigenschaft nicht auf Enabled setzen? Visual Basic legt keine Enabled-Eigenschaft auf dem Extender an! Die Enabled-Eigenschaft, die Sie im VB-Eigenschaftenfenster sehen, ist die Enabled-Eigenschaft des Steuerelements.

Wenn Sie die Prozedur-ID für das Steuerelement EnCtl1.ctl auf (Keine) setzen und das Programm erneut ausführen, sehen Sie die folgenden Debug-Meldungen im Direkt-Fenster (möglicherweise müssen Sie Ihr Projekt speichern und neu laden, so daß es nach dieser Änderung korrekt eingestellt wird):

```
Enabled-Eigenschaft: False
Enabled-Eigenschaft des Steuerelements: True
Report: Im Steuerelement-Code -
Enabled-Eigenschaft: True
UserControl.Enabled-Eigenschaft: True
Command1.Enabled: True
Extended Enabled: True
Ende Steuerelement-Code.
```

Wie Sie sehen, gibt es keinen Unterschied darin, wie der Entwickler, der das Steuerelement einsetzt, die Enabled-Eigenschaft sieht, und der Perspektive des Steuerelement-Autors.

Wenn Sie nun das Standardverhalten von Visual Basic für die Enabled-Eigenschaft steuern möchten, gehen Sie wie folgt vor:

- Fügen Sie Ihrem Steuerelement eine Enabled-Eigenschaft hinzu (dazu können Sie einen beliebigen Namen verwenden, aber Sie sollten hier einfach Enabled verwenden, es sei denn, es sprechen wirklich gute Gründe dagegen).
- Setzen Sie die Prozedur-ID für diese Eigenschaft auf Enabled.

- Innerhalb der Eigenschaftsprozeduren für die `Enabled`-Eigenschaft setzen Sie die interne `Enabled`-Eigenschaft für das Steuerelement und gegebenenfalls alle Standardelemente (wenn Sie möchten, daß sie deaktiviert aussehen, wenn das Steuerelement deaktiviert wird). Für benutzerdefinierte Steuerelemente können Sie das Erscheinungsbild auch so ändern, daß es den Status anzeigt.
- Machen Sie den Wert der Eigenschaft während der Ereignisse `ReadProperties` und `WriteProperties` persistent.

`Align` **und** `Negotiate`. Wenn Sie in Ihrem Steuerelement die `Alignable`-Eigenschaft auf `True` setzen, zeigt der `Extender` eine `Align`-Eigenschaft an, und möglicherweise auch eine `Negotiate`-Eigenschaft (abhängig vom Container).

Ihr Steuerelement kann die `Align`-Eigenschaft des Extenders lesen, um festzustellen, ob Ihr Steuerelement ausgerichtet wurde. Sie brauchen in diesem Fall überhaupt nichts zu tun; der Container ist dafür verantwortlich, Ihr Steuerelement neu zu positionieren.

Wenn Ihr Steuerelement ein Symbolleisten-Steuerelement verwendet, um eine Symbolleiste zu implementieren, wird die Negotiate-Eigenschaft bereitgestellt, so daß der Entwickler festlegen kann, wie die Symbolleiste auf einem MDI-Formular angezeigt wird. Weitere Informationen über das Symbolleisten-Werkzeug finden Sie in Ihrer Visual-Basic-Dokumentation.

`Cancel` **und** `Default`. Wenn Sie die `DefaultCancel`-Eigenschaft Ihres Steuerelements auf `True` setzen, fügt Visual Basic dem Extender Ihres Steuerelements eine `Default`- und eine `Cancel`-Eigenschaft hinzu. Sie können diese `Extender`-Eigenschaften lesen, um zu erkennen, ob Ihr Steuerelement zum Standard- oder Abbrechen-Element für den Container gemacht wurde.

Ein Standard-Steuerelement weist in der Regel ein anderes Erscheinungsbild als ein Standardelement desselben Typs auf. Sie könnten die Standard-Extender-Eigenschaft auswerten, aber Visual Basic bietet eine einfachere Methode, sie zu erkennen. Das `Ambient`-Objekt hat eine `DisplayAsDefault`-Eigenschaft, die auf `True` gesetzt ist, wenn Ihr Steuerelement in seinem Standardstatus angezeigt werden soll. Warum sollte man statt des Extenders die `Ambient`-Eigenschaft verwenden?

- Weil die `Ambient`-Eigenschaft früh gebunden werden kann.
- Weil Sie mit Hilfe des `UserControl_AmbientChanged`-Ereignisses erkennen können, wenn sich die `DisplayAsDefault`-Eigenschaft geändert hat.

Das Steuerelement `ch18CtlB.ctl` demonstriert das Ganze im folgenden Code:

```
Private Sub UserControl_AmbientChanged(PropertyName As String)
   If PropertyName = "DisplayAsDefault" Then
      UserControl.Refresh
   End If
End Sub
```

```
Private Sub UserControl_GotFocus()
   Label2.Caption = "Has Focus"
End Sub

Private Sub UserControl_LostFocus()
   Label2.Caption = "Lost Focus"
End Sub

Private Sub UserControl_Paint()
   Dim s$
   If Extender.Default Then
      s$ = "Default"
   End If
   If Extender.Cancel Then
      s$ = s$ & " Cancel"
   End If
   If s$ = "" Then s$ = "Normal"
   Label1.Caption = s$
End Sub
```

Das Steuerelement enthält zwei Standard-Bezeichnungsfelder, in denen sein aktueller Status angezeigt wird. Öffnen Sie das Formular `frmCh18Test2` im Entwurfsmodus und versuchen Sie, die `Default`- und die `Cancel`-Eigenschaft im VB-Eigenschaftenfenster zu ändern (beachten Sie, daß Sie damit direkt die `Extender`-Eigenschaften ändern). Wenn Sie die `Default`-Eigenschaft ändern, wird das Steuerelement sofort aktualisiert. Wenn Sie die `Cancel`-Eigenschaft ändern, sehen Sie die Änderung erst, nachdem das Steuerelement neu gezeichnet wurde (versuchen Sie, die Größe des Steuerelements zu ändern, nachdem Sie die `Cancel`-Eigenschaft geändert haben, um den Effekt zu erkennen).

Gibt es eine Möglichkeit zu erkennen, wenn sich eine `Extender`-Eigenschaft geändert hat? Ich kenne nur das Abfragen. Wenn Sie es besser wissen, dann teilen Sie es mir doch bitte mit.

18.1.4 Container-Abhängigkeiten

Erlauben Sie mir, diesen Abschnitt mit zwei Warnungen abzuschließen. Das `Extender`-Steuerelement und die Entwurfsumgebung von Visual Basic sind außergewöhnlich robust. Andere Container haben sehr wahrscheinlich weniger `Extender`-Eigenschaften und bieten eine weniger komplexe Unterstützung der Funktionen, die Sie möglicherweise in Ihr Steuerelement eingebaut haben. Sie sollten die Verwendung aller `Extender`-Eigenschaften überprüfen und sicherstellen, daß Ihr Steuerelement auch noch funktioniert, wenn sie nicht verwendet werden können. Es gibt mehrere Möglichkeiten, wie sich andere Container in Hinblick auf `Extender`-Eigenschaften verhalten:

- Sie deaktivieren die von der `Extender`-Eigenschaft unterstützte Funktion so, daß es der Entwickler nicht merkt.
- Sie deaktivieren die von der `Extender`-Eigenschaft unterstützte Funktion und benachrichtigen den Entwickler (indem sie beim Zugriff auf die Eigenschaft einen Fehler aufwerfen, ein Nachrichtenfeld anzeigen oder auf irgendeine andere Weise darauf aufmerksam machen).
- Sie deaktivieren das Steuerelement vollständig, wenn die `Extender`-Eigenschaft für seine Arbeit zwingend erforderlich ist. Ihr Steuerelement kann in so einem Container nicht ausgeführt werden.
- Sie lassen einen Laufzeitfehler zu, wenn ein Zugriff auf die `Extender`-Eigenschaft erfolgt. Diese Vogel-Strauß-Verhalten sollten Sie unbedingt vermeiden.

Üble, üble Dinge, die Sie nie tun sollten. Der Container und die `Parent`-Eigenschaften geben Ihnen Zugriff auf das `Container`-Objekt und alle seine Eigenschaften und Methoden. Das bedeutet, Ihr Steuerelement kann die folgenden Dinge tun:

- Sich selbst auf den Container zeichnen.
- Die Farbe des Containers ändern.
- Andere Steuerelemente im Container neu anordnen, ihre Größe oder die Sichtbarkeit ändern.
- Seine eigene `Extender.Container`-Eigenschaft auf einen anderen Container der Applikation setzen.
- Neue Steuerelemente anlegen und sie auf dem Container plazieren.
- Den Container aus dem Speicher entfernen.

Ich könnte seitenlang weiterschreiben. Steuerelemente können im höchsten Maße ungezogen sein.

Grundlegend für die Philosophie der Komponenten ist, daß der Entwickler, der Ihr Steuerelement einsetzt, für den Container verantwortlich ist. Darüber hinaus ist er verantwortlich für das Setzen und die Verwendung der `Extender`-Eigenschaften und Ereignisse (weil diese vom Container bereitgestellt werden). Als Steuerelement-Autor sollten Sie Ihre Finger von Containern und anderen Steuerelementen lassen!

18.2 Ambient-Eigenschaften

Manchmal kann man zwischen `Extender`- und `Ambient`-Eigenschaften nicht so richtig unterscheiden. Wie bei so vielen Aspekten der Entwicklung von ActiveX-Steuerelementen ist der Unterschied zwischen diesen Eigenschaften von der Perspektive des Betrachters abhängig. `Extender`-Eigenschaften sind diejenigen, die

der Container Ihrem Steuerelement hinzufügt, und die hauptsächlich von Entwicklern eingesetzt werden, die Ihr Steuerelement in ihren Applikationen verwenden. `Ambient`-Eigenschaften dagegen sind diejenigen, die der Container bereitstellt, und die von Steuerelement-Autoren genutzt werden sollen.

Der Zugriff auf `Ambient`-Eigenschaften erfolgt über das `AmbientProperties`-Objekt, das über die `Ambient`-Eigenschaft des `UserControl`-Objekts für Ihr Steuerelement bereitsteht. Das `AmbientProperties`-Objekt beinhaltet immer eine Kernmenge von Standardeigenschaften, anders als das `Extender`-Objekt, dessen Eigenschaften abhängig vom Container und der Konfiguration des Steuerelements variieren. Das bedeutet jedoch nicht, daß alle Standard-Ambient-Eigenschaften von jedem Container unterstützt werden. Wenn eine `Ambient`-Eigenschaft von einem Container nicht unterstützt wird, gibt das `AmbientProperties`-Objekt einfach einen Standardwert für die Eigenschaft zurück.

Betrachten Sie beispielsweise die `UserMode`-Eigenschaft. Wenn ein Container keinen Entwurfsmodus unterstützt, muß er auch keine UserMode-Ambient-Eigenschaft bereitstellen.

Das `AmbientProperties`-Objekt erkennt, daß die Eigenschaft im Container fehlt, und gibt für seine eigene `UserMode`-Eigenschaft immer den Standardwert `True` zurück.

Ein Container kann zusätzliche, für ihn spezifische `Ambient`-Eigenschaften bereitstellen. Diese Eigenschaften werden immer spät gebunden, und Sie sollten für den Zugriff darauf eine Fehlerverarbeitung einführen. Tabelle 18.3 zeigt die Standardeigenschaften des `Ambient`-Objekts. Der Standardwert gibt den Wert an, der für die Eigenschaft zurückgegeben wird, wenn sie vom Container nicht unterstützt wird. Das `AmbientChanged`-Ereignis des `UserControl`-Objekts wird immer dann ausgelöst, wenn eine `Ambient`-Eigenschaft geändert wird.

Eigenschaft	Kommentar	Standardwert
`BackColor`	Hintergrundfarbe des Containers.	&H80000005
`DisplayAsDefault`	Wenn die `DefaultCancel`-Eigenschaft des Steuerelements `True` ist, und der Entwickler dies als Standard-Steuerelement vorschreibt, ergibt diese Eigenschaft `True`. Eine genauere Beschreibung der `DefaultCancel`-Eigenschaft finden Sie in Kapitel 17.	

Tab. 18.3: Die Standardeigenschaften des Ambient-Objekts

Eigenschaft	Kommentar	Standardwert
`DisplayName`	Der Name, den der Entwickler dem Steuerelement zugeordnet hat.	""
Font	Die Schrift des Containers oder die Standardschrift, die der Container für das Steuerelement empfiehlt. Beachten Sie, daß die `FontTransparent`-Eigenschaft des Containers kein `AmbientChanged`-Ereignis auslöst.	MS Sans Serif 8
`ForeColor`	Die Vordergrundfarbe des Containers.	&H80000008
`LocaleID`	Siehe Kapiteltext.	Aktueller Systemstandard.
`MessageReflect`	Siehe Kapiteltext.	`False`
`Palette`	Eine `Picture`-Eigenschaft, die die vom Container empfohlene Palette angibt (das ist in der Regel die Palette, die der Container verwendet).	
`RightToLeft`	Gibt an, daß das Steuerelement Text von rechts nach links ausgeben soll, wie beispielsweise in hebräischen oder arabischen Windows-Versionen.	
`ScaleUnits`	Der Name der vom Container benutzten Einheiten. Siehe Kapiteltext.	
`ShowGrabHandles`	Siehe Kapiteltext.	True
`ShowHatching`	Siehe Kapiteltext.	True
`SupportsMnemonics`	Siehe Kapiteltext.	False
`TextAlign`	Gibt die Textausrichtung des Containers oder die Standard-Textausrichtung des Steuerelements an.	Null
`UserMode`	True zeigt an, daß sich das Steuerelement im Endbenutzer-Modus befindet, d.h. in Visual Basic-Laufzeit.	True
`UIDead`	Siehe Kapiteltext.	False

Tab. 18.3: Die Standardeigenschaften des Ambient-Objekts

18.2.1 Spezielle Eigenschaften

Die folgenden Ambient-Eigenschaften sollen detaillierter beschrieben werden.

Eingekapselte Eigenschaften. Die Ambient-Eigenschaften MessageReflect, ShowGrabHandles, ShowHatching, SupportsMnemonics und UIDead sind wichtig für Steuerelemente, werden aber ausschließlich durch die Implementierung der ActiveX-Steuerelemente von Visual Basic verarbeitet. Hier eine kurze Beschreibung dieser Eigenschaften, falls Sie daran interessiert sind:

- MessageReflect – Gibt an, daß der Container bestimmte Fensternachrichten an das Steuerelement weitergeben soll.
- ShowGrabHandles – Gibt an, daß der Container Anfasser für die Größenänderung des Steuerelements anzeigen kann.
- ShowHatching – Gibt an, daß der Container gegebenenfalls ein Schraffurmuster für ein nicht aktives Steuerelement anzeigen kann.
- SupportsMnemonics – Gibt an, daß der Container Zugriffstasten für ein Steuerelement unterstützen kann.
- UIDead – Gibt an, daß ein Steuerelement alle Eingaben des Benutzers ignorieren soll.

ScaleUnits. Diese Eigenschaft kann genutzt werden, um die Skalierungseinheiten für den Container festzulegen. Handelt es sich bei dem Container um Visual Basic, kann ScaleUnits die Werte Benutzer, Twip, Punkt, Pixel, Zeichen, Zoll, Millimeter und Zentimeter annehmen. Das sind die Konstantennamen für die ScaleMode-Eigenschaften.

Die möglichen Werte dieser Eigenschaft werden nicht durch die ActiveX-Spezifikation definiert. Container können also beliebige Strings verwenden. Das bedeutet, diese Eigenschaft legt nicht für alle Container den korrekten Skalierungsmodus fest. Sie wird hauptsächlich für Steuerelemente bereitgestellt, die gegebenenfalls im Koordinatensystem des Containers dargestellt werden können.

Handelt es sich bei dem Container um Visual Basic, können Sie auch direkt über die Container-Eigenschaft des Extender-Objekts auf die ScaleMode-Eigenschaft zugreifen.

LocaleID. Eine Lokale ist ein 32-Bit-Wert, der die Sprache und Plattform für den Thread oder das System angibt. Die Bits von 0 bis 15 (im unteren Wort) spezifizieren die Sprache. Die Bits 16 bis 19 geben die Sortierreihenfolge in dieser Sprache an. Das ist in der Regel 0, kann aber für Unicode-Umgebungen in Fernost auch auf 1 gesetzt werden. Abbildung 18.2 zeigt die Struktur einer LocaleID.

Das untere Wort ist unterteilt. Die unteren 10 Bit (Bits 0 bis 9) geben die Sprache an. Die oberen 6 Bit (Bit 10 bis 15) geben eine Untermenge der einzustellenden Sprache an, beispielsweise die Unterscheidung zwischen Englisch (USA) und Englisch (Großbritannien). Die Werte für die unterstützte Sprache finden Sie in

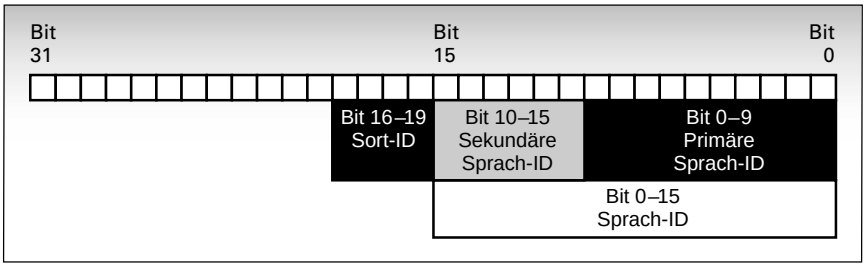

Abb. 18.2: Die LocaleID.

jeder guten Win32-API-Deklarationsdatei: Suchen Sie nach den Konstanten mit dem Präfix `LANG_`. Die Sprachuntermengen werden durch das Präfix `SUBLANG_` gekennzeichnet. Viele Funktionen verwenden das Standardsystem oder benutzerdefinierte Lokale, die durch die speziellen Konstantenwerte vorgegeben sind:

```
Public Const LOCALE_SYSTEM_DEFAULT = &H800
Public Const LOCALE_USER_DEFAULT = &H400
```

In den Lokalen-Definitionen ist sehr viel Information enthalten, beispielsweise Ländername, Datums- und Monatsnamen (und ihre Abkürzungen), das lokale Währungssymbol, das Format für Datum und Zeit und andere Dinge, die sich zwischen den verschiedenen Ländern unterscheiden. Diese Information kann Ihnen helfen, mit Visual Basic wirklich internationale Applikationen zu entwikkeln. Das Thema der Lokalisierung und der entsprechenden Win32-API-Funktionen werden in meinem Buch *Dan Appleman's Visual Basic 5.0 Programmer's Guide to the Win32 API* genauer beschrieben.

`UserMode`. Die `Ambient`-Eigenschaft `UserMode` ist die vielleicht wichtigste Eigenschaft. Aber mit Sicherheit ist sie diejenige, die Sie am häufigsten verwenden werden. Wenn Sie diese Eigenschaft auf `True` setzen, bedeutet das, daß sich der Container im Entwurfs- oder Entwickler-Modus befindet. Beachten Sie, daß nicht jeder Container eine Entwurfsumgebung unterstützt.

Die gebräuchlichste Verwendung dieser Eigenschaft ist:

- zu bewirken, daß Eigenschaften nur zur Entwurfszeit gesetzt werden können (das wird in Kapitel 19 noch gezeigt);
- die Definition verschiedener Erscheinungsbilder für Entwurfszeit und Laufzeit;
- die Definition verschiedener Verhalten für Entwurfszeit und Laufzeit (unter anderem die Verwaltung der Aktivierung zur Laufzeit, falls Sie das unterstützen).

18.2.2 Strategien für die Verwendung von Ambient-Eigenschaften

Es gibt zahlreiche Ansätze für die Arbeit mit `Ambient`-Eigenschaften. Um Ihnen das genauer zu zeigen, betrachten wir die `BackColor`-Eigenschaft.

Dabei handelt es sich um eine der Eigenschaften, bei der Sie ganz genau unterscheiden müssen, welches Objekt Sie betrachten. Die Projektgruppe `ch18Test1` enthält das Steuerelement `ch18ctlD` mit drei Standardtextfeldern. Das bedeutet, daß Sie es als Steuerelement-Autor mit sechs verschiedenes `BackColor`-Eigenschaften für sechs verschiedene Objekte zu tun haben, wie in Tabelle 18.4 gezeigt.

Eigenschaft	Beschreibung
`Text1.BackColor`	Nur Autor – setzt die Hintergrundfarbe des `Text1`-Steuerelements
`Text2.BackColor`	Nur Autor – setzt die Hintergrundfarbe des `Text2`-Steuerelements
`Text3.BackColor`	Nur Autor – setzt die Hintergrundfarbe des `Text3`-Steuerelements
`UserControl.BackColor`	Nur Autor – setzt die Hintergrundfarbe des ActiveX-Steuerelements
`BackColor`	Öffentlich – macht, was immer Sie als Steuerelement-Autor möchten
`Ambient.BackColor`	Öffentlich – die aktuelle Hintergrundfarbe des Containers für das Steuerelement

Tab. 18.4: Jede Menge BackColor-Eigenschaften

Das führt zu einigen interessanten Fragen. Wie genau soll dieses Steuerelement aussehen? Wodurch wird die Hintergrundfarbe der verschiedenen Textfelder und des eigentlichen Steuerelements festgelegt? Es gibt zahlreiche Möglichkeiten:

- Das Steuerelement und seine Standardelemente können eine einzige öffentliche `BackColor`-Eigenschaft gemeinsam nutzen.
- Jedes Standardelement kann unabhängig gesetzt werden.
- Das Steuerelement und seine Standardelemente können die `BackColor`-Eigenschaft des Containers benutzen
- Sie können diese Ansätze beliebig kombinieren.

Ich würde nicht wagen, Ihnen einen bestimmten Ansatz besonders zu empfehlen. Sie sollten sich jedoch beim Entwurf Ihres Steuerelements diese Fragen stellen.

Das Steuerelement `ch18CtlD` zeigt alle diese Ansätze, die hoffentlich lehrreich und hoffnungslos uninteressant sind. Die öffentliche `BackColor`-Eigenschaft definiert die Hintergrundfarbe für das Steuerelement und das Textfeld. Die Eigenschaftsfunktionen 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
   Text1.BackColor = UserControl.BackColor
   PropertyChanged "BackColor"
End Property
```

Hier gibt es zwei Dinge zu beachten. Erstens, den Eigenschaftstyp `OLE_COLOR`. Dieser Eigenschaftstyp wird unter Verwendung der Standard-OLE-Bibliothek (`stdole`) definiert, auf die Visual Basic standardmäßig zugreift. Die Variable `OLE_COLOR` ist ein 32-Bit `Long`, und Variablen dieses Typs können direkt numerischen Variablen von Visual Basic zugewiesen werden und umgekehrt. Warum sollten Sie den Typ `OLE_COLOR` verwenden? Weil Visual Basic ein sehr gescheiter Container ist, und wenn es eine Eigenschaft mit dem Typ `OLE_COLOR` sieht, diese dem Popup-Menü zur Farbauswahl im VB-Eigenschaftenfenster für dieses Steuerelement hinzufügt.

Der andere interessante Aspekt ist, daß zwei Objekte mit der `BackColor`-Eigenschaft »verbunden« sind. Wenn die Eigenschaft gesetzt wird, wird auch die `BackColor`-Eigenschaft für das `UserControl` und das `Text1`-Steuerelement gesetzt. Dieses Beispiel will zeigen, daß diese beiden Objekte auf diese Weise immer dieselbe Hintergrundfarbe haben. Die Eigenschaft wird in der Funktion `UserControl_WriteProperties` gespeichert:

```
Call PropBag.WriteProperty("BackColor", UserControl.BackColor, &H8000000F)
```

Sie wird wie folgt gelesen:

```
UserControl.BackColor = PropBag.ReadProperty("BackColor", &H8000000F)
     Text1.BackColor = UserControl.BackColor
```

Wenn wir die Eigenschaft schreiben, können wir den Wert aus dem `UserControl` oder aus dem `Text1`-Objekt verwenden, weil sie gleich sind. Wenn wir die Eigenschaft lesen, müssen wir sicherstellen, daß beide auf den neuen Wert gesetzt werden. Außerdem muß die Eigenschaft im `UserControl_InitProperties`-Ereignis initialisiert werden. Dazu gibt es zwei Möglichkeiten. Die eine sieht wie folgt aus:

```
     UserControl.BackColor = PropBag.ReadProperty("BackColor",
&H8000000F)
     Text1.BackColor = UserControl.BackColor
     PropertyChanged "BackColor"
```

Die andere geht einfach so vor:

```
     BackColor = &H8000000F
```

In diesem Fall dient die öffentliche BackColor-Einstellung genau demselben Zweck. Die Vorteile bei der Verwendung der öffentlichen Eigenschaft auf diese Weise ist offensichtlich: Sie erlaubt, denselben Code wiederzuverwenden, und hilft Ihnen, wichtige Operationen nicht zu vergessen (beispielsweise den Aufruf von PropertyChanged, oder das Setzen aller erforderlichen Werte). Es gibt jedoch Situationen, wo das nicht möglich ist, beispielsweise, wenn Sie Code oder eine Fehlerverarbeitung in der Routine zum Setzen der öffentlichen Eigenschaft haben, die nicht aufgerufen werden sollen, wenn die Eigenschaftseinstellungen aus einer Datei gelesen werden.

Das Standardelement Text2 verwendet einen anderen Ansatz. Es wird unter Verwendung einer unabhängigen Eigenschaft gesetzt, BackColor2. Damit ist es völlig unabhängig von den anderen Hintergrundfarben und seine Eigenschaftsprozeduren sind ganz einfach:

```
' Hintergrundfarbe für Text2
Public Property Get BackColor2() As OLE_COLOR
   BackColor2 = Text2.BackColor
End Property

Public Property Let BackColor2(ByVal vNewValue As OLE_COLOR)
   Text2.BackColor = vNewValue
   PropertyChanged "BackColor2"
End Property
```

Das Standardelement Text3 ist etwas komplizierter. Seine Hintergrundfarbe wird abhängig von der Containerfarbe oder unabhängig unter Verwendung der BackColor3-Eigenschaft gesetzt. Das stellt weitere Anforderungen an den Code für das Steuerelement:

- Sie müssen eine Möglichkeit für den Entwickler schaffen, für das Text3-Steuerelement entweder die Hintergrundfarbe der Umgebung oder den Eigenschaftswert BackColor3 auszuwählen.
- Weil die BackColor-Eigenschaft des Text3-Steuerelements die Hintergrundfarbe des Containers reflektieren kann, ist es erforderlich, den Wert von BackColor3 unabhängig zu speichern.
- Sie müssen erkennen, wenn sich die BackColor-Eigenschaft des Containers ändert, so daß das Text3-Steuerelement gegebenenfalls aktualisiert werden kann.

Listing 18.1 zeigt das restliche Steuerelementmodul. Dieses Beispiel verfolgt einen einfachen Ansatz, wie der Entwickler zwischen den verschiedenen Modi für das Text3-Steuerelement wählen kann. Es verwendet eine separate Eigenschaft, Text3Ambient, die auf True gesetzt wird, wenn das Text3-Steuerelement den Ambient-Eigenschaftswert BackColor verwenden soll. Der Wert für diese

Eigenschaft wird in der privaten Variablen `m_Text3Ambient` gespeichert. Eine zweite private Variable, `m_Text3Backcolor`, nimmt den Wert der `BackColor3`-Eigenschaft auf.

Möglich wäre auch, einfach einen »unmöglichen« Farbwert zu definieren, was dazu führt, daß das Steuerelement die Hintergrundfarbe der Umgebung annimmt. In diesem Fall würden Sie den Parameter der `Property Let`-Funktion, `BackColor2`, auswerten.

Hat er einen unzulässigen Wert, beispielsweise &HFFFFFFF, setzen Sie die `m_Text3Ambient`-Eigenschaft auf `True`, statt die Variable `m_Text2Backcolor` zu setzen. Dieser Ansatz ist nicht besonders schön, weil er erforderlich macht, daß sich der Entwickler merkt, daß das Steuerelement auf den `Ambient`-Hintergrundmodus zurückgesetzt wird, indem in das Bearbeitungsfeld für `BackColor3` im VB-Eigenschaftenfenster -1 oder &HFFFFFFF eingegeben wird. Weil Farben in der Regel über die Popup-Farbpalette gesetzt werden, ist dieser Ansatz nicht sonderlich intuitiv. Eine separate Variable, die den `Text3`-Modus steuert, kann sich der Entwickler viel einfacher merken.

```
' Sollen wir den Ambient- oder den back3color-Wert verwenden?
    Private m_Text3Ambient As Boolean
    Private m_Text3Backcolor As OLE_COLOR

    Private Sub UserControl_AmbientChanged(PropertyName As String)
       If PropertyName = "BackColor" Then
          SetText3Color
       End If
    End Sub

    ' Eigenschaftswerte laden
    Private Sub UserControl_ReadProperties(PropBag As PropertyBag)
       UserControl.BackColor = PropBag.ReadProperty("BackColor", &H8000000F)
       Text1.BackColor = UserControl.BackColor
       Text2.BackColor = PropBag.ReadProperty("BackColor2", &H80000005)
       m_Text3Backcolor = PropBag.ReadProperty("BackColor3", &H80000005)
       m_Text3Ambient = PropBag.ReadProperty("Text3Ambient", True)
       SetText3Color  ' Nach dem Lesen aktualisieren
    End Sub

    'Eigenschaftswerte schreiben
    Private Sub UserControl_WriteProperties(PropBag As PropertyBag)
       Call PropBag.WriteProperty("BackColor", UserControl.BackColor, &H8000000F)
       Call PropBag.WriteProperty("BackColor2", Text2.BackColor, &H80000005)
       Call PropBag.WriteProperty("BackColor3", m_Text3Backcolor, &H80000005)
       Call PropBag.WriteProperty("Text3Ambient", m_Text3Ambient, True)
    End Sub

    Public Property Get BackColor3() As OLE_COLOR
       If Ambient.UserMode Then
```

```
      ' Zur Laufzeit wieder die aktuelle Farbe verwenden
      BackColor3 = Text3.BackColor
   Else
      BackColor3 = m_Text3Backcolor
   End If
End Property

Public Property Let BackColor3(ByVal vNewValue As OLE_COLOR)
   m_Text3Backcolor = vNewValue
   PropertyChanged "BackColor3"
   SetText3Color
End Property

Public Property Get Text3Ambient() As Boolean
   Text3Ambient = m_Text3Ambient
End Property

Public Property Let Text3Ambient(ByVal vNewValue As Boolean)
   m_Text3Ambient = vNewValue
   PropertyChanged "Text3Ambient"
   SetText3Color
End Property

Private Sub SetText3Color()
   If m_Text3Ambient Then
      Text3.BackColor = Ambient.BackColor
   Else
      Text3.BackColor = m_Text3Backcolor
   End If
End Sub
```

Abb. 18.1: Ausschnitt aus dem Code für ch18ctlD.ctl

Die eigentliche Einstellung der Hintergrundfarbe von `Text3` erfolgt in der Funktion `SetText3Color`. Diese Funktion muß aufgerufen werden, wenn sich die Hintergrundfarbe von `Text3` ändern könnte. Das ist auch der Fall, nachdem die Eigenschaften während des `UserControl_ReadProperties`-Ereignisses gelesen werden, und immer, wenn die Eigenschaften `Text3Ambient` oder BackColor3 gesetzt werden.

Um die Arbeitsweise dieses Codes nachzuvollziehen, laden Sie das Formular `frmCh18Test3` im Entwurfsmodus und probieren die verschiedenen Eigenschaftswerte im VB-Eigenschaftenfenster aus.

Wie Sie gesehen haben, arbeiten die `Ambient`- und `Extender`-Objekte eng mit den Eigenschaften zusammen, die Sie für Ihre ActiveX-Steuerelemente definieren. Aber wie Sie im nächsten Kapitel erkennen werden, haben Sie gerade erst begonnen, die Eigenschaften von ActiveX-Steuerelementen kennenzulernen.