Kapitel 11

Ereignisse

Wie haben wir das nur geschafft, nun schon bei Kapitel 11 eines Buches über das Entwickeln von ActiveX-Komponenten angelangt zu sein und dabei noch keinmal Ereignisse erwähnt zu haben? Nun, das stimmt so nicht ganz – Sie wissen bereits eine ganze Menge über Ereignisse und wie sie funktionieren.

Wie ist das denn zu verstehen? Na, lesen Sie doch weiter ...

11.1 Methoden unter anderem Namen

Blicken wir zurück: Worin besteht der Unterschied zwischen einer Methode und einer Eigenschaft?

- Die Syntax des Aufrufs ist in Visual Basic leicht unterschiedlich.
- Eine Eigenschaft wird mit Hilfe von zwei Methoden implementiert: mit einer Methode zum Setzen und mit einer zum Lesen.

Im Prinzip sind Methoden und Eigenschaften doch nichts anderes als zwei nur leicht unterschiedliche Wege, Funktionen aufzurufen, die zu einem COM-Interface gehören.

Es herrscht in den meisten ActiveX-Dokumentationen (einschließlich der von Visual Basic) die Tendenz vor, Ereignisse als einen in sich abgeschlossenen Bereich zu behandeln. In Wirklichkeit jedoch sind sie das gleiche wie Methoden und Eigenschaften – und wiederum eine nur leicht veränderte Art und Weise des Aufrufs von Funktionen eines COM-Interfaces.

Der einzige wirkliche Unterschied liegt in der Perspektive. Wenn Ihre Anwendung als Client eine Methode eines Objekts aufruft, wird diese eben »Methode« genannt. Ruft umgekehrt das Objekt eine Methode eines der Objekte Ihrer Anwendung auf, dann wird das »Ereignis« genannt.

Halten wir kurz inne und schauen uns an, wie das eigentlich funktioniert, ohne uns vorläufig um den Begriff »Ereignis« zu kümmern.

11.1.1 OLE-Rückrufe

Stellen Sie sich eine Client-Anwendung vor, die ein Objekt verwendet, das von einer DLL- oder einer EXE-Code-Komponente angeboten wird. Dieses Objekt soll eine Operation im Hintergrund ausführen. Wenn diese Hintergrundoperation abgeschlossen ist, soll das Objekt auf irgendeine Weise die Fertigstellung an Ihre Anwendung melden. In Visual Basic 4.0 war die einzige Möglichkeit, dies zu bewerkstelligen, die Technik der OLE-Rückrufe.

Das Szenario dafür sehen Sie in Abbildung 11.1. Zu Beginn steht die Wahl eines Objekts in einer Anwendung, das die Benachrichtigung entgegennehmen soll. Fügen Sie in dieses Objekt eine Methode ein, die zur Signalisierung des Ereignisses dienen soll. Im Beispiel `TickTst1.vbp` heißt diese Methode `DelayedCall`.

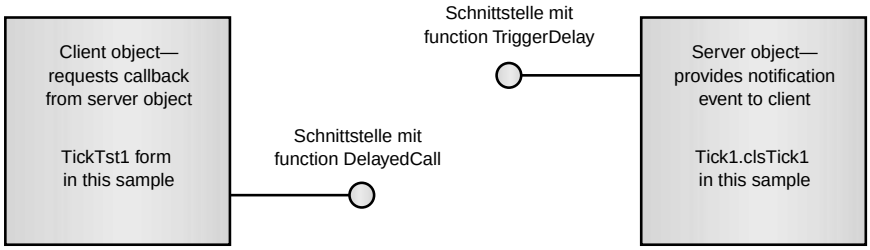

Abb. 11.1: Szenario eines OLE-Rückrufs

Das Formular `TickTst1` verwendet ein Objekt, das von der Code-Komponente `Tick1.vbp` angeboten wird. Die Anwendung legt eine Instanz dieses Objekts an, wenn sie geladen wird (und gibt diese beim Anwendungs-Ende wieