Kapitel 17

Das UserControl-Objekt

Das `UserControl`-Objekt bildet Herz und Seele jedes unter Visual Basic entwikkelten ActiveX-Steuerelements. Man könnte sagen, es ist das Formular für ActiveX-Steuerelemente. Es enthält die Schnittstellen, die ein ActiveX-Steuerelement zum Leben braucht, und die ihm die Kommunikation mit dem Container erlauben. Vor allem jedoch ist das `UserControl`-Objekt ein COM-Objekt. Und um ein COM-Objekt zu verstehen, müssen Sie seine Schnittstellen kennen.

Hier werden nicht alle Ereignisse und Eigenschaften von `UserControl`-Objekten beschrieben. Viele davon sollten Ihnen von der allgemeinen Visual-Basic-Programmierung her bekannt sein. Statt dessen will ich mich auf die Ereignisse und Methoden konzentrieren, die es nur bei ActiveX-Steuerelementen gibt, oder die sich für ActiveX-Steuerelemente anders verhalten als in Formularen oder Steuerelementen. Ich habe versucht, die Eigenschaften und Ereignisse so zu gruppieren, wie sie zusammengehören, und Beispiele bereitgestellt, die ihr Verhalten demonstrieren.

17.1 Ereignisse, die die Lebensdauer betreffen

In Kapitel 16 haben Sie erfahren, in welchen verschiedenen Statuszuständen sich ein Steuerelement befinden kann. Das typische Leben eines Steuerelements während seiner Entwicklung (beim Erstellen und Testen) sieht wie folgt aus:

- **Designer geöffnet** – Das Steuerelement befindet sich im Entwurfsmodus.
- **Designer geschlossen –** Es existiert noch keine Instanz des Steuerelements.
- **Das Steuerelement wird auf einen Container gezeichnet –** Eine Instanz des Steuerelements wird angelegt und befindet sich im Ausführungsmodus auf einem Container, der sich im Entwurfsmodus befindet.
- **Der Container wird ausgeführt –** Jede existierende Entwurfszeit-Instanz des Steuerelements wird zerstört, und anschließend wird eine neue Instanz angelegt und auf einem Container ausgeführt, der sich ebenfalls im Ausführungsmodus befindet. Beim dynamischen Anlegen wird eine neue Instanz eines Steuerelements im Ausführungsmodus erzeugt.

Wenn Sie eine Programmdatei ausführen, die ein Steuerelement verwendet, findet nur Schritt 4 statt: Eine Instanz des Steuerelements wird erzeugt und auf einem Container ausgeführt, der sich im Ausführungsmodus befindet.

Weil Ihr Steuerelement einen Großteil seiner Zeit in Containern verbringen wird, die ausgeführt werden, müssen Sie als Autor des Steuerelements auch die Entwurfszeit des Containers berücksichtigen. Aus Ihrer Perspektive wird das Steuerelement nämlich auch in dieser Umgebung ausgeführt.

Das `UserControl`-Objekt hat zahlreiche Ereignisse. Es ist wichtig, ihre Arbeitsweise in den verschiedenen Container-Modi zu verstehen, wann sie auftreten – und welche Dinge Sie während dieser Ereignisse tun dürfen, nicht tun dürfen, sollen und nicht sollen. Dasselbe gilt für die Eigenschaften.

Wir beginnen mit den Ereignissen, die sich direkt auf die Initialisierung und das Beenden eines Steuerelements beziehen. Die Gruppe `ch17tst1.vbg` enthält zwei Projekte, `ch17ctl1` und `ch17tst1.vbp`. Das Projekt `ch17ctl1.vbp` verwendet mehrere Steuerelemente, die viele der in diesem Kapitel beschriebenen Aspekte verdeutlichen.

Das erste Steuerelement im Projekt ist `ch17ctlA`. Dieses Steuerelement hat eine einzige öffentliche Eigenschaft, `myprop`. Es handelt sich dabei um eine `Variant`-Eigenschaft, die zur Entwurfszeit gesetzt werden kann. Sie beinhaltet `debug.print`-Anweisungen für mehrere `UserControl`-Ereignisse.

Öffnen Sie die Projektgruppe und stellen Sie sicher, daß die Steuerelement-Designer geschlossen sind. Möglicherweise müssen Sie das Steuerelement `ch17ctl1.ocx` für die Projektgruppe zunächst registrieren, um es fehlerfrei öffnen zu können. Wenn ein neues `ch17ctlA`-Steuerelement auf dem Formular `ch17tst1` gezeichnet wird, werden die folgenden Ereignisse ausgelöst:

```
UserControl_Initialize
UserControl_InitProperties
```

Wenn Sie das Projekt `ch17tst1` ausführen, treten die folgenden Fehler auf:

```
UserControl_Terminate
UserControl_Initialize
UserControl_ReadProperties
```

Das beweist, daß ein Steuerelement zerstört wird, sobald der Container vom Entwurfsmodus in den Ausführungsmodus wechselt.

Die Ereignisse `Initialize` und `Terminate` sind Ihnen von den Klassenmodulen her bekannt. `Initialize` ist immer das erste Ereignis, das stattfindet, wenn ein Objekt erzeugt wird. `Terminate` ist das letzte Ereignis, das auftritt, bevor es zerstört wird.

Wenn Sie wieder in den Entwurfsmodus wechseln, werden die folgenden Ereignisse ausgelöst:

```
UserControl_Terminate
UserControl_Initialize
UserControl_ReadProperties
```

Die Ereignisse, die sich auf Eigenschaften beziehen, werden wir später genauer betrachten.

17.1.1 Das Initialize-Ereignis

Das `Initialize`-Ereignis ist das erste Ereignis, das Ihr Steuerelement erhält. Während dieses Ereignisses sollten Sie die folgenden Dinge tun:

- Initialisierung aller Variablen für Ihr Steuerelement auf Modulebene.
- Initialisierung der Eigenschaften von Standardelementen mit ihren Standardwerten.

Während dieses Ereignisses können Sie:

- Auf die Eigenschaften aller Standardelemente zugreifen oder sie initialisieren. Zu diesem Zeitpunkt existieren bereits alle Standardelemente.
- Variablen, die die Eigenschaftswerte des Steuerelements aufnehmen, mit den Standardwerten initialisieren (das wird in einem späteren Abschnitt noch beschrieben).

Während dieses Ereignisses dürfen Sie nicht:

- Auf die `Extender`- oder `Ambient`-Eigenschaften des `UserControl`-Objekts oder eine ihrer Eigenschaften zugreifen. Ihr Steuerelement existiert zu diesem Zeitpunkt zwar schon, aber es wurde noch nicht auf dem Container plaziert (oder, wie man es heute auch nennt, das Steuerelement wurde noch nicht auf der Site angelegt).
- Auf Eigenschaftswerte zugreifen, die zuvor für das Steuerelement gespeichert wurden. Zu diesem Zeitpunkt wurden noch keine Eigenschaften gelesen.

Während dieses Ereignisses sollten Sie nicht:

- Standardelemente neu anordnen oder Operationen ausführen, die auf der erwarteten Größe des Steuerelements basieren. Das Steuerelement wurde noch nicht plaziert und es hat auch noch nicht seine endgültige Größe.
- Operationen ausführen, die auf Funktionen, Attributen oder Eigenschaften des Containers basieren.
- Irgend etwas anzeigen.

17.1.2 Das Terminate-Ereignis

Das `Terminate`-Ereignis wird ausgelöst, bevor das Objekt geschlossen wird. Dies ist das letzte Ereignis, das für ein Steuerelement ausgelöst wird, und (mit wenigen Ausnahmen) der letzte Code, der im Steuerelementmodul ausgeführt wird.

Während dieses Ereignisses sollten Sie:

- Sicherstellen, daß alle Objekte, die Ihr Steuerelement verwendet, korrekt freigegeben werden.

Während dieses Ereignisses können Sie:

- Alle anderen »Aufräum«-Operationen ausführen, die für Ihr Steuerelement nötig sind.

- Auf die Eigenschaften aller Standardelemente zugreifen. Alle Standardelemente existieren zu diesem Zeitpunkt noch.

Während dieses Ereignisses dürfen Sie nicht:

- Auf die `Extender`- oder `Ambient`-Eigenschaften des `UserControl`-Objekts oder eine ihrer Eigenschaften zugreifen. Ihr Steuerelement wurde vom Container bereits entfernt, wenn dieses Ereignis aufgerufen wird. Handelt es sich bei diesem Container um Visual Basic, wurde das Formular bereits aus dem Speicher entfernt, wenn dieses Ereignis aufgerufen wird.

Während dieses Ereignisses sollten Sie nicht:

- Operationen ausführen, die von Funktionen, Attributen oder Eigenschaften des Containers abhängig sind.

Aspekte beim Beenden. Es gibt noch einige andere wichtige Dinge, die für das `Terminate`-Ereignis in Betracht gezogen werden müssen. Sie können die Reihenfolge nicht wissen, in der Steuerelemente beendet werden. Das ist größtenteils sicher kein Problem, aber es kann zu einem werden, wenn eine Applikation geschlossen wird.

Was passiert, wenn Ihr Steuerelement Variablen enthält oder auf globale Variablen zugreift, die auf andere Objekte oder ActiveX-Komponenten verweisen. Normalerweise verhindert der Verweis auf das Objekt als solcher, daß es beendet wird, aber wenn eine Visual-Basic-Applikation geschlossen wird, werden alle Objekte und Steuerelemente beendet, auch wenn es noch Verweise darauf gibt. Sie können nicht feststellen, in welcher Reihenfolge, diese Objekte beendet werden.

Was passiert, wenn Sie versuchen, auf ein anderes Steuerelement oder Objekt zuzugreifen, während das `Terminate`-Ereignis für Ihr Steuerelement stattfindet? Wenn das `Terminate`-Ereignis für das andere Steuerelement noch nicht aufgerufen wurde, haben Sie wahrscheinlich kein Problem. Aber was ist, wenn das andere Objekt oder Steuerelement beendet wurde? Überraschenderweise kann die Operation funktionieren. Beispielsweise können Sie auf Eigenschaften im anderen Steuerelement zugreifen. Das bedeutet, daß der Code für dieses Steuerelement ausgeführt werden kann, nachdem sein `Terminate`-Ereignis aufgerufen wurde! Das eigentliche Problem tritt auf, wenn das andere Steuerelement Aufräumarbeiten während seines `Terminate`-Ereignisses ausgeführt hat, die bewirken könnten, daß ein Zugriff auf eine Eigenschaft fehlschlägt (beispielsweise wenn ein angeforderter Objektverweis ins Nichts zeigt). In diesem Fall wird ein Fehler aufgeworfen, wenn Ihr Steuerelement versucht, auf die Eigenschaft zuzugreifen.

Das ist nicht gerade ein übliches Szenario, aber Sie sollten es überprüfen, wenn Sie auf Objekt-Eigenschaften oder Methoden zugreifen, während ein `Terminate`-Ereignis stattfindet. Eine Möglichkeit, das Problem zu verarbeiten, ist das Fehler-Trapping während des Ereignisses.

Außerdem sollten Sie die Befehle STOP oder END von Visual Basic in Ihrer Applikation nicht verwenden, und die Menübefehle AUSFÜHREN/BEENDEN nicht aufrufen, während Sie ein Komponentenprojekt ausführen. Diese Befehle beenden Ihre Applikation oder Komponente sofort und verhindern, daß das Terminate-Ereignis des Objekts ausgelöst wird.

Dieses Problem gilt nicht für kompilierte Steuerelemente. Wenn Sie ein kompiliertes Steuerelement in ein Projekt laden und das Projekt mit dem Befehl AUSFÜHREN/BEENDEN beenden, wird das Terminate-Ereignis für das Formular des Projekts nicht ausgelöst. Das Terminate-Ereignis für das kompilierte Ereignis wird jedoch ausgelöst. Das kompilierte Steuerelement wird nämlich aus der Perspektive Ihres Projekts nicht als Visual-Basic-Code betrachtet, und die Befehle STOP und END beziehen sich nur auf die Ausführung von VB-Code.

17.2 Persistenz von Eigenschaften – Grundlagen

Einer der Unterschiede zwischen ActiveX-Komponenten und ActiveX-Steuerelementen ist, daß von Ihnen entwickelte Steuerelemente zur Entwurfszeit in der Umgebung eines Entwicklers ausgeführt werden können, der diese Steuerelemente einsetzt. Das bedeutet, ActiveX-Steuerelemente bieten den Entwicklern die Möglichkeit, Eigenschaften zur Entwurfszeit zu setzen. Diese Eigenschaftswerte können von der Entwicklungsumgebung gespeichert werden, egal, welches Format diese zum Speichern der aktuell entwickelten Applikationen nutzt. Die Eigenschaften können wieder geladen werden, wenn das Steuerelement erneut erzeugt wird. Dieser Prozeß, Eigenschaften zu speichern und zu laden, wird auch als Eigenschaftspersistenz bezeichnet.

Betrachten Sie das einfache Textfeld. Tabelle 17.1 zeigt die typische Eigenschaftspersistenz für das Textfeld. Seine Operationen sollten Ihnen bereits vertraut sein.

Ereignis während der Lebenszeit eines Steuerelements	Eigenschaftsoperationen
Das Steuerelement wird auf ein Formular gezeichnet.	Die Eigenschaften des Steuerelements werden initialisiert. Die Text-Eigenschaft wird mit dem Namen des Steuerelements initialisiert.
Das Projekt oder die Datei, die das Steuerelement enthalten, werden gespeichert.	Die Eigenschaften des Steuerelements werden in der .FRM-Datei abgelegt (bei VB).
Das Projekt wird ausgeführt.	Die Eigenschaften des Steuerelements werden im Speicher abgelegt, wenn das Steuerelement zerstört wird. Die Eigenschaften werden in das Steuerelement zurückgelesen, wenn es in der Laufzeitumgebung erzeugt wird.

Tab. 17.1: Eigenschaftspersistenz bei einem Textfeld

Ereignis während der Lebenszeit eines Steuerelements	Eigenschaftsoperationen
Das Projekt wird unterbrochen.	Die Eigenschaften des Steuerelements werden nicht geschrieben, wenn das Steuerelement unterbrochen wird, weil die Laufzeitwerte nicht persistent gemacht werden. Die Eigenschaften des Steuerelements werden aus dem Speicher gelesen (wo sie zuletzt abgelegt wurden), wenn das Steuerelement in der Entwurfszeit-Umgebung erzeugt wird.
Das Projekt wird kompiliert.	Die Eigenschaften des Steuerelements werden in die Programmdatei geschrieben.
Die Programmdatei wird ausgeführt.	Die Eigenschaften des Steuerelements werden aus der Programmdatei gelesen, wenn das Steuerelement erzeugt wird.

Tab. 17.1: Eigenschaftspersistenz bei einem Textfeld

Wie Sie sehen, müssen für Eigenschaften drei verschiedene Operationen unterstützt werden. Sie müssen initialisiert werden, sie müssen gespeichert werden, und sie müssen geladen werden. Ein Steuerelement weiß anhand von drei Ereignissen vom `UserControl`-Objekt, wann es diese drei Operationen ausführen soll: `InitProperties`, `ReadProperties` und `WriteProperties`.

17.2.1 Die Standardwerte

Manchmal scheint der Umgang mit den Eigenschaften von ActiveX-Steuerelementen etwas schwierig zu sein, weil zwei Arten von Initialisierung berücksichtigt werden müssen. Diese Situation entsteht, weil die Eigenschaften eines ActiveX-Steuerelements Standardwerte haben können. Ich werde Ihnen das gleich erklären.

Offensichtlich muß nicht jede Eigenschaft in Ihrem Steuerelement persistent gehalten werden. Die `hWnd`-Eigenschaft beispielsweise gibt den aktuellen Fenster-Handle des Steuerelements zurück (außer für »schlanke« Steuerelemente, die keine Fenster-Handles haben). Weil dieser Handle zur Laufzeit festgelegt wird, und sich bei jedem Mal ändert, wenn das Steuerelement erzeugt wird, wäre es unsinnig, diese Eigenschaft in einer Projekt- oder Programmdatei zu speichern.

Für die Eigenschaften, die persistent sein sollen, müssen Sie eines berücksichtigen: Das Speichern und Laden von Eigenschaftswerten dauert. Es dauert lang. Der genaue Mechanismus, wie man Eigenschaften persistent macht, wird in Kapitel 19 beschrieben, aber hier sollten Sie einfach nur eine `.FRM`-Datei betrachten (öffnen Sie eine mit einem Texteditor, beispielsweise Notepad). Die Eigenschaften von Steuerelementen werden alle zusammen mit dem Eigenschaftsnamen und einer Textdarstellung des Werts aufgelistet. Binäre Werte (beispielsweise

Bilder) werden in der .FRX-Datei abgelegt. Die Konvertierung in und von einer Textdarstellung eines Werts ist offensichtlich ein relativ langsamer Prozeß, ebenso wie das Öffnen von Dateien, und die Suche, wo in der Datei sich ein bestimmter Eigenschaftswert befindet.

Die Entwickler der ActiveX-Spezifikation erkannten, daß die Autoren von Steuerelementen häufig Eigenschaften mit absehbar gebräuchlichen Werten belegen. Das bedeutet, daß die Eigenschaftswerte von den Entwicklern häufig überhaupt nicht geändert werden. Wenn ein Steuerelement eine Eigenschaft ohnehin mit dem korrekten Wert initialisiert, warum sollte man dann den Aufwand betreiben, es zu speichern und dann wieder zu laden?

Deshalb kann jeder Eigenschaft in Ihrem Steuerelement ein Standardwert zugewiesen werden. Wenn die Eigenschaft geschrieben werden soll, prüft Visual Basic, ob die Eigenschaft den Standardwert hat. Ist das der Fall, muß die Eigenschaft nicht gespeichert werden. Wird das Steuerelement neu geladen, versucht Visual Basic nicht, die Eigenschaft zu lesen (und das kann es auch gar nicht, weil sie gar nicht erst gespeichert wurde). Statt dessen weist VB den angegebenen Standardwert zu.

Betrachten Sie noch einmal Tabelle 17.1 Wenn das Steuerelement zum ersten Mal auf ein Formular gezeichnet wird, ist die Text-Eigenschaft der Name des Steuerelements (z.B. Text1). Aber was ist die Standardeigenschaft? Es macht keinen Sinn, den Namen des Steuerelements zu verwenden, weil dieser für jedes Steuerelement anders ist. Der leere String ist eine viel bessere Wahl, schon alleine deshalb, weil es durchaus üblich für Textfelder ist, sie beim ersten Anzeigen eines Formulars leer darzustellen. Aber wie wird es auf Text1 gesetzt, wenn ein Steuerelement einem Formular zum ersten Mal hinzugefügt wird? Offensichtlich muß es ein separates Ereignis geben, anzuzeigen, wann ein Steuerelement zum ersten Mal angelegt wird. Das ist das InitProperties-Ereignis.

17.2.2 Das InitProperties-Ereignis

Die Instanz eines Steuerelements erhält nur dann das InitProperties-Ereignis, wenn sie zum ersten Mal erzeugt wird (beispielsweise, wenn sie auf ein Formular gezeichnet wird).

Während dieses Ereignisses sollten Sie:

- Persistente Eigenschaften auf ihren Ausgangswert setzen, nicht auf ihre Standardwerte. Standardwerte werden während des ReadProperties-Ereignisses gesetzt. Bei einem Standard-Textfeld ist der Standardwert für die Text-Eigenschaft der leere String. Der Ausgangswert ist der Name des Steuerelements.
- Initialisierungsroutinen für das Steuerelement aufrufen, die ausgeführt werden müssen, nachdem ein Steuerelement erzeugt wurde. Sie können diese Art der Initialisierung in einer separaten Funktion unterbringen, weil Sie möglicherweise dieselben Operationen nach dem ReadProperties-Ereignis ausführen müssen.

- Nichtpersistente Eigenschaften auf ihre Ausgangswerte setzen. Die Idee der »Standard«-Werte existiert für nichtpersistente Eigenschaften nicht. Möglicherweise könnten Sie diesen Code in einer separaten Routine ablegen, weil er normalerweise auch während des `ReadProperties`-Ereignisses aufgerufen wird.

Während dieses Ereignisses können Sie:

- Die meisten der vom Container abhängigen Initialisierungen vornehmen. Das Steuerelement befindet sich bereits auf der Site, wenn dieses Ereignis auftritt, die `Extender`- und `Ambient`-Eigenschaften sind also gültig.

Während dieses Ereignisses sollten Sie nicht:

- Irgend etwas anzeigen. Das Steuerelement ist zu diesem Zeitpunkt noch nicht sichtbar.
- Operationen ausführen, die von der Größe des Steuerelements abhängig sind. Das Steuerelement und der Container haben möglicherweise noch nicht die endgültige Größe.
- Operationen ausführen, die voraussetzen, daß das Steuerelementfenster auf dem Container bereits existiert. Beispiele dafür sind unter anderem API-Funktionen zur Manipulation von Fenstern. Das Steuerelementfenster befindet sich während dieses Ereignisses noch nicht wirklich auf dem Container (wenn es das auch, vom Standpunkt der Objektrelationen aus gesehen, bereits tun müßte).

17.2.3 Das ReadProperties-Ereignis

Das `ReadProperties`-Ereignis tritt auf, wenn ein Steuerelement erzeugt wird, außer wenn es zum ersten Mal auf einem Formular angelegt wird.

Während dieses Ereignisses sollten Sie:

- Das `PropBag`-Objekt verwenden, um die Werte persistenter Eigenschaften zu lesen. Der eigentliche Mechanismus zum Lesen und Schreiben von Eigenschaften wird in Kapitel 19 genauer beschrieben.
- Alle Routinen für die Initialisierung des Steuerelements aufrufen, die ausgeführt werden müssen, nachdem ein Steuerelement erzeugt wurde. Sie könnten diese Art Initialisierung in einer separaten Funktion unterbringen, weil es sein kann, daß Sie dieselben Operationen nach dem `InitProperties`-Ereignis ausführen müssen.
- Die nichtpersistenten Eigenschaften auf ihre Ausgangswerte setzen. Das Prinzip der Standardwerte gibt es für nichtpersistente Eigenschaften nicht. Sie könnten diesen Code in einer separaten Routine ablegen, weil er in der Regel auch während des `InitProperties`-Ereignisses aufgerufen wird.

Während dieses Ereignisses können Sie:

- Die meisten der vom Container abhängigen Initialisierungen ausführen. Das Steuerelement ist bereits positioniert, wenn dieses Ereignis auftritt, die `Extender`- und `Ambient`-Eigenschaften sind also gültig.

Während dieses Ereignisses sollten Sie nicht:

- Irgend etwas anzeigen. Das Steuerelement ist zu diesem Zeitpunkt noch nicht sichtbar.
- Operationen ausführen, die von der Größe des Steuerelements abhängig sind. Das Steuerelement und der Container haben möglicherweise noch nicht die endgültige Größe.
- Operationen ausführen, die voraussetzen, daß das Steuerelementfenster auf dem Container bereits existiert. Beispiele dafür sind unter anderem API-Funktionen zur Manipulation von Fenstern. Das Steuerelementfenster befindet sich während dieses Ereignisses noch nicht wirklich auf dem Container (wenn es das auch, vom Standpunkt der Objektrelationen aus gesehen, bereits tun müßte).

17.2.4 Das WriteProperties-Ereignis

Diese Funktion wird aufgerufen, wenn ein Steuerelement in einer Projektdatei oder einer Programmdatei gespeichert oder im Speicher abgelegt wird, um den Eintritt in den Ausführungsmodus vorzubereiten. Sie können nicht feststellen, welche dieser Operationen ausgeführt wird, und das ist auch nicht nötig. Die Funktionen, die Visual Basic für die Eigenschaftspersistenz bereitstellt, leiten die Daten automatisch an die richtige Stelle weiter.

Während dieses Ereignisses sollten Sie:

- Mit Hilfe des `PropBag`-Objekts die Werte persistenter Eigenschaften speichern. Der Mechanismus, wie Eigenschaften gelesen und geschrieben werden, wird in Kapitel 19 genauer beschrieben.

Während dieses Ereignisses sollten Sie nicht:

- Irgendwelche Operationen für die Terminierung ausführen. Es gibt Situationen, in denen dieses Ereignis nicht ausgelöst wird.

Es gibt zwei Situationen, die Sie berücksichtigen sollten, in denen dieses Ereignis nicht ausgelöst wird:

- Wenn Sie in der VB-Umgebung vom Ausführungsmodus in den Entwurfsmodus wechseln.
- Wenn Visual Basic glaubt, es hätte sich kein Eigenschaftswert geändert.

Beachten Sie, daß die Entscheidung, wann das `WriteProperties`-Ereignis aufgerufen wird, dem Container überlassen bleibt. Container, die den Entwurfsmodus nicht unterstützen, rufen diese Funktion möglicherweise nie auf.

17.2.5 Die PropertyChanged-Methode

Betrachten wir ein Steuerelement mit einer einzigen `Text`-Eigenschaft. Immer wenn Sie vom Entwurfsmodus in den Ausführungsmodus wechseln, muß Visual Basic die persistenten Eigenschaftswerte im Speicher ablegen, so daß sie wieder geladen werden können, sobald das Steuerelement im Ausführungsmodus erzeugt wird. Aber warum sollte man alle Eigenschaften jedesmal speichern, auch wenn sie sich nicht geändert haben? Könnte Visual Basic die Eigenschaftswerte nicht in den Cache stellen und nur diejenigen aktualisieren, die sich geändert haben, um so die Performance zu erhöhen?

Natürlich! Visual Basic schreibt nur Eigenschaften, von denen es weiß, daß sie sich geändert haben. Das bedeutet, Sie müssen Visual Basic benachrichtigen, wenn Sie einen Eigenschaftswert ändern. Das erfolgt über die `PropertyChanged`-Methode des `UserControl`-Objekts. Diese Methode nimmt den Namen der geänderten Eigenschaft als Parameter entgegen.

Container verwenden diese Methode auf unterschiedliche Weisen:

- Um zu verfolgen, welche Eigenschaften geschrieben werden müssen, wenn das Projekt gespeichert wird, oder wenn vom Entwurfmodus in den Ausführungsmodus gewechselt wird.
- Um den Wert der Eigenschaft auf der Eigenschaftsseite der Entwicklungsumgebung zu aktualisieren. Beachten Sie, daß es möglich (aber selten) ist, daß nichtpersistente Eigenschaften auf einer Eigenschaftsseite erscheinen. In diesem Fall sollten Sie `PropertyChanged` für die Eigenschaft aufrufen, auch wenn diese nicht persistent ist.
- Wenn das Steuerelement die Datenbindung unterstützt, wird diese Methode verwendet, um das System darüber zu informieren, daß sich die gebundene Eigenschaft geändert hat und in der Datenbank möglicherweise aktualisiert werden muß. Diese Situation hat man häufig, wenn sich der Container auch im Ausführungsmodus befindet.

Es passiert nichts, wenn die `PropertyChanged`-Methode für Eigenschaften aufgerufen werden, für die es nicht nötig ist (außer vielleicht eine leichte Performanceverschlechterung). Wird `PropertyChanged` jedoch nicht aufgerufen, wenn das erforderlich ist, kann es passieren, daß die Einstellungen der Entwurfszeit nicht geschrieben wird, die VB-Eigenschaftsseite sich nicht selbst aktualisiert, wenn Werte vom Steuerelement geändert werden, oder gebundene Eigenschaften die zugrundeliegenden Datenbanken nicht aktualisieren. Fragen Sie sich also immer, ob die Methode aufgerufen werden soll, wenn eine Eigenschaft geändert wird.

Noch ein Tip. Angenommen, Sie haben eine typische, in einem Steuerelement definierte Eigenschaft:

```
'Eigenschaftsvariablen:
Dim m_myprop As Variant

Public Property Get myprop() As Variant
    myprop = m_myprop
End Property

Public Property Let myprop(ByVal New_myprop As Variant)
    m_myprop = New_myprop
    PropertyChanged "myprop"
End Property
```

Nun möchten Sie die `myprop`-Eigenschaft an anderer Stelle in Ihrem Steuerelement ändern. Ihr Steuerelement kann direkt auf die Variable `m_myprop` zugreifen, aber vielleicht ist es sinnvoller, statt dessen die `myprop`-Eigenschaft des Steuerelements zu verwenden. Wenn Sie sich angewöhnen, existierende Eigenschaften Ihres Steuerelements zu verwenden, ist es nicht mehr so gefährlich, den Aufruf von `PropertyChanged` zu vergessen. Außerdem genießen Sie die Vorteile der Fehlerprüfung, die Sie den Eigenschaftsprozeduren hinzugefügt haben.

17.3 Ereignisse und Eigenschaften zum Plazieren und Anzeigen

Beim sogenannten »Siting« wird ein Steuerelement auf einem Container plaziert. Dieses Siting und die Anzeige sind ein Bereich, in dem ein `UserControl`-Objekt möglicherweise vom Status und vom Typ des Containers abhängig ist.

Während der Initialisierung des Steuerelements treten die folgenden drei Ereignisse auf:

1. `Resize` – Zeigt an, daß sich die Größe des Steuerelements geändert hat.
2. `Show` – Gibt an, daß das Steuerelement plaziert und/oder sichtbar ist.
3. `Paint` – Gibt an, daß es Zeit für Sie ist, den Inhalt des Steuerelements zu zeichnen.

Wir werden sie nun einzeln genauer betrachten.

17.3.1 Das Resize-Ereignis

Das `Resize`-Ereignis gibt an, daß sich die Größe des Steuerelements geändert hat. Dieses Ereignis tritt auf, nachdem ein Steuerelement auf dem Container plaziert wurde und alle Eigenschaften gelesen wurden. Außerdem tritt es auf, wenn das Steuerelement erzeugt wird.

Während dieses Ereignisses können Sie:

- Standardelemente abhängig von der endgültigen Größe des Steuerelements neu anordnen (endgültig zumindest so lange, bis die Größe des Steuerelements wieder verändert wird).
- Andere Operationen ausführen, die von der Größe des Steuerelements oder des Containers abhängig sind.

Während dieses Ereignisses sollten Sie nicht:

- Ihr Steuerelement neu zeichnen. Warten Sie auf das `Paint`-Ereignis.
- Operationen ausführen, die von den Eigenschaftswerten des Steuerelements abhängig sind. Die Ereignisse `InitProperties` oder `ReadProperties` finden abhängig vom Container während dieses Ereignisses statt oder nicht.

Das Steuerelement `ch17ctlc` im Projekt `ch17ctls1` demonstriert einige der Funktionen zur Größenänderung von Standardelementen und Steuerelementen. Beim Experimentieren mit diesem Steuerelement werden Sie die folgenden Dinge feststellen.

Erstens, Sie erhalten möglicherweise kein `Resize`-Ereignis für Standardelemente, während das Steuerelement erzeugt wird. Wenn Sie ein `Initialize`-Ereignis in Ihrem Steuerelement erhalten, wurden nämlich die Standardelemente bereits plaziert und auf Ihrem Steuerelement angelegt. Betrachten Sie dazu die `Debug`-Nachricht vom Standardelement `picture1`. Es gibt keine, wenn das Steuerelement erzeugt wird, aber wenn Sie auf das Steuerelement doppelklicken, während es in einem Container ausgeführt wird (nicht im Entwurfsmodus), sehen Sie das Ereignis, wenn die Größe des Steuerelements durch den folgenden Code geändert wird:

```
Private Sub Picture1_DblClick()
    ' Resize-Ereignis von picture1 erzeugen
    Picture1.Width = Picture1.Width + 1
 End Sub
```

Zweitens ist das `Resize`-Ereignis vielleicht nicht auf allen Containern zur Entwurfszeit zuverlässig. Die Container sind nicht gezwungen, ein Steuerelement während der Entwurfszeit zu plazieren. Dies wird im Abschnitt über das `Paint`-Ereignis noch genauer beschrieben.

Schließlich sollten Sie sehr vorsichtig vorgehen, um keine Abhängigkeiten für `InitProperties`- und `ReadProperties`-Ereignisse in bezug auf das `Resize`-Ereignis zu erzeugen. Diese sind wesentlich vom Container abhängig. In Visual Basic 5 beispielsweise werden Eigenschaften initialisiert, bevor das `Resize`-Ereignis auftritt. In Visual Basic 6 tritt das `Resize`-Ereignis manchmal auf, bevor Eigenschaften gelesen werden.

Feste Größen für Steuerelemente. Zunächst möchte ich ganz allgemein sagen, daß Sie es dem Entwickler ermöglichen sollten, die Größe Ihres Steuerelements zu ändern. Sie sollten Ihr Steuerelement nie auf dem Container verschieben, aber es gibt einige Situationen, wo Sie vielleicht eine feste Größe für Ihr Steuerelement brauchen. Beispielsweise könnte ein Steuerelement, das zur Laufzeit unsichtbar ist, zur Entwurfszeit durch ein einfaches Icon dargestellt werden. Oder Sie wollen die Größe eines Steuerelements der Größe einer bestimmten Bitmap anpassen.

Um die Größe eines Steuerelements zu ändern, verwenden Sie die `Size`-Methode, wie im folgenden Beispiel aus dem Steuerelement `ch17ctlc` gezeigt:

```
     Private m_AutoResize As Boolean

     Public Property Let AutoResize(ByVal bResize As Boolean)
        m_AutoResize = bResize
        PropertyChanged "AutoResize"
        SetDefaultSize
     End Property

     Public Property Get AutoResize() As Boolean
        AutoResize = m_AutoResize
     End Property

     Private Sub UserControl_ReadProperties(PropBag As PropertyBag)
        m_AutoResize = PropBag.ReadProperty("AutoResize", False)
     End Sub

     Private Sub UserControl_WriteProperties(PropBag As PropertyBag)
        PropBag.WriteProperty "AutoResize", m_AutoResize, False
     End Sub


     Private Sub UserControl_Resize()
        Debug.Print "C: resize"
        ' War in VB5 ausreichend; für VB6 ist das Show-Ereignis erforder-
lich
     SetDefaultSize
     End Sub

Private Sub UserControl_Show()
        Debug.Print "C: Show"
        Debug.Print "Parent is " & Hex$(GetParent(hwnd))
        SetDefaultSize
End Sub


     Private Sub SetDefaultSize()
```

```
    if m_AutoResize then
        Size 200 * Screen.TwipsPerPixelX, 100 * Screen.TwipsPerPixelY
    EndIf
End Sub
```

Versuchen Sie, dem Steuerelement zur Entwurfszeit eine neue Größe zu geben, wobei die `AutoResize`-Eigenschaft zunächst auf `False` und dann auf `True` gesetzt ist, um zu sehen, wie das Ganze funktioniert. Die `Size`-Methode erzeugt keine zusätzlichen `Resize`-Ereignisse, wenn das Steuerelement bereits die vorgegebene Größe hat. Die Funktion `SetDefaultSize` muß ebenfalls während des `UserControl_Show`-Ereignisses aufgerufen werden. Warum? Weil das `Resize`-Ereignis auftreten kann, bevor die Variable `m_AutoResize` geladen ist. Nur das `Show`-Ereignis findet garantiert nach dem `ReadProperties`-Ereignis statt. Warum wird die Eigenschaft immer noch während des `Resize`-Ereignisses aufgerufen? Weil nur das `Resize`-Ereignis aufgeworfen wird, wenn die Größe des Steuerelements durch externe Faktoren geändert wird (beispielsweise durch den Benutzer).

17.3.2 Die Ereignisse Show und Hide

Das `Show`-Ereignis tritt auf, wenn das Fenster für das Steuerelement plaziert oder das Steuerelement auf einem Container sichtbar gemacht wird. Diese Erklärung mag seltsam klingen, aber es gibt einen Grund für diese Form der Beschreibung. Erstens wird davon ausgegangen, daß jedes Steuerelement, das ein Fenster hat, das Container-Fenster als Elternfenster angeben muß. Außerdem wird damit gefordert, daß wir sorgfältig definieren, was es für ein Steuerelement heißt, sichtbar zu sein.

Zunächst wollen wir uns auf Steuerelemente konzentrieren, die Fenster haben. In Visual Basic erzeugen Sie fensterlose Steuerelemente, indem Sie die `Windowless`-Eigenschaft auf `True` setzen. Für nichtfensterlose Steuerelemente hat das Steuerelement-Fenster nicht immer den Container als Elternfenster. Wurde beispielsweise ein Steuerelement noch nicht auf dem Container plaziert, kann das Fenster zwar existieren, aber es hat ein anderes Elternfenster (eines, das Visual Basic erzeugt, um Steuerelementfenster aufzunehmen, bis sie benötigt werden). Die erste Bedingung für das `Show`-Ereignis ist also, daß das Steuerelement plaziert wird.

Als nächstes müssen Sie ein bißchen über die Sichtbarkeit unter Windows erfahren. Angenommen, Sie haben ein Elternfenster und ein Kindfenster, und beide sind sichtbar. Wenn Sie das Kindfenster verbergen, ändert es seinen Status in unsichtbar. (Für diejenigen unter Ihnen, die das Win32-API kennen: Der Sichtbarkeitsstatus eines Fensters wird durch den Stil `WS_VISIBLE` bestimmt.) Wenn Sie das Elternfenster verbergen, werden beide verborgen, aber das Kindfenster behält seinen sichtbaren Status. Auf diese Weise wird es automatisch angezeigt, wenn das Elternfenster wieder sichtbar gemacht wird. Mit anderen Worten, ein Kindfenster ist nur dann wirklich sichtbar, wenn es selbst und auch das Elternfenster sichtbar ist. Ein sichtbares Fenster kann dennoch verdeckt sein, wenn es sich bei-

spielsweise hinter einem anderen Fenster befindet, aber das hat keinen Einfluß auf den internen Status des Fensters. Tabelle 17.2 zeigt die möglichen Sichtbarkeitszustände eines Fensters, und ob Sie es sehen oder nicht.

Status des Elternfensters	Status des Kindfensters	Befindet sich ein anderes Fenster oben?	Das Fenster, das Sie sehen
Sichtbar	Sichtbar	Nein	Beide
Sichtbar	Verborgen	Nein	Elternfenster
Verborgen	Sichtbar	Nein	Keines
Verborgen	Verborgen	Nein	Keines
Beliebig	Beliebig	Ja	Keines

Tab. 17.2: Sichtbarkeitszustände eines Kindfensters in einem Elternfenster

Das `Show`-Ereignis tritt auf, wenn das Kindfenster sichtbar wird, oder wenn es zum ersten Mal auf einem Container plaziert wird. Die Visual-Basic-Dokumentation schlägt vor, daß dieses Ereignis unmittelbar nach der Plazierung des Steuerelements auftreten soll. Tests haben jedoch gezeigt, daß das Steuerelement bei einigen Containern bereits während des `Resize`-Ereignisses plaziert wird, das vor dem `Show`-Ereignis auftritt. Nichtsdestotrotz sollten Sie nie davon ausgehen, daß das Steuerelementfenster plaziert wurde, bevor das `Show`-Ereignis aufgeworfen wurde.

Es tritt auch auf, wenn Sie ein Formular laden, das ein Steuerelement enthält, aber zeigt das Formular nicht an. Es tritt immer auf, wenn Sie die `Visible`-Eigenschaft des Steuerelements auf `True` setzen (so daß sein Status auf sichtbar gesetzt wird). Es tritt immer dann auf, wenn ein Internet-Browser eine Seite zurückgibt, die das Steuerelement enthält.

Es tritt nicht auf, wenn Sie den Sichtbarkeitsstatus des Containers ändern. Wenn Sie also den Container verbergen oder minimieren und ihn dann wieder anzeigen, wird das Show-Ereignis nicht erzeugt. Es tritt nicht zur Entwurfszeit für Container auf, die keine Steuerelementfenster plazieren, beispielsweise Visual Basic 4. Das Szenario wird später im Abschnitt über das `Paint`-Ereignis genauer beschrieben.

Der Prozeß, ein Steuerelement zu plazieren, kann etwas verwirrend sein, wir wollen ihn also kurz noch einmal betrachten:

- Wenn das `Initialize`-Ereignis aufgeworfen wird, ist das Steuerelement noch nicht plaziert.
- Wenn die Ereignisse `ReadProperties` und `Resize` auftreten, wird das Steuerelement plaziert und Sie können auf die `Parent`-Eigenschaft verweisen – die Objektbeziehung zwischen Container und Steuerelement existiert bereits. Das Fenster für das Steuerelement ist jedoch noch kein Kindfenster des Containerfensters.

- Wenn das `Show`-Ereignis auftritt, ist das Steuerelement endgültig plaziert und die Fensterbeziehung wurde eingerichtet.

Diese Abfolge gilt auch für fensterlose Steuerelemente, außer daß es dabei kein Steuerelementfenster gibt, das zum Kindfenster des Containerfensters gemacht wird. Das können Sie nachvollziehen, indem Sie sich das Steuerelement `ctl17wsl` im Beispielprojekt `ch17wsl.vbp` ansehen.

Während des `Show`-Ereignisses können Sie:

- Beliebige Operationen ausführen, die ein Web-basiertes Steuerelement tun kann, wenn der Benutzer auf die Seite mit dem Steuerelement zurückkehrt.
- Eine Variable setzen, um damit anzuzeigen, daß Ihr Steuerelement bereits auf einem Container plaziert wurde. Einige Entwicklungsumgebungen plazieren keine Steuerelemente zur Entwurfszeit. Das wird später im Abschnitt über das `Paint`-Ereignis genauer beschrieben.
- Wenn dieses Ereignis auftritt (zumindest in VB5), wurden alle anderen Steuerelemente auf einem Formular plaziert und das `Load`-Ereignis ausgeführt. Das unterscheidet sich vom `ReadProperties`-Ereignis, das nur garantiert, daß Ihr Steuerelement plaziert wurde. Wenn Ihr Steuerelement mit anderen Steuerelementen auf einem Formular interagieren muß (was nicht empfohlen ist), dann ist dies ein sicherer Zeitpunkt dafür, obwohl dieses Verhalten nicht für jeden Container garantiert werden kann.

Das `Show`-Ereignis bietet auch eine gute Gelegenheit, Initialisierungen vorzunehmen, die von der Größe des Containers und den Eigenschaften abhängig sind. Weil die relative Reihenfolge beim Laden von Eigenschaften und die Größenanpassung von Steuerelementen nicht definiert ist, ist dies die letzte Möglichkeit, diese Initialisierungen vor dem `Paint`-Ereignis auszuführen. Eine gebräuchliche Technik ist, eine `Flag`-Variable einzusetzen, die anzeigt, daß das Steuerelement initialisiert wird. Setzen Sie dieses Flag während des `Initialize`-Ereignisses des Steuerelements. Fragen Sie es während des `Show`-Ereignisses ab – wenn es `True` ist, setzen Sie die endgültige Initialisierung fort und löschen das Flag. Auf diese Weise können Sie sicherstellen, daß diese Art der Initialisierung nur einmal stattfindet, nämlich wenn das Steuerelement zum ersten Mal angezeigt wird.

Das `Hide`-Ereignis ist die Umkehrung des `Show`-Ereignisses. Es tritt auf, wenn das Steuerelement-Fenster vom Container entfernt wird, und immer, wenn die `Visible`-Eigenschaft für das Steuerelement auf `False` gesetzt wird.

17.3.3 Das Paint-Ereignis

Das `Paint`-Ereignis tritt auf, wenn Ihr Steuerelement oder ein Teil davon gezeichnet werden muß. Das `UserControl_Paint`-Ereignis tritt vor den `Paint`-Ereignissen der Standardelemente auf. Das ist eines der wichtigsten Ereignisse für benutzerdefinierte Steuerelemente, aber es ist relativ unwichtig für Steuerelemente, die einfach Bestandteil anderer Steuerelemente sind.

Steuerelemente, die unter Visual Basic erzeugt wurden, unterstützen viele der Standardeigenschaften, die das Zeichnen beeinflussen, unter anderem `AutoRedraw` und `ClipControls`. Im Verzeichnis für Kapitel 17 auf Ihrer CD-ROM finden Sie das Beispielprojekt `clipctls.vbp`, das Ihnen erlaubt, ein bißchen mit der `ClipControls`-Eigenschaft zu experimentieren. Diese Eigenschaften sollten Ihnen von der allgemeinen VB-Programmierung her vertraut sein.

Beachten Sie, daß das `Paint`-Ereignis nicht für Steuerelemente auftritt, deren `AutoRedraw`-Eigenschaft gleich `True` ist. Ich empfehle Ihnen, `AutoRedraw` in Ihren Steuerelementen möglichst nicht auf `True` zu setzen. Es bedeutet einen wesentlichen Overhead für Ihr Steuerelement und hat nur wenig Nutzen. Wenn Sie ein persistentes Bild brauchen, kann es viel effizienter sein, mit Hilfe der Win32-API-Techniken selbst ein solches zu erzeugen.

Mit Hilfe der Unterklassenbildung (»Subclassing«) erkennen Sie den Aktualisierungsbereich für ein Steuerelement, wenn die Zeichnung sehr komplex ist und Sie nur die Teile des Steuerelements neu zeichnen wollen, die sich wirklich geändert haben. Dazu müssen die `WM_PAINT`-Nachricht für das Steuerelement aufgefangen und der Aktualisierungsbereich gespeichert werden. Diese Technik wird in Kapitel 22 genauer beschrieben.

Während des `Paint`-Ereignisses können Sie:

- Beliebige Zeichenoperationen ausführen.

Während des `Paint`-Ereignisses sollten Sie nicht:

- Die Größe Ihres Steuerelements ändern oder die `Refresh`-Methode aufrufen. Diese Operationen können ein weiteres `Paint`-Ereignis auslösen und damit zu einer Kette von `Paint`-Ereignissen führen, die wiederum einen Stapelüberlauf verursachen können.
- Eigenschaften ändern, die ein weiteres `Paint`-Ereignis auslösen könnten (beispielsweise die `BackColor`-Eigenschaft).

Hinweise zur Plazierung und zum Zeichnen. Es gibt zwei grundlegende Techniken, auf ein Steuerelement zu zeichnen: Visual-Basic-Befehle und Win32-API-Befehle. API-Funktionen machen erforderlich, daß Sie in ein bestimmtes Objekt zeichnen, den sogenannten Gerätekontext. API-Funktionen können nicht für fensterlose Steuerelemente verwendet werden. In diesem Buch ist keine ausführliche Beschreibung möglich, wie das genau geht, aber Sie finden fast alle Details in *The Visual Basic 5.0 Programmer's Guide to the Win32 API*. Ich weiß, daß die Leser, die mit dem Win32-API nicht vertraut sind, sich von den folgenden Beschreibungen überfordert fühlen werden, und ich möchte mich dafür entschuldigen, aber die Information ist, wenn auch etwas komplex, doch wichtig für alle, die mehrere Container unterstützen wollen.

Das `UserControl_Paint`-Ereignis sehen Sie hier:

```
Private Sub UserControl_Paint()
   Dim usedc&
   Dim pt As POINTAPI

   Debug.Print "C: control paint"

   usedc = GetDC(UserControl.hwnd)  ' Den vom Fenster erhaltenen
                                    ' Gerätekontext verwenden
   Call MoveToEx(usedc, 0, 5, pt)
   Call LineTo(usedc, 100, 5)
   Call ReleaseDC(UserControl.hwnd, usedc)
   UserControl.Line (5, 15)-(100, 15), &HFF
   Call MoveToEx(UserControl.hdc, 10, 25, pt)
   Call LineTo(UserControl.hdc, 100, 25)
End Sub
```

Bei der Verwendung von API-Funktionen stellt sich die Frage, wie man den Gerätekontext erhält. Dieser Code demonstriert zwei Methoden dafür – aus dem Fenster-Handle oder aus der `hDC`-Eigenschaft des `UserControl`-Objekts.

Ein ähnlicher Code wird für das `Paint`-Ereignis des Standard-Bildfelds bereitgestellt, nämlich wie folgt:

```
Private Sub Picture1_Paint()
   Dim usedc&
   Dim pt As POINTAPI

   Debug.Print "C: picture paint"
   ' Jetzt das Bildfeld
   usedc = GetDC(Picture1.hwnd)  ' Den vom Fenster erhaltenen
                                 ' Gerätekontext verwenden
   Call MoveToEx(usedc, 0, 5, pt)
   Call LineTo(usedc, 100, 5)
   Call ReleaseDC(Picture1.hwnd, usedc)
   Picture1.Line (5, 15)-(100, 15), &HFF
   Call MoveToEx(Picture1.hdc, 10, 25, pt)
   Call LineTo(Picture1.hdc, 100, 25)
End Sub
```

Diese Ereignisse zeichnen drei horizontale Linien, zwei schwarze oben und unten, und in der Mitte eine rote. Wenn Sie dieses Steuerelement auf einem Formular unter VB 5 oder VB 6 plazieren, funktioniert alles wunderbar, sowohl zur Entwurfszeit als auch zur Laufzeit. Versuchen Sie aber jetzt, das Steuerelement auf einem VB4-Formular anzulegen. (Registrieren Sie die OCX-Datei mit `Regsvr32` oder kompilieren Sie das Projekt neu.) Sie sehen, daß zur Entwurfszeit die obere Linie im `UserControl` und alle drei Linien im Bildfeld fehlen.

Warum? Weil ein Container ein Steuerelement zur Entwurfszeit nicht unbedingt plazieren und aktivieren muß – und Visual Basic 4 tut das eben nicht. Das enthaltene Bildfeld wird überhaupt nicht verarbeitet. Sein Fenster existiert irgendwo anders im System, und VB4 ist nicht intelligent genug, um zu erkennen, daß es an eine andere Position zeichnen muß.

Aber das Zeichnen von `UserControl` funktioniert zum Teil. Der `Line`-Befehl funktioniert, und das Zeichnen mit der `hDC`-Eigenschaft des `UserControl`-Objekts funktioniert. Warum? Weil die ActiveX-Spezifikation diese Situation berücksichtigt und dem Container einen Mechanismus bietet, einen Gerätekontext bereitzustellen, der zum Zeichnen verwendet wird, wenn ein Steuerelement nicht plaziert wird. Visual Basic ist intelligent genug, diesen Gerätekontext als `hDC`-Eigenschaft bereitzustellen und ihn für die Zeichenbefehle unter Visual Basic zu verwenden. Zu welchem Fenster gehört dieser Gerätekontext? Zum Container-Formular! Ihr Steuerelement denkt, es würde in das `UserControl`-Fenster zeichnen, aber in Wirklichkeit zeichnet es direkt auf den Container.

Solange Sie bei den VB-Zeichenbefehlen bleiben, werden Sie wahrscheinlich keine Probleme haben. Aber mit den API-Funktionen müssen Sie vorsichtig sein. In diesem Beispiel funktioniert es, auf den durch die `hDC`-Eigenschaft bereitgestellten Gerätekontext zu zeichnen. Das Koordinatensystem für diesen Gerätekontext ist dasselbe, wie es für ein plaziertes Steuerelement wäre, aber es gibt keinen Grund, warum dies immer der Fall sein sollte. Dieser Code sollte den Viewport, das Ausmaß und den Ursprung des Gerätekontexts mit Hilfe der Funktionen `GetViewPortOrgEx` und `GetViewPortExtEx` ermitteln und in den bereitgestellten Bereich zeichnen.

Keinesfalls sollten Sie API-Funktionen verwenden, die mit Fenster-Handles arbeiten, um zu zeichnen oder Größenberechnungen auszuführen.

Beachten Sie, daß das Steuerelementfenster nicht auf dem Steuerelement plaziert wird, die Ergebnisse, die Sie erhalten, sind also sehr wahrscheinlich fehlerhaft oder unpraktikabel. Diese Probleme treten nur für Container auf, die die Steuerelemente nicht zur Entwurfszeit plazieren.

Wie können Sie erkennen, ob Ihr Steuerelement plaziert wurde? Sie haben zwei Möglichkeiten:

- Sie setzen während des `Show`-Ereignisses ein Flag. Dieses Ereignis tritt nur auf, wenn das Steuerelement plaziert wird.
- Sie verwenden den folgenden Code:

```
' If Parent.hwnd <> GetParent(hwnd) Then MsgBox "Ich wurde nicht plaziert!"
```

Diesen letzteren Ansatz können Sie während des Paint-Ereignisses und während des `ReadProperties`-Ereignisses sowohl unter VB4 als auch unter VB6 ausprobieren, um den Unterschied zwischen beidem zu erkennen.

Visual Basic scheint Probleme zu haben, Formulare auszugeben, die Steuerelemente enthalten, die während des `Paint`-Ereignisses eines Standardelements etwas zeichnen. Die Linien im Bildfeld werden also bei der `PrintForm`-Methode nicht ausgegeben. Ich habe derzeit keine Informationen darüber, wie man dieses Problem lösen könnte.

17.3.4 Die InvisibleAtRuntime-Eigenschaft

Diese `UserControl`-Eigenschaft können Sie auf `True` setzen, um zu verhindern, daß Ihr Steuerelement zur Laufzeit auf einem Container plaziert wird.

Sie setzt ein Flag für das Steuerelement, das dem Container anzeigt, daß er das Steuerelement nicht plazieren oder aktivieren soll. Sie verhindert Ereignisse der Benutzeroberfläche, wie beispielsweise `Show`, `Paint` und `Resize`, ebenso wie Eingabeereignisse (der Fokus kann verständlicherweise nicht auf ein unsichtbares Steuerelement gesetzt werden). Es stehen keine anderen VB-Extender-Eigenschaften als `Name`, `Index`, `Left`, `Top` und `Tag` zur Verfügung.

ActiveX-Steuerelemente, die unter Visual Basic 5 erzeugt wurden, haben immer ein Steuerelementfenster, auch wenn diese Eigenschaft `True` ist. Auch Standard-Steuerelemente werden erzeugt.

Es gibt nur einen Haken. Container müssen diese Option nicht unterstützen. Wenn Sie Ihr Steuerelement auf einem solchen Container einsetzen, ist es sichtbar. Sie erkennen diese Bedingung, indem Sie prüfen, ob das `Resize`- oder das `Paint`-Ereignis auftreten. Anschließend haben Sie mehrere Möglichkeiten. Sie könnten die `Extender`-Eigenschaft nutzen, um die `Visible`-Eigenschaft des Steuerelements auf `False` zu setzen, oder das Steuerelement an eine Stelle außerhalb des sichtbaren Bereichs des Containers schieben. Sie müssen die `Resize`- oder `Paint`-Ereignisse überwachen, um sicherzustellen, daß der Entwickler nicht versucht, Ihr Steuerelement anzuzeigen. Unterscheiden Sie außerdem immer zwischen Entwurfszeit und Laufzeit des Containers – zur Entwurfszeit soll Ihr Steuerelement sichtbar sein!

Für viele Situationen, in denen Sie eine `InvisibleAtRuntime`-Eigenschaft einsetzen, könnte eine ActiveX-Codekomponente eine bessere Wahl darstellen. Die Vorteile von `InvisibleAtRuntime`-Steuerelementen sind, daß sie Ihnen erlauben, eine Benutzeroberfläche für die Entwurfszeit (unter anderem Eigenschaftsseiten) bereitzustellen und Daten persistent zu machen.

17.3.5 Fensterlose Steuerelemente

Als die ActiveX-Steuerelemente immer beliebter wurden, suchte Microsoft nach Wegen, sie effizienter zu machen. Das war insbesondere wichtig, weil einige Programmierer darauf bestanden, Dutzende oder Hunderte von Steuerelementen auf einem Formular anzulegen. Der Overhead, für jedes Steuerelement ein Fenster zu unterstützen, hatte einen wesentlichen Einfluß auf die Ladezeiten und die Performance des Formulars. Die Entwickler bei Microsoft verfolgen einen interessanten

Ansatz, wenn sie feststellen, daß eines ihrer Objekte zu »heavy« ist. Weil sie mehr oder weniger an die existierende Architektur gebunden sind, erzeugen sie eine neue Architektur, die irgendwie ein bißchen weniger funktional, aber dafür effizienter ist. Auf diese Weise minimieren sie die Möglichkeit, die Abwärtskompatibilität mit existierenden Applikationen zu zerstören.

Damit geht auch eine völlig neue Methode einher, wie mit Hilfe dieser neuen Funktionen gearbeitet wird, und eine neue Möglichkeit für Container oder Betriebssysteme, miteinander inkompatibel zu sein, aber ich denke mal, das ist der Preis des Fortschritts. Bei den Steuerelementen hat Microsoft einen Mechanismus für Container erfunden, ActiveX-Steuerelemente zu unterstützen, die keine Fenster verwenden. Diese »fensterlosen« oder schlanken Steuerelemente bieten eine verbesserte Performance und Ressourcennutzung, auf Kosten einiger Funktionen und der Kompatibilität. Was opfern Sie, wenn Sie ein fensterloses Steuerelement erzeugen?

- Sie können keine Win32-API-Funktionen für die Arbeit mit dem Steuerelement verwenden. Das betrifft auch das Zeichnen, das Versenden von Nachrichten, Steueroperationen, Unterklassenbildung usw. Diese Steuerelemente haben keine Fenster-Handles.
- Schlanke Steuerelemente können keine Standardelemente aufnehmen, die nicht selbst fensterlos sind.
- Ihr Steuerelement kann nicht als Container-Steuerelement agieren. Die `ControlContainer`-Eigenschaft muß `False` sein.
- Einige Container unterstützen keine fensterlosen Steuerelemente. Visual Basic sollte ein Fenster für Ihr Steuerelement erzeugen, wenn es auf einem solchen Container eingesetzt wird.
- Die Eigenschaften `BorderStyle` und `EditAtDesignTime` sind für diesen Typ Steuerelement deaktiviert (weitere Informationen über diese Eigenschaften erhalten Sie später).

Warum sollten Sie fensterlose Steuerelemente anlegen? Erstens bieten sie einige interessante Möglichkeiten, Steuerelemente transparent zu machen (wie später noch erklärt wird). In Hinblick auf die Ressourcen sind sie effizienter. Ich persönlich vermeide die Verwendung von fensterlosen Steuerelementen für alles, was dem Benutzer Eingaben erlaubt (auch wenn diese Steuerelemente den Fokus erhalten können), und auch für sehr komplizierte Steuerelement. Sie sind ideal geeignet für einfache Steuerelemente, die Bilder oder Text anzeigen sollen. Beispielsweise sind das Bezeichnungsfeld und das Anzeigefeld von Visual Basic fensterlos. Das scheint ein bißchen konservativ zu klingen, aber ich habe einen Grund dafür.

Die Spezifikationen für fensterlose Steuerelemente sind relativ neu, und ihr Verhalten wird von jedem Container anders implementiert. Das bedeutet, der Contai-

ner ist dafür verantwortlich, Methoden für die Schnittstellen des Steuerelements aufzurufen, um alles mögliche zu erledigen, unter anderem das Zeichnen, die Positionierung, die Sichtbarkeit bis hin zur Fokusverwaltung. Wie groß ist die Wahrscheinlichkeit, daß unterschiedliche Container diese Funktionen wirklich korrekt und auf dieselbe Weise implementieren? Wieviele Tests hat jeder Container durchlaufen, um diese Technologie auszuprobieren?

Ein Windows-basiertes Steuerelement dagegen überläßt einen Großteil der Positionierung, Sichtbarkeits- und Fokusverwaltung dem Betriebssystem. Damit haben Sie nicht nur mit weniger Betriebssystemen zu tun, sondern das Betriebssystem unterliegt mit ziemlicher Sicherheit mehr Tests als jeder einzelne Client. Außerdem werden die Bugs in Betriebssystemen meist relativ schnell gefunden, weil fast jede Applikation damit arbeitet. Wieviele fensterlose Steuerelemente gibt es, um die Container zu testen, und wie sorgfältig sind diese Tests?

Würde ich ein privates Steuerelement für die Verwendung in einer Applikation schreiben, würde ich die fensterlose Form verwenden, um Ressourcen zu sparen und die Performance zu verbessern. Ich könnte es auch für ein einfaches Steuerelement verwenden, das den Fokus nicht erhält und keine Standardelemente enthält. Aber ich werde noch ein bißchen warten, bevor ich riskiere, ein fensterloses Steuerelement kommerziell anzubieten.

17.3.6 Fokus-Ereignisse und Eigenschaften

Es gibt einen riesigen Unterschied zwischen benutzerdefinierten Steuerelementen und Standardelement-basierten Steuerelementen, was die Fokusverwaltung betrifft. Wenn ein Steuerelement ein Standardelement enthält, erhält das `UserControl`-Objekt nie selbst den Fokus. Andernfalls ist es möglich, daß es den Fokus erhält. Diesen Sachverhalt sehen Sie in den Abbildungen 17.1 und 17.2.

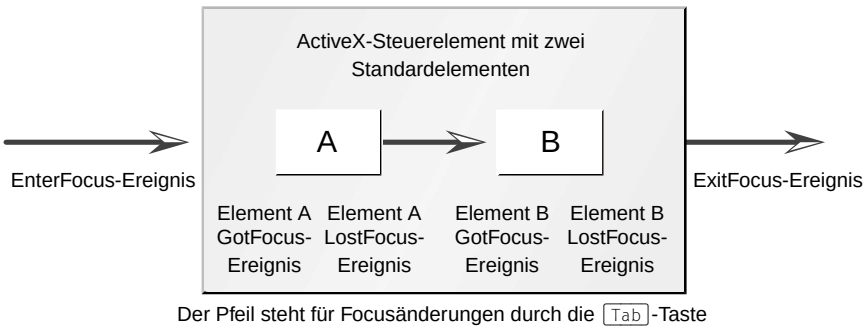

Abb. 17.1: Fokusreihenfolge für ein Steuerelement mit Standardelementen.

Die Ereignisse `EnterFocus` und `ExitFocus` zeigen an, wenn ein Standardelement in einem ActiveX-Steuerelement den Fokus hat.

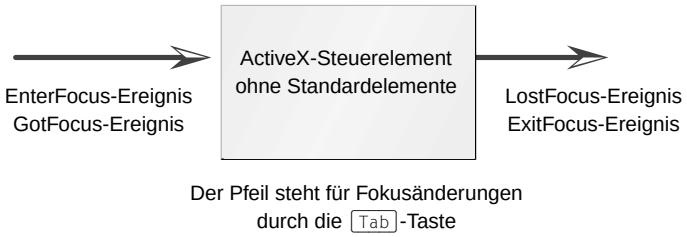

Abb. 17.2: Fokusreihenfolge für ein Steuerelement ohne Standardelemente

Die Verwendung von Fokus-Ereignissen ist im allgemeinen sehr kompliziert. Viele Visual-Basic-Programmierer verwenden sie zur Datenauswertung. Die Auswertung ist ein komplexes Thema, das später in diesem Kapitel noch kurz angesprochen wird. Sie können die `GotFocus`- und `LostFocus`-Ereignisse in Steuerelementen so verwenden wie in normalen Visual-Basic-Programmen.

17.3.7 Das GotFocus-Ereignis

Wenn ein Steuerelement das `GotFocus`-Ereignis erhält, hat es den Eingabefokus bereits. Das Ereignis tritt nach dem `EnterFocus`-Ereignis für ein Steuerelement auf.

Der Container kann sein `GotFocus`-Ereignis für ein Steuerelement auslösen, bevor Sie das `GotFocus`-Ereignis des `UserControl`-Objekts in Ihrem Steuerelement empfangen. Und genau das passiert in Visual Basic, aber es gibt keine Garantie, daß jeder Container sich gleich verhält.

Das bedeutet, daß ein Entwickler, der Ihr Steuerelement verwendet, während des `GotFocus`-Ereignisses des Containers Code ausführen könnte, weil er glaubt, Ihr Steuerelement hätte den Fokus. Ihr Steuerelement hat zwar den Fokus, aber es hat sein `GotFocus`-Ereignis noch nicht erhalten.

Die `HasTheFocus`-Methode im Steuerelement `ch17ctlD` demonstriert eine zuverlässige Möglichkeit, festzustellen, ob Ihr Steuerelement wirklich den Fokus besitzt. Schreiben Sie die folgende API-Deklaration in das Modul.

```
Private Declare Function GetFocus Lib "user32" () As Long
```

Die Methode `HasTheFocus` vergleicht den Fenster-Handle, der von der Funktion zurückgegeben wurde, mit dem Fenster-Handle des `UserControl`-Objekts. Sie könnten dieses Beispiel ganz einfach abändern, um auch den Fokus von Standardelementen abzutesten.

```
Public Function HasTheFocus() As Boolean
   If GetFocus() = hwnd Then
      HasTheFocus = True
   End If
End Function
```

Ein wichtiger Verwendungszweck für dieses Ereignis ist, einen Hinweis darauf bereitzustellen, daß das Steuerelement den Fokus erhalten hat (bei einem benutzerdefinierten Steuerelement). Es ist sehr wichtig, daß die Entwickler, die Ihr Steuerelement verwenden (und auch ihre Endbenutzer), erkennen können, wann Ihr Steuerelement den Fokus hat, und wann nicht.

Während dieses Ereignisses sollten Sie:

- Den Fokusstatus für benutzerdefinierte Steuerelemente anzeigen.

Während dieses Ereignisses dürfen Sie nicht:

- Versuchen, das `GotFocus`-Ereignis des `Extender`-Objekts aufzuwerfen. Dafür ist der Container verantwortlich.

Während dieses Ereignisses sollten Sie nicht:

- Dieses Ereignis verwenden, um ein Flag zu setzen, das zuverlässig anzeigt, ob das Steuerelement den Fokus hat. Das Steuerelement hat den Fokus möglicherweise schon, bevor dieses Ereignis auftritt.

17.3.8 Das LostFocus-Ereignis

Wenn ein Steuerelement das `LostFocus`-Ereignis erhält, hat es den Eingabefokus bereits verloren. Dieses Ereignis tritt vor dem `ExitFocus`-Ereignis für ein Steuerelement auf.

Der Steuerelement-Container sollte das `LostFocus`-Ereignis für ein Steuerelement auslösen, nachdem Sie die Ereignisse `LostFocus` und `ExitFocus` in Ihrem Steuerelement erhalten. Genau das passiert in Visual Basic, aber es gibt keine Garantie, daß sich jeder Container so verhält.

Während dieses Ereignisses können Sie den Fokus auf das Steuerelement zurücksetzen, das ihn gerade verloren hat, aber nicht, bevor das nächste Steuerelement ihn erhalten hat (und nach einem `GotFocus`- und `LostFocus`-Ereignis). Das Ereignis wird nämlich nicht ausgelöst, wenn das Steuerelement den Fokus verliert. Es wird später im Lauf der normalen Windows-Ereignisverarbeitung ausgelöst. Damit wird die Verwendung dieses Ereignisses auf Ausgaben aus der Datenauswertung begrenzt, aber dabei handelt es sich um ein ganz allgemeines Problem von Visual Basic, das nicht nur für ActiveX-Steuerelemente auftritt. Das `Validate`-Ereignis, das später beschrieben wird, bietet eine bessere Möglichkeit, die Auswertung vorzunehmen.

Während dieses Ereignisses sollten Sie:

- Für benutzerdefinierte Steuerelemente anzeigen, wenn sie den Fokus nicht haben.

17.3.9 Die Ereignisse EnterFocus und ExitFocus

Das EnterFocus-Ereignis tritt auf, wenn ein Steuerelement oder eines seiner Standardelemente den Fokus zum ersten Mal erhalten. Das Steuerelement hat den Fokus bereits, wenn dieses Ereignis auftritt. Das ExitFocus-Ereignis tritt auf, wenn ein Steuerelement oder alle seine Standardelemente den Fokus verlieren. Das Steuerelement hat den Fokus bereits verloren, wenn dieses Ereignis auftritt.

Dies sind die vielleicht wichtigsten Ereignisse für die Fokusverwaltung, wenn Steuerelemente angelegt werden, auf denen Standardelemente genutzt werden. Man kann sie sich auch als globales GotFocus oder LostFocus für das gesamte Steuerelement vorstellen.

Nachdem Sie die vier wichtigsten Fokusereignisse kennen, wollen wir genauer betrachten, in welcher Reihenfolge sie auftreten.

Die Steuerelemente ch17ctlD und ch17ctlE in der Projektgruppe ch17tst1 verwenden debug.print-Anweisungen, um die Abfolge der Ereignisse zu verfolgen. Das Steuerelement ch17ctlD verwendet keine Standardelemente. Das Steuerelement ch17ctlE enthält zwei Standard-Textfelder. Das Formular frmTest3 enthält beide Steuerelemente. Wenn Sie das Projekt ausführen, dieses Formular anzeigen und dann mit der Tab-Taste zwischen den Steuerelementen wechseln, entstehen sehr interessante Ergebnisse. Das interessanteste Ergebnis, das Sie hier sehen, bezieht sich auf das EnterFocus-Ereignis:

```
D:EnterFocus            ' Hier wird das Formular zuerst geladen
D: Has the focus? True
Form: D: GotFocus        ' Der Fokus befindet sich im Steuerelement
ch17ctlD
Form: D has focus? True
D:GotFocus
Form: D: Validate        ' Tab zum Steuerelement ch17ctlE
E:EnterFocus
D:LostFocus
D:ExitFocus
D: Has the focus? False
Form: D: LostFocus
Form: D has focus? False
Form: E: GotFocus
E:Text1:GotFocus
E:Text1:Validate            ' Tab zum 2. Textfeld auf dem Steuerelement
ch17ctlE
E:Text1:LostFocus
E:Text2:GotFocus
Form: E: Validate        ' Tab zum Steuerelement ch17ctlD
D:EnterFocus
D: Has the focus? True
E:Text2:LostFocus
E:ExitFocus
```

```
Form: E: LostFocus
Form: D: GotFocus
Form: D has focus? True
D:GotFocus
D:LostFocus            ' Formular schließen
D:ExitFocus
D: Has the focus? False
E:LostFocus
```

Ignorieren Sie die `Validate`-Ereignisse bis auf weiteres. Die Frage »D has focus?« zeigt, ob das Steuerelement D den Fokus zum Zeitpunkt des vorhergehenden Ereignisses wirklich hat. Wie Sie sehen, tritt das `EnterFocus`-Ereignis auf, sobald Sie mit der Tab-Taste in ein Steuerelement wechseln. Es tritt nicht nur auf, bevor das `GotFocus`-Ereignis für das Steuerelement oder seine Elemente aufgetreten ist, sondern bevor die `LostFocus`- oder `ExitFocus`-Ereignisse der Steuerelemente den Fokus verlieren!

17.3.10 Das Validate-Ereignis und die CausesValidation-Eigenschaft

Visual Basic 6 bietet ein neues Auswertungssystem in Form der `CausesValidation`-Eigenschaft und des `Validation`-Ereignisses. Als erstes bemerken Sie vielleicht, daß das `UserControl`-Objekt keine `CausesValidation`-Eigenschaft hat, und auch kein `Validate`-Ereignis empfängt. Die Eigenschaft und das Ereignis sind nämlich nicht Teil des Steuerelement-Extenders. Sie werden vom Container des Steuerelements bereitgestellt, und nicht vom Steuerelement selbst. Versuchen Sie, das Steuerelement `ch17ctl1.ocx` in ein Visual-Basic-5-Formular zu laden. Sie werden feststellen, daß die Steuerelemente weder ein `Validate`-Ereignis noch eine `CausesValidation`-Eigenschaft bereitstellen. Andererseits unterstützen die Standardelemente in Ihrem Steuerelement die Auswertung, weil der Container aus ihrer Perspektive Ihr VB6-Steuerelement ist, das dieses Ereignis und diese Methode unterstützt. Bedeutet das, Sie können eine Auswertung von Standardelementen in Ihr Steuerelement einbauen? Nicht unbedingt; aber wir werden gleich noch darauf zu sprechen kommen.

Erstens, was genau macht die Auswertung? Zunächst müssen Sie etwas verstehen, das nur vom Container unterstützt wird. Wenn Visual Basic einen Versuch erkennt, daß der Fokus von einem Steuerelement zum nächsten weitergegeben werden soll, prüft es zunächst, ob die `CausesValidation`-Eigenschaft `True` ist (das ist die Standardeinstellung). Ist dies der Fall, wirft es das `Validate`-Ereignis für das Steuerelement auf. Während dieses Ereignisses haben Sie die Gelegenheit, das Steuerelement zu betrachten und seinen Inhalt auszuwerten. Wenn Sie möchten, können Sie die Fokusweitergabe blockieren. Die Auswertung ist das erste, was passiert, wenn Sie versuchen, den Fokus von einem Steuerelement weiterzugeben. Aber es gibt einen Haken.

Angenommen, Sie legen ein Formular mit zwei Standardelementen an (wie im Steuerelement `ch17ctlE`), und ein Entwickler verwendet das `Validate`-Ereignis

Ihres Steuerelements. Der Entwickler erhält das `Validate`-Ereignis, wenn Sie den Fokus vom Steuerelement entfernen, sieht aber keine Auswertung, weil Sie zwischen den Standardelementen weiterschalten. Ihr erster Gedanke wäre vielleicht, die Auswertung der Standardelemente innerhalb Ihres Steuerelements vorzunehmen, aber dafür gibt es ein kleines Problem. Die Auswertung zwischen den Standardelementen funktioniert nämlich nur, wenn Sie zwischen den Standardelementen wechseln. Wenn Sie den Fokus nach außerhalb des UserControls weitergeben, wird das `Validate`-Ereignis Ihres Standardelements nicht ausgelöst.

Ein Ansatz wäre, zwei Auswertungsebenen bereitzustellen. Die erste Auswertung sollte im `Validate`-Ereignis des Steuerelements bereitgestellt werden, ohne daß Sie selbst irgend etwas dazu tun müßten. Wenn eine interne Auswertung erforderlich ist, sollten Sie in Betracht ziehen, ein Ereignis für den Client bereitzustellen, das das `Validate`-Ereignis der Standardelemente reflektiert, die Sie auswerten möchten, aber Sie sollten das Ganze so dokumentieren, daß die Benutzer wissen, daß sie während des `Validate`-Ereignisses des Steuerelements ebenfalls eine Auswertung vornehmen müssen. Sie können auch eine Methode bereitstellen, die `ValidateControls` aufruft. Diese Methode erzwingt eine Auswertung des letzten Standardelements. Wenn Sie eine Auswertung innerhalb Ihres Steuerelements vornehmen müssen, könnten Sie weiterhin die gefährliche, aber exakte Technik der Unterklassenbildung für die Methode `WM_KILLFOCUS` anwenden (beachten Sie jedoch, daß dieser Ansatz für fensterlose Steuerelemente nicht funktioniert).

17.3.11 Die CanGetFocus-Eigenschaft

Diese Eigenschaft gibt an, ob ein ActiveX-Steuerelement den Fokus erhalten kann. Wenn Sie sie auf `False` setzen, können Sie alles, was Sie hier über den Fokus gehört haben, sofort vergessen, weil es nie auftreten wird.

Diese Eigenschaft kann nur für benutzerdefinierte Steuerelemente oder Steuerelemente mit Standardelementen, die den Fokus nicht erhalten können, auf `False` gesetzt werden (beispielsweise das Timer-Steuerelement).

17.3.12 Die AccessKeys-Eigenschaft

Zugriffstasten (manchmal auch als Beschleunigertasten bezeichnet) erlauben Ihnen, über eine Alt-Tastenkombination direkt auf ein Steuerelement zu springen.

Aus der Sicht des Entwicklers gibt es zwei Methoden, eine Zugriffstaste festzulegen. Bei einigen Steuerelementen wird die betreffende Taste im Titel unterstrichen dargestellt, indem ihr ein Ampersand-Zeichen vorangestellt wird. Bei Steuerelementen ohne Titel ist es gebräuchlich, den Titel in einem Bezeichnungsfeld dafür zu verwenden, der dem betreffenden Steuerelement in der Tabulatorreihenfolge vorausgeht.

Um also Alt-S als Zugriffstaste für eine Schaltfläche festzulegen, könnten Sie den Titel auf `&Schaltfläche` setzen, so daß auf der Schaltfläche SCHALTFLÄCHE angezeigt wird.

Bei einem Textfeld könnten Sie unmittelbar davor in der Tabulatorreihenfolge ein Beschriftungsfeld anlegen und den Titel auf dieselbe Weise setzen. Beispielsweise könnten Sie ihn auf `&Textfeld` setzen, so daß sich TEXTFELD ergibt. Als Autor von Steuerelementen sollten Sie sich fragen, was ein Steuerelement intern tun muß, um diese Möglichkeit zu unterstützen.

In einem ersten Schritt muß also festgelegt werden, welche Tasten die Zugriffstaste für ein Steuerelement sein sollen. Es gibt zwei Möglichkeiten, eine Zugriffstaste für ein unter Visual Basic entwickeltes Steuerelement zu spezifizieren:

- Sie realisieren den Zugriff über den Titel eines Standardelements (indem Sie beispielsweise eine Zugriffstaste für eine Standardschaltfläche oder ein Bezeichnungsfeld setzen).
- Sie fügen das Zeichen der `AccessKeys`-Eigenschaft hinzu. Diese Eigenschaft enthält einen String aus Zeichen, die alle als Zugriffstasten für Ihr Steuerelement verwendet werden können.

Dabei können Groß- und Kleinbuchstaben verwendet werden.

Wie das Steuerelement auf die Zugriffstaste reagiert, ist von der `ForwardFocus`-Eigenschaft abhängig.

17.3.13 Die ForwardFocus-Eigenschaft

Wenn für ein Bezeichnerfeld eine Zugriffstaste angegeben wurde, wird der Fokus auf das nächste Steuerelement in der Tabulatorreihenfolge gesetzt, sobald diese Taste gedrückt wird. Ihr ActiveX-Steuerelement weist dasselbe Verhalten auf (Weitergabe des Fokus an das nächste Steuerelement), wenn Sie die `ForwardFocus`-Eigenschaft für das Steuerelement auf `True` setzen.

Ist diese Eigenschaft `False`, ist das Verhalten der Zugriffstasten davon abhängig, ob es sich um ein benutzerdefiniertes Steuerelement handelt, oder ob es Standardelemente enthält. Ist es benutzerdefiniert, geht der Fokus an das Steuerelement selbst. Andernfalls geht er entweder auf das erste Standardelement (für die durch die `AccessKeys`-Eigenschaft festgelegten Zugriffstasten), oder an das der Zugriffstaste zugeordnete Standardelement.

Ist diese Eigenschaft `True`, geht der Fokus immer an das nächste Steuerelement im Container. Das passiert, egal ob die Zugriffstaste in der `AccessKeys`-Eigenschaft oder durch ein Standardelement festgelegt ist. Die Interaktion zwischen den Eigenschaften `AccessKeys` und `ForwardFocus` ist in Tabelle 17.3 verdeutlicht.

In Access-Keys?	Zugriffstaste für Standard-element?	Fokus weitergeben?	Ergebnis
Nein	Ja	Nein	Der Fokus geht zum nächsten Standardelement oder zum Steuerelement selbst (falls es benutzerdefiniert ist). Das `AccessKeyPress`-Ereignis wird nicht ausgelöst.
Nein	Ja	Ja	Der Fokus geht zum nächsten Steuerelement im Container (nicht zum nächsten Standardelement).
Ja	Nein	Nein	Dies hat keine Auswirkung, wenn der Fokus bereits auf einem Standardelement liegt. Andernfalls geht der Fokus zum nächsten Standardelement oder zum Steuerelement selbst (falls dieses benutzerdefiniert ist). Das `AccessKeyPress`-Ereignis wird ausgelöst.
Ja	Nein	Ja	Der Fokus geht zum nächsten Steuerelement auf dem Container (nicht zum nächsten Standardelement).

Tab. 17.3: Überblick über das Verhalten von AccessKey und ForwardFocus

17.3.14 Das AccessKeyPressed-Ereignis

Dieses Ereignis wird ausgelöst, um anzuzeigen, daß eine Zugriffstaste gedrückt wurde. Es tritt nur auf, wenn `ForwardFocus False` ist.

Es wird ausgelöst, wenn der Benutzer eine Zugriffstaste verwendet, die in der durch die `AccessKeys`-Eigenschaft spezifizierten Liste enthalten ist (es wird nicht für Zugriffstasten ausgelöst, die durch Standardelemente angegeben werden).

Außerdem wird es ausgelöst, wenn die `DefaultCancel`-Eigenschaft `True` ist (mehr dazu später), und das Steuerelement wird vom Entwickler als Standard- oder Abbrechen-Schaltfläche für das Formular festgelegt. Weitere Informationen darüber finden Sie in der Beschreibung der `DefaultCancel`-Eigenschaft.

17.3.15 Die Arbeit mit Zugriffstasten

Wie Sie gesehen haben, gibt es zahlreiche mögliche Permutationen bei der Verwendung von Zugriffstasten, Standardelementen, `ForwardFocus` und `CanGetFocus`. Nun betrachten wir die sinnvollsten und zugleich realistischsten Szenarios:

Steuerelemente im Stil von Bezeichnungsfeldern: Setzen Sie die Zugriffstasten mit Hilfe der `AccessKeys`-Eigenschaft basierend auf den Eigenschaftseinstellungen des Entwicklers. Setzen Sie `CanGetFocus` auf `False` und `ForwardFocus` auf `True`.

Benutzerdefinierte Steuerelemente, die den Fokus erhalten: Setzen Sie die Zugriffstasten mit Hilfe der `AccessKeys`-Eigenschaft basierend auf den Eigenschaftseinstellungen des Entwicklers. Setzen Sie `CanGetFocus` auf `True` und `ForwardFocus` auf `False`. Zeigen Sie den Fokusstatus abhängig vom `GotFocus`-Ereignis an. Führen Sie alle Aktionen aus, die für das `AccessKeyPressed`-Ereignis erforderlich sind.

Steuerelemente, die sich aus Standardelementen zusammensetzen: Zugriffstasten können über die `AccessKeys`-Eigenschaft oder unter Verwendung der Standardelemente gesetzt werden. Die Tasten sollten vom Entwickler gewählt werden, der Ihr Steuerelement einsetzt. `CanGetFocus` ist natürlich `True` (weil das für diese Art Steuerelement erforderlich ist). Setzen Sie `ForwardFocus` auf `False`, um die Fokusweitergabe über die Tab-Taste innerhalb des Steuerelements selbst zu erlauben.

Sie sollten die Werte für die Zugriffstasten nie im Code festschreiben. Bieten Sie den Entwicklern immer eine Möglichkeit, bei der Verwendung Ihres Steuerelements ihre eigenen Zugriffstasten auszuwählen. Sie wollen schließlich niemandem vorschreiben, wie seine Benutzeroberfläche auszusehen hat, oder?

Achten Sie außerdem darauf, einen visuellen Hinweis bereitzustellen, der angibt, welche Zugriffstasten für Ihr Steuerelement festgelegt sind.

17.3.16 Die DefaultCancel-Eigenschaft

Die `DefaultCancel`-Eigenschaft wird genutzt, um dem Container anzuzeigen, daß Ihr Steuerelement als Standard- oder Abbrechen-Schaltfläche verwendet werden kann. Falls der Container diese Möglichkeit unterstützt, fügt er dem `Extender` die Eigenschaften `Default` und `Cancel` hinzu, um dem Entwickler so zu ermöglichen, Ihr Steuerelement zum Standard- oder Abbrechen-Steuerelement für den Container zu machen. Ist das Steuerelement die Standard- oder Abbrechen-Schaltfläche für einen Container, wird das `AccessKeyPress`-Ereignis ausgelöst, wenn der Benutzer die Eingabetaste (mit dem Tastencode 13) oder die Escape-Taste (mit dem Tastencode 27) drückt, abhängig von der gewählten Einstellung.

Komischerweise wird die Eingabetaste immer erkannt, nachdem die `DefaultCancel`-Eigenschaft einmal gesetzt wurde, wenn sie gedrückt wird, während das Steuerelement den Fokus hat. Damit wird das Verhalten der Standardschaltfläche nachgebildet, die durch die Eingabetaste angeklickt wird, auch wenn ihre `Default`-Eigenschaft `False` ist.

17.4 Transparente Steuerelemente

Als erstes wollen wir die Transparenz für normale fensterbasierte Steuerelemente betrachten, weil fensterlose Steuerelement etwas andere Transparenzeigenschaften aufweisen. Visual Basic bietet nahezu optimale Unterstützung der Transparenz für Ihr gesamtes Steuerelement oder einen Teil davon. Die Transparenz wird aktiviert, indem die `BackStyle`-Eigenschaft des Steuerelements auf Null gesetzt wird. Warum sage ich »nahezu"? Weil es, wie Sie gleich sehen werden, einige kleinere Probleme gibt, die damit zu tun haben, wie die Transparenz unter Windows 95 implementiert wurde. Diese Probleme wurden scheinbar mit Windows 98 aus der Welt geschaffen, aber Sie sollten alle Steuerelemente, die die Transparenz nutzen, unter allen Betriebssystemen, auf denen sie eingesetzt werden könnten, sorgfältig testen.

Wenn die `BackStyle`-Eigenschaft auf 1 gesetzt ist (undurchsichtig), erscheint das Steuerelement als rechteckiger Bereich, und nichts von dem Container scheint durch. Ist `BackStyle` auf Null gesetzt (transparent), werden Teile des Steuerelements durchsichtig. Diese transparenten Bereiche weisen das folgende Verhalten auf:

- Der Container, der das Steuerelement enthält, legt das Erscheinungsbild der transparenten Bereiche fest. Mit anderen Worten: Der Container wird durch das Steuerelement hindurch angezeigt.
- Mausereignisse auf dem transparenten Bereich gehen an den Container, nicht an das Steuerelement. Das gilt unter anderem für `MouseUp`-, `MouseDown`- und `MouseMove`-Ereignisse, ebenso wie für erzeugte `Click`- und `DblClick`-Ereignisse.

Transparente Bereiche in einem Steuerelement werden gemäß der folgenden Regeln spezifiziert:

- Alle nichttransparenten Standardelemente sind undurchsichtig.
- Wenn Sie der `MaskPicture`-Eigenschaft eines Steuerelements eine Bitmap zuordnen, sind alle Bereiche in dieser Bitmap, die nicht die in der `MaskColor`-Eigenschaft angegebene Farbe haben, undurchsichtig. Die `MaskColor`-Eigenschaft sollte immer auf Weiß oder Schwarz gesetzt werden, wenn Sie möchten, daß Ihr Steuerelement unter Windows 95 korrekt funktioniert. Windows 95 verarbeitet die Transparenz intern unter Verwendung einer schwarzweißen Maskierung. Das bedeutet, während des Maskierungsprozesses wird der `MaskColor`-Wert in Schwarz oder Weiß konvertiert. Die einzige Möglichkeit, ihn zuverlässig zu setzen, ist also die Verwendung von Schwarz oder Weiß in der `MaskColor`-Eigenschaft und einer schwarzweißen Bitmap als `MaskPicture`-Eigenschaft.
- Transparente Bereiche in einem transparenten Steuerelement (beispielsweise ein Bezeichnungsfeld oder ein Figur-Steuerelement, deren `BackStyle`-Eigen-

schaft auf Null gesetzt ist) sind durchsichtig. Undurchsichtige Bereiche dieser Steuerelemente sind undurchsichtig, außer bei Bezeichnungsfeldern, die eine Nicht-TrueType-Schriftart verwenden.

Diese Regeln führen zu zahlreichen sehr nützlichen Einsatzbereichen für die Transparenz von Steuerelementen.

17.4.1 Transparente Steuerelemente mit Standardelementen

Diese Art von Steuerelementen setzt sich aus Standardelementen zusammen, wobei der Hintergrund des Containers zwischen den Steuerelementen erscheinen soll. Dazu setzen Sie die `BackStyle`-Eigenschaft von `UserControl` einfach auf 0 – Transparent.

Diese Art Steuerelement ist praktisch, wenn Sie ein Steuerelement verwenden möchten, um eine begrenzte Anzahl von Elementen zusammen mit einer definierten Funktionalität zu gruppieren, und Sie erwarten, daß diese Steuerelemente auf einem Container erscheinen, der ein komplexes Muster oder Bild (beispielsweise eine Landkarte) aufweist.

Sie sollten diesen Ansatz nicht als einfache Methode nutzen, immer die Hintergrundfarbe des Containers beizubehalten. Die Transparenz erhöht nämlich den Overhead Ihres Steuerelements ganz wesentlich, sowohl in Hinblick auf zusätzlich erforderliche Ressourcen, als auch auf eine verringerte Performance.

Wenn Sie einfach die Hintergrundfarbe des Containers verwenden möchten, sollten Sie Änderungen der `BackColor`-Eigenschaft des `Ambient`-Objekts erkennen und die Hintergrundfarbe Ihres Steuerelements entsprechend anpassen. Wie das geht, erfahren Sie in Kapitel 19.

17.4.2 Unregelmäßig geformte Steuerelemente

Wenn Ihr Steuerelement einen unregelmäßigen Umriß haben soll, haben Sie drei Möglichkeiten:

- Sie plazieren ein undurchsichtiges Figur-Steuerelement auf Ihrem Steuerelement, um den Umriß des Steuerelements zu definieren. Anschließend können Sie unter Verwendung von Standard-Zeichentechniken auf den Bereich zeichnen, den das Figur-Steuerelement einnimmt. Standardelemente erscheinen weiterhin so, wie Sie das erwarten. Der Haken bei diesem Ansatz ist, daß das Figur-Steuerelement nach dem `Paint`-Ereignis von `UserControl` gezeichnet wird, so daß es viel schwieriger wird, benutzerdefinierte Steuerelemente zu implementieren. Sie können eine Unterklasse von dem Steuerelement anlegen und das Zeichnen ausführen, nachdem das `Paint`-Ereignis wie üblich verarbeitet wurde.

- Dieser Ansatz ist im Beispielprojekt `OvalTest.vbp` auf der CD-ROM zum Buch demonstriert. Dieses Beispiel verwendet das Demo-Steuerelement für das Subclassing, `dwsbc32d.ocx` von Desaware SpyWorks. Ich würde Desawares In-Prozeß-Codekomponenten-Subclasser `dwSpyvb.dll` empfehlen, der eine effizientere Lösung in einer echten Applikation darstellt. Mehr über diese Komponente erfahren Sie in Kapitel 22.
- Sie definieren eine Bitmap, die in der `MaskPicture`-Eigenschaft für das Steuerelement gesetzt wird. Für diesen Zweck werden nur Bitmaps (`.BMP`, `.DIB`, `.GIF` und `.JPG`) unterstützt. Es ist nicht möglich, Metadateien oder Icons zu verwenden. Die `MaskColor`-Eigenschaft bestimmt, welche Farbe transparent ist. Dieser Ansatz ist relativ effizient, leidet aber unter der Unfähigkeit, eine gute Maske beizubehalten, wenn die Größe des Steuerelements geändert wird. Beachten Sie, daß die Einstellung für `MaskColor` Weiß sein muß, wenn die Transparenzfunktionen Ihres Steuerelements korrekt unter Windows 95 arbeiten sollen. Die Microsoft-Dokumentation empfiehlt, nur schwarzweiße Bitmaps für die `MaskPicture`-Eigenschaft unter Windows 95 zu verwenden.
- Helle Farben werden nämlich ebenfalls als Weiß abgebildet und als transparent betrachtet, so daß es schwierig ist, im voraus zu erkennen, welche Bereiche bei der Verwendung einer Farb-Bitmap transparent sein werden.
- Sie nutzen die Funktion `SetWindowRgn`, um einen unregelmäßig geformten Bereich für das Steuerelementfenster zu schaffen. Das ist im Projekt `OvalTst2.vbp` demonstriert, das den folgenden Code im Steuermodul enthält:

```
Private Declare Function CreateEllipticRgn Lib "gdi32" _
     (ByVal X1 As Long, ByVal Y1 As Long, ByVal X2 As _
     Long, ByVal Y2 As Long) As Long
     Private Declare Function SetWindowRgn Lib "user32" _
     (ByVal hWnd As Long, ByVal hRgn As Long, ByVal bRedraw _
     As Long) As Long

     Private Sub UserControl_Resize()
         Dim hr&, dl&
         Dim usew&, useh&
         usew& = ScaleWidth / Screen.TwipsPerPixelX
         useh& = ScaleHeight / Screen.TwipsPerPixelY
         hr& = CreateEllipticRgn(0, 0, usew, useh)
         dl& = SetWindowRgn(hWnd, hr, True)
     End Sub
```

Dieser Ansatz ist möglicherweise der effizienteste der drei, aber man muß dazu wirklich verstehen, wie mit Hilfe des Win32-API Region-Objekte erzeugt werden. Dieses Thema ist in Kapitel 7 des *Visual Basic 5.0 Programmer's Guide to the Win32 API* detailliert beschrieben.

17.4.3 Steuerelemente im Stil von Bezeichnungsfeldern

Die bei weitem einfachste Methode, ein transparentes Steuerelement im Stil von Bezeichnungsfeldern zu erzeugen, ist die Verwendung von einem oder mehreren transparenten Standard-Bezeichnungsfeldern. Dieser Ansatz hat jedoch zwei große Nachteile:

- Er funktioniert nur für TrueType-Schriften. Transparente Bezeichnungsfelder bilden keine korrekten Masken für Bitmap-Schriften.
- Sie sind auf die Fähigkeiten des Bezeichnungsfeldes beschränkt. Sie können also nicht Dinge wie Schriften, Stile und Farben in einem einzigen Steuerelement kombinieren.

Die Lösung für diese Situation ist etwas kompliziert, aber unendlich flexibel. Sie basiert auf der Idee, daß die undurchsichtigen Bereiche des Steuerelements durch die `MaskPicture`-Eigenschaft definiert werden können, und daß diese Eigenschaft zur Laufzeit definiert werden kann.

Das Projekt `lblTest.vbp` demonstriert einen einfachen Ansatz. Das Steuerelement enthält ein unsichtbares Bildfeld, dessen `AutoRedraw`-Eigenschaft auf `True`, und dessen `Visible`-Eigenschaft auf Null gesetzt sind. Dadurch bleibt das Bildfeld unsichtbar, erzwingt aber gleichzeitig, daß eine Bitmap reserviert wird, die den Inhalt des Steuerelements jederzeit speichert. Mit Hilfe von API-Funktionen wird Text in dieses Bildfeld geschrieben. In diesem Beispiel wird die Textausgabe in einem `Timer`-Ereignis plaziert, nämlich wie folgt:

```
Private Declare Function TextOut Lib "gdi32" Alias "TextOutA" _
     (ByVal hdc As Long, ByVal x As Long, ByVal y As Long, ByVal _
     lpString As String, ByVal nCount As Long) As Long

     Private Sub Timer1_Timer()
        Static counter&
        Dim t$
        counter = counter + 1
        t$ = "Elapsed " & counter & " seconds"
        Picture1.Cls
        Call TextOut(Picture1.hdc, 0, 0, t$, Len(t$))
        MaskPicture = Picture1.Image
     End Sub
```

Die Standard-Hintergrundfarbe für das Picture-Steuerelement und die `MaskColor`-Eigenschaft für `UserControl` werden in diesem Beispiel beide auf Weiß gesetzt, so daß es unter Windows 95 korrekt arbeitet. Der Befehl `Picture1.Cls` setzt das gesamte Bildfeld auf die aktuelle Hintergrundfarbe.

Der Befehl `TextOut` zeichnet mit der Standard-Textfarbe Text auf das Steuerelement `Picture1` – in diesem Fall ist das schwarz. Das win32-API beinhaltet zahlreiche extrem flexible Funktionen zur Textausgabe, die die unterschiedlichsten

Textausrichtungen sowie Zeilenumbrüche unterstützen. Auch sie sind in *The Visual Basic Programmer's Guide to the Win32 API* detailliert beschrieben.

Nachdem der Text gezeichnet wurde, wird die `MaskPicture`-Eigenschaft auf das aktuelle Bild des `Picture1`-Steuerelements gesetzt. In diesem Beispiel-Steuerelement wird der Text blau angezeigt, weil das die Hintergrundfarbe des Steuerelements ist. Was dabei letztlich passiert, ist, daß nur die Pixel im Steuerelement, die Text enthalten, undurchsichtig sind, und damit die Hintergrundfarbe des Steuerelements zeigen, statt die des Containers. Blau ist eine dunkle Farbe, und es ist unwahrscheinlich, daß es unter Windows 95 in Hinblick auf die Transparenz in Weiß abgebildet wird.

Das Steuerelement verwendet den folgenden Code im `Click`-Ereignis:

```
Private Sub UserControl_Click()
   Line (0, 0)-(ScaleWidth, ScaleHeight), 0, BF
End Sub
```

Wenn Sie auf den Text klicken (dazu sind möglicherweise mehrere Versuche notwendig, weil der Text relativ klein ist), wird das gesamte Steuerelement schwarz ausgefüllt. Weil die Maske jedoch nur die Textbereiche als undurchsichtig definiert, werden nur diese Bereiche schwarz dargestellt. Damit hat es den Anschein, als hätte der Text einfach die Farbe gewechselt.

Dieser Ansatz kann erweitert werden, um beliebige Masken und Zeichnungen zur Laufzeit zu erzeugen. Wenn Sie so vorgehen, können Sie unter Zuhilfenahme mehrerer Techniken den Overhead reduzieren, der bei der Verwendung eines separaten Bildfelds mit einer persistenten Bitmap entsteht. Sie können die `AutoRedraw`-Eigenschaft auf `False` setzen, wenn sie nicht gebraucht wird, um die zusätzliche Bitmap zu eliminieren. Außerdem können Sie weitere komplexe API-Techniken einsetzen, um nach Bedarf dynamisch eine Bitmap zu erzeugen und sie direkt in die `MaskPicture`-Eigenschaft zu laden (dazu wird die OLE-API-Funktion `OleCreatePictureIndirect` verwendet).

17.4.4 Bitmap-basierte Steuerelemente

Sie fragen sich vielleicht, warum Sie überhaupt eine Farb-Bitmap in der `MaskPicture`-Eigenschaft eines Steuerelements zur Bereitstellung der Maske verwenden sollten. Schließlich tut es eine schwarzweiße Bitmap auch; Sie brauchen nur zwei Pixel, um die Maske und die undurchsichtigen Bereiche zu spezifizieren.

Die Eleganz dieses Ansatzes liegt darin, daß Sie dieselbe Bitmap in den `MaskPicture`- und `Picture`-Eigenschaften eines Steuerelements verwenden können. Die `MaskColor`-Eigenschaft erlaubt Ihnen, eine einzelne Farbe im Bild als transparent zu spezifizieren. Sie können diesen Ansatz natürlich nur zuverlässig einsetzen, wenn Sie wissen, daß Ihr Steuerelement nur unter Windows NT eingesetzt wird, weil nur Windows NT in der Lage ist, beliebige Werte für die `MaskColor`-Eigenschaft und Farb-Bitmaps zu verarbeiten.

Das Steuerelement `ch17ctlG.ctl` im Projekt `ch17test1` demonstriert, wie Sie die hier beschriebenen Techniken einsetzen, um eine Vielzahl von Transparenzeffekten zu erzielen. Beachten Sie, daß transparente Steuerelemente nicht als Steuerelement-Container konfiguriert werden sollten.

Das Beispielprojekt `TranTest.vbp` finden Sie auf der CD-ROM zum Buch im Verzeichnis zu Kapitel 17. Dieses Programm erlaubt Ihnen, mit der Transparenz zu experimentieren, indem Sie unterschiedliche Bitmap-Typen und verschiedene Werte für die `MaskColor`-Eigenschaft einsetzen. Ich empfehle Ihnen, damit sowohl unter Windows 95 als auch unter Windows NT zu experimentieren.

17.4.5 Transparenz bei fensterlosen Steuerelementen

Ich muß zugeben, daß mich Microsoft mit seiner Erklärung der Transparenz und Treffertests in Hinblick auf fensterlose Steuerelemente zunächst völlig verwirrt hat. Was sollte das nur alles bedeuten, mit den »angrenzenden Bereichen« und »transparenten Bereichen«, von denen Visual Basic überhaupt nichts weiß? Wenn Visual Basic nichts davon weiß, warum schreibt man dann darüber. Nachdem ich die ganze Dokumentation sorgfältig gelesen hatte, war ich noch mehr verwirrt. Ich habe also Code geschrieben und begann, mit dem Beispielprogramm HitTest zu experimentieren. Anschließend versuchte ich das anzuwenden, was mir die Microsoft-Dokumentation gesagt hatte, und erreichte irgendwann den Punkt der völligen Frustration. Schließlich verwarf ich alles, was mir die Microsoft-Dokumentation über dieses Thema gesagt hatte. Ich bin nicht ganz sicher, was mir Microsoft damit mitteilen wollte, aber ich versuche hier mein Bestes, zu schildern, was ich herausgefunden habe.

Erstens, die Steuerelemente `MaskColor` und `MaskPicture` funktionieren genau wie Steuerelemente mit eigenen Fenstern. Das können Sie nachvollziehen, indem Sie das Beispielprogramm `Trantst2.vbp` ausführen, das gleich dem `trantest`-Projekt ist, außer daß das Steuerelement fensterlos ist. Der einzige feine Unterschied hat mit der `ClipBehavior`-Eigenschaft zu tun. Wenn Sie sie auf 1 setzen (das ist der Standardwert), erlaubt Visual Basic nur, in Bereiche des Steuerelements zu zeichnen, die unter Verwendung der Eigenschaften `MaskPicture` und `MaskColor` als undurchsichtig definiert wurden.

Der Unterschied zwischen den beiden Steuerelementtypen kommt erst bei Maus-Ereignissen ins Spiel.

Wenn Sie ein Steuerelement mit Fenster haben, ist alles relativ einfach. Sie klikken ein transparentes Steuerelement an. Wenn Sie auf einen undurchsichtigen Bereich klicken (aufgrund der Maskenfarbe von Bildfeld oder Standardelement), geht die Mausnachricht an das Steuerelement. Andernfalls stellt Windows fest, welches Fenster sich unterhalb des Steuerelements befindet (das nächste in der Z-Reihenfolge), und versucht es erneut, usw. usf.

Ich habe bereits erwähnt, daß fensterlose Steuerelemente völlig von dem Container verarbeitet werden. Das bedeutet, Visual Basic muß herausfinden, welches fensterlose Steuerelement die Mausklicks erhält. Das klingt ganz einfach und

könnte es auch sein, hätten nicht die Entwickler von Visual Basic beschlossen, noch eine zusätzliche Funktionalität bereitzustellen, falls Sie komplexe Schichten transparenter Steuerelemente anlegen möchten. Und jetzt wird alles kompliziert.

Als erstes müssen Sie wissen, daß es einen sogenannten Maskenbereich gibt, der nur Bereiche beinhaltet, die durch die Eigenschaften `MaskPicture` und `MaskColor` als undurchsichtig definiert wurden. Das bedeutet, alles was Sie unter Verwendung grafischer Steuerelemente außerhalb dieses Bereichs gezeichnet haben, kann undurchsichtig erscheinen, ist aber nicht Teil des Maskenbereichs für Mausereignisse. Alle Standardelemente außerhalb dieses Bereichs sind ebenfalls nicht Teil des Maskenbereichs für Mausereignisse.

Vor irgendeinem Mausereignis, das irgendwo über einem grafischen Steuerelement auftritt, wirft Visual Basic das `HitTest`-Ereignis des Steuerelements auf. Der `HitResult`-Parameter für dieses Ereignis kann nur eine 0 oder eine 3 sein. Der Wert ist von der Einstellung der `ClipBehavior`-Eigenschaft abhängig, wie in Tabelle 17.4 gezeigt.

ClipBehavior	Maus über Maskenbereich	Maus nicht über Maskenbereich
0 – None	0	0
1 – UseRegion	3	0
2 – Painted	3	3, falls ClipSiblings gleich 0 ist und der Bereich gezeichnet wird.

Tab. 17.4: Parameter für die ClipBehavior-Eigenschaft

Dies sind die Werte des `HitResult`-Parameters, wie er Ihrem Steuerelement übergeben wird. Sie können sie jedoch auch nutzen, um Ergebnisse an Visual Basic zurückzugeben.

Und das funktioniert so. Visual Basic startet oben in der Z-Reihenfolge (Steuerelement auf der höchsten Ebene) und wirft für jedes Steuerelement das `HitTest`-Ereignis auf. Ihr Steuerelement kann die `ClipBehavior`- und `ClipSiblings`-Eigenschaften nutzen, um Visual Basic aufzufordern, die beste Schätzung abzugeben, wo auf einem undurchsichtigen Bereich auf dem Steuerelement die Maus wirklich geklickt wurde (gemäß Tabelle 17.4). Wenn Ihnen diese Schätzung gefällt, ignorieren Sie das Ereignis einfach.

Sobald Visual Basic das Ergebnis 3 erhält, unterbricht es die Operation und sendet das Mausereignis an dieses Steuerelement.

Was ist, wenn kein Steuerelement ein `HitResult` von 3 zurückgibt?

Dann beginnt Visual Basic wieder oben in der Z-Reihenfolge und sucht jetzt nach einer 2. Sobald es das `HitResult` 2 erkennt, sendet es die Nachricht an das betreffende Steuerelement.

Das wird schließlich für einen `HitResult`-Wert von 1 wiederholt. Wenn alle Steuerelemente für alle drei Durchgänge den Wert 0 zurückgeben, sendet Visual Basic das Mausereignis an den Container.

All das Zeug in der Dokumentation über angrenzende Bereiche und transparente Bereiche ist nichts anderes als eine Empfehlung, wie man mit dieser Abfolge zurechtkommen könnte. Sie könnten beliebig 3 definieren, das einen direkten Treffer bedeutet, 2, das bedeutet, daß Sie relativ nahe liegen, und 1, das besagt, daß Sie sich irgendwo in dem transparenten Steuerelement befinden, und Sie Steuerelementen unterhalb dem Ihren, die undurchsichtig sind, Priorität geben wollen, aber das Mausereignis vom Container verarbeiten lassen, wenn keines der anderen Steuerelemente es will.

Ein anderer Ansatz wäre, 3 als direkten Treffer zu definieren, 2 als relativ nah, und 1 als irgendwo im Steuerelement-Rechteck.

Wichtig dabei ist: Die Definition, wann 3, 2 oder 1 zurückgegeben wird, ist völlig von Ihnen und Ihrem Steuerelement abhängig. Was Microsoft über »angrenzende« und »transparente« Abschnitte des Steuerelements sagt, ist völlig irrelevant. Konzentrieren Sie sich auf das, was VB macht, und nicht die Interpretation, die in der Dokumentation dargelegt ist.

Letzte Kommentare zur Transparenz. Das Verhalten, das ich beschreibe, scheint nicht mit der Dokumentation zusammenzupassen. Wie ich in der Dokumentation gelesen habe, sollten alle Standardelemente (wie beispielsweise das Figur-Steuerelement im Beispielprojekt `HitTest`) in den Maskenbereich für die Erkennung von Mausereignissen aufgenommen werden. Das scheint aber nicht der Fall zu sein. Ist das nun ein Bug oder eine Funktion? Ich weiß es nicht.

Achten Sie auf die Kommentare, die Microsoft zur Performance bei Treffertests macht. Microsoft ist nicht gerade erpicht darauf, irgendwo im Umfeld von Visual Basic (oder einer anderen Sprache) von einer schlechten Performance zu sprechen, Wenn sie also so vehement und häufig wie hier erwähnt wird, können Sie davon ausgehen, daß Sie einem größeren Performance-Problem gegenüberstehen, wenn Sie irgend etwas falsch machen.

Aus Kapitel 16 wissen Sie, daß Ihr Steuerelement immer ausgeführt wird, wenn es auf einem Container plaziert wird, auch wenn sich dieser Container im Entwurfsmodus befindet. Das bedeutet, alle hier in Hinblick auf die Transparenz beschriebenen Aspekte, auch für den Treffertest der Maus, gelten. Was passiert, wenn der Maskenbereich für Mausereignisse für ein Steuerelement nicht existiert, weil Ihr Steuerelement nur Standardelemente verwendet? Das `HitResult` ist immer Null. Das bedeutet, der Container, in diesem Fall die Visual-Basic-Entwurfsumgebung, erkennt den Treffer auf dem Steuerelement nie, und Sie können es nie auswählen oder verschieben! Sehr unpraktisch. Das Beispielprogramm `HitTest` fügt dem `HitTest`-Ereignis den folgenden Code für das Steuerelement hinzu, um sicherzustellen, daß das Steuerelement auch in der Entwurfszeitumgebung funktioniert:

```
If Ambient.UserMode Then
      ' Hier das Laufzeitverhalten definieren
   Else
      HitResult = 3 ' Treffer im Entwurfsmodus immer erkennen
   End If
```

Ich möchte Ihnen nahelegen, mit dem `HitTest`-Programm ein bißchen zu spielen und Änderungen daran vorzunehmen. Es verwendet die Bitmap `2Tran2.bmp` als `MaskPicture`, um einen Maskenbereich für die Experimente mit Mausereignissen zu definieren. Lassen Sie mich bitte wissen, wenn Sie Fehler erkennen oder meine Analyse ergänzen können. Ich glaube, daß sie genau das beschreibt, was Visual Basic tut, zumindest für den ersten Release von VB6. Ich wäre nicht überrascht, würde sich ein Teil dieses Verhaltens später als Bug erweisen, der in einem zukünftigen Service-Pack korrigiert wird. Wenn dem so ist, werde ich Informationen darüber auf unserer Web-Site unter `www.desaware.com` bereitstellen.

17.5 Weitere Eigenschaften und Methoden

Alle weiteren `UserControl`-Eigenschaften und -Ereignisse sind entweder identisch in ihrem Verhalten zu dem, was Sie von Formularen gewöhnt sind, oder sie sind in der Microsoft-Dokumentation so genau beschrieben, daß ich nichts weiter dazu zu sagen habe. In diesem Abschnitt werde ich Eigenschaften, Methoden und Ereignisse ansprechen, die noch einige zusätzliche Kommentare verdienen, und die in den folgenden Kapiteln nicht mehr weiter erklärt werden.

Die folgenden Eigenschaften werden an anderer Stelle im Buch besprochen:

Eigenschaft	Kapitel
Ambient	18
AsyncRead	19
CancelAsyncRead	19
CanPropertyChange	19
DataBindingBehavior	19
DataSourceBehavior	19
Extender	18
HyperLink	21
PropertyPages	20

17.5.1 Die Alignable-Eigenschaft

Wenn diese Eigenschaft auf `True` gesetzt wird, kann ein Container eine `Align`-Eigenschaft zum `Extender`-Objekt für das Steuerelement hinzufügen. Der Container ordnet das Steuerelement im Container dann basierend auf der durch den

Entwickler festgelegten Ausrichtung an, ohne daß Sie etwas dazu tun müßten. Wenn Sie möchten, markieren Sie die `Align`-Eigenschaft im `Extender`-Objekt, um eigene Ausrichtungen hinzuzufügen.

17.5.2 Die Eigenschaften ControlContainer und ContainedControls

Wenn Sie die `ControlContainer`-Eigenschaft auf `True` setzen, kann ein Entwickler, der Ihr Steuerelement einsetzt, zur Entwurfszeit zusätzliche Elemente darauf anlegen. Visual Basic übernimmt einen Großteil der Arbeit, um diese Möglichkeit zu unterstützen.

Mit Hilfe der `ContainedControls`-Auflistung können Sie auf Elemente zugreifen, die der Entwickler auf Ihrem Steuerelement plaziert hat. Diese Auflistung steht zur Verfügung, sobald das Steuerelement angelegt ist. Es ist nicht möglich, dieser Auflistung Elemente hinzuzufügen oder solche zu entfernen.

Einige Container unterstützen die Schnittstellen nicht, die man braucht, um Container-Steuerelemente einzusetzen. Diese Container werfen einen Fehler auf, wenn Sie versuchen, auf die `ContainedControls`-Eigenschaft zuzugreifen.

Vermeiden Sie, Steuerelement-Container aus Steuerelementen zu machen, die sichtbare Standardelemente enthalten. Das funktioniert, kann aber zu seltsamen Ergebnisse führen, abhängig davon, wo der Entwickler zusätzliche Steuerelemente plaziert.

Container-Steuerelemente verursachen ein bißchen Overhead. Sie sind vielleicht versucht, Container-Steuerelemente zu erzeugen, die organisatorische Aufgaben erfüllen, beispielsweise die Neuanordnung oder Größenänderung von Steuerelementen, die darauf plaziert sind. Es ist viel effizienter, diese Aufgaben mit Hilfe von Code auf einem Formular auszuführen, als spezialisierte Container-Steuerelemente zu verwenden.

Elemente, die durch ein `UserControl` dynamisch auf ein Steuerelement geladen werden (indem das Element der `Controls`-Auflistung hinzugefügt wird), werden zu neuen Standardelementen, nicht zu enthaltenen Elementen. Betrachten Sie die Beispielanwendung `DynTest.vbp`. Dort finden Sie einen Beweis für diese Aussage, ebenso wie den Beispielcode. Das dynamische Anlegen von Steuerelementen wurde bereits in Kapitel 11 beschrieben.

17.5.3 Die EditAtDesignTime-Eigenschaft

Wenn Sie diese Eigenschaft auf `True` setzen, wird es für einen Entwickler möglich, Ihr Steuerelement zur Entwurfszeit des Containers zu aktivieren und zu nutzen. Sie aktiviert den BEARBEITEN-Befehl im Kontextmenü des Steuerelements (das ist das Menü, das erscheint, wenn Sie mit der rechten Maustaste auf das Steuerelement klicken).

Wenn ein Entwickler Ihr Steuerelement zur Entwurfszeit auf einem Formular oder Container plaziert, wird normalerweise das gesamte Steuerelement als eine

Einheit betrachtet. Wenn Sie auf das Steuerelement klicken, wird das gesamte Steuerelement ausgewählt und es erscheinen die Felder für die Größenänderung, so daß seine Größe angepaßt und das Steuerelement plaziert werden können. Das alles übernimmt der Container. Im Code Ihres Steuerelements werden keine Ereignisse der Benutzeroberfläche ausgelöst. Der Container fängt alle Maus- und Tastaturereignisse auf, bevor sie zu Ihrem Steuerelement gelangen.

Wenn die `EditAtDesignTime`-Eigenschaft gleich `True` ist und der Entwickler den BEARBEITEN-Befehl im Kontextmenü auswählt, wird Ihr Steuerelement aktiviert, auch wenn sich der Container im Entwurfsmodus befindet. Ihr Steuerelement erhält Ereignisse der Benutzeroberfläche, die entsprechend verarbeitet werden müssen. Beachten Sie, daß die `UserMode`-Eigenschaft des `Ambient`-Objekts korrekt zeigt, daß sich der Container im Entwurfsmodus befindet. Das Beispiel-Steuerelement `ch17ctlf` in der Programmgruppe `ch17tst1` demonstriert das Ganze.

Alle Ereignisse, die Ihr Steuerelement in diesem Modus im Container aufwirft, werden ignoriert.

17.5.4 Die Enabled-Eigenschaft

Wenn Sie ein `UserControl`-Objekt deaktivieren, werden auch seine Standardelemente deaktiviert, aber sie nehmen nicht das Erscheinungsbild eines deaktivierten Elements an, es sei denn, Sie deaktivieren sie explizit. Das ist identisch mit dem, was passiert, wenn Sie ein Formular deaktivieren. Weitere Informationen über diese Eigenschaft finden Sie in Kapitel 18.

17.5.5 Die EventsFrozen-Eigenschaft

Es gibt Momente im Leben eines Containers, in denen er nicht in der Lage ist, Ereignisse von Steuerelementen entgegenzunehmen. Ein klassisches Beispiel dafür ist, daß Visual-Basic-Formulare keine Ereignisse entgegennehmen können, während ein `MsgBox`-Befehl ein Nachrichtenfeld anzeigt.

Es gibt einige Steuerelemente, die Ereignisse erzeugen können, während die Ereignisse im Container eingefroren sind. Ein gutes Beispiel dafür ist das Timer-Steuerelement.

Wenn diese Eigenschaft gleich `True` ist, werden alle Ereignisse, die Ihr Steuerelement aufwirft, vom Container ignoriert.

Was tun Sie, wenn das Ereignis, das Sie aufwerfen möchten, sehr wichtig ist, und nicht einfach verworfen werden soll?

Sie können das Ereignis in die Warteschlange einreihen, und es später wieder aufwerfen, wenn die Ereignisse wieder aktiviert werden.

Sie können einige Standardmethoden für das Steuerelement definieren, wie es auf das Ereignis reagieren soll, und den Entwickler eine davon auswählen lassen, wenn die Ereignisse eingefroren sind. Wie können Sie erkennen, wann die Ereig-

nisse wieder freigegeben werden? In Visual Basic ist das nicht ganz einfach, aber Sie könnten unter Verwendung eines Timers die `EventsFrozen`-Eigenschaft abfragen. SpyWorks von Desaware bietet eine einfache Methode, Ihr Steuerelement zu benachrichtigen, wenn sich die `EventsFrozen`-Eigenschaft des Containers ändert. Während sich ein Container im Entwurfsmodus befindet, sind Ereignisse immer eingefroren.

17.5.6 Die HasDC-Eigenschaft

Das Zeichnen unter Windows erfolgt über ein Objekt, den sogenannten Gerätekontext. Vor VB6 reservierten die meisten Steuerelemente den Gerätekontext für das Steuerelement, der dann während dessen gesamter Lebensdauer nicht geändert wurde. VB6 erlaubt Ihnen, die `HasDC`-Eigenschaft auf `False` zu setzen.

Wenn Sie häufig API-basierte Techniken verwenden, die `HasDC` auf `False` setzen, kann das Einfluß auf Ihren existierenden Code haben. Wenn Sie den Beispielen in meinem Win32-API-Programmierbuch gefolgt sind, dann sind Sie wahrscheinlich gut gerüstet. Diese Beispiele fordern nämlich immer einen neuen Gerätekontext an, wenn sie einen brauchen, und stellen den Status des Gerätekontexts wieder her, wenn sie beendet werden. Wenn Sie jedoch die `hDC`-Eigenschaft in eine globale Variable oder in eine Variable auf Modulebene geladen haben, können Sie Probleme bekommen, weil sich der Gerätekontext zwischen VB-Ereignissen ändert.

Sie können immer die API-Funktionen `GetDC` oder `CreateDC` verwenden, um einen Gerätekontext zu erhalten, unabhängig vom Status der `HasDC`-Eigenschaft.

Allgemein ausgedrückt, wenn man `HasDC` auf `False` setzt, benötigt man weniger Speicher, allerdings auf Kosten einer leicht verringerten Performance. Wenn diese Eigenschaft nämlich auf `False` gesetzt ist, muß Visual Basic einen Gerätekontext ermitteln und ihn aus den Eigenschaften des Steuerelements komplett initialisieren, wenn es auf ein Steuerelement zeichnen will. Dagegen muß Visual Basic nicht für jedes Steuerelement einen separaten Gerätekontext aufbewahren. Es liegt an Ihnen. Keiner der Ansätze hat besondere Vorteile gegenüber dem jeweils anderen.

17.5.7 Die PaletteMode-Eigenschaft

Neben den allgemeinen Möglichkeiten, die es für Formulare und Steuerelemente gibt, können ActiveX-Steuerelemente so eingerichtet werden, daß sie die Palette des Containers oder überhaupt keine Palette verwenden.

17.5.8 Die Parent-Eigenschaft

Diese Eigenschaft erlaubt Ihnen, auf den Container des Steuerelements zuzugreifen. Sie ist ähnlich den Eigenschaften Container und `Parent` des `Extender`-Objekts von `UserControl`, steht aber immer zur Verfügung. Sie können diese

Eigenschaft nutzen, um festzustellen, in welchem Container Ihr Steuerelement ausgeführt wird. Weitere Informationen darüber, was Sie mit diesem Steuerelement nicht tun sollten, finden Sie in Kapitel 18.

17.5.9 Die ParentControls-Eigenschaft

Diese Eigenschaft ist eine Auflistung der anderen Steuerelemente in dem Container, in dem auch Ihr Steuerelement ausgeführt wird. Es ist nicht möglich, Steuerelemente aus dieser Auflistung zu entfernen oder ihr welche hinzuzufügen. Diese Eigenschaft ist praktisch für solche Steuerelemente, die helfen sollen, andere Steuerelemente auf einem Formular zu verwalten. Beispielsweise könnten Sie sie benutzen, um ein Werkzeug anzulegen, das Steuerelemente ausrichtet oder ihnen allen dieselbe Farbe zuweist.

Nicht jeder Container unterstützt diese Eigenschaft.

Steuerelemente können im Container mit Hilfe der Parent-Eigenschaft und dem folgenden Code erzeugt werden:

```
Dim WithEvents cmd As VB.CommandButton

Private Sub Command1_Click()
   Set cmd = UserControl.Parent.Controls.Add("vb.commandbutton",
"button1")
   cmd.Visible = True
End Sub
```

Ich möchte Ihnen jedoch wirklich abraten, einem Steuerelement zu erlauben, auf diese Weise gleichgeordnete Steuerelemente zu erzeugen. Der Container – und nicht ein anderes Steuerelement – sollte für die Verwaltung seiner Steuerelemente verantwortlich sein.

17.5.10 Die Public-Eigenschaft

Wenn die `Public`-Eigenschaft gleich `True` ist, wird das Steuerelement für die Verwendung durch andere Applikationen freigegeben. Es ist möglich, private ActiveX-Steuerelemente zu erzeugen, die innerhalb von Programmdateien oder anderen ActiveX-Steuerelementen verwendet werden.

Bei der Entscheidung, ob Sie ein ActiveX-Steuerelement öffentlich oder privat machen sollten, gehen Sie genauso vor wie bei der Entscheidung, ob Sie eine Klasse privat machen oder sie über eine ActiveX-DLL-Codekomponente bereitstellen sollen.

17.5.11 Die RightToLeft-Eigenschaft

Wenn Ihr Steuerelement unter einer Windows-Version ausgeführt wird, die die Formatierung von rechts nach links unterstützt (beispielsweise das hebräische

oder arabische Windows), ändert diese Eigenschaft die Richtung der Textausgabe auf dem Steuerelement. Wenn Sie diese Möglichkeit unterstützen möchten, sollten Sie diese Eigenschaft basierend auf der `RightToLeft`-Eigenschaft von `AmbientProperties` setzen.

17.5.12 Die ToolBoxBitmap-Eigenschaft

Wählt eine 16x15-Pixel-Bitmap aus, die das Steuerelement in der Symbolleiste des Containers repräsentiert.

Das untere linke Pixel definiert die transparente Farbe für das Steuerelement (der Hintergrund der Symbolleiste scheint durch die Teile der Bitmap, die auf diese Farbe gesetzt sind). In der Regel verwende ich eine bestimmte Farbe, beispielsweise Grün, um den Hintergrund darzustellen, dann mache ich nicht versehentlich eine Farbe transparent.

Nachdem Sie nun das `UserControl`-Objekt kennengelernt haben, werden wir uns noch dem Rest zuwenden: den `Extender`- und `Ambient`-Objekten.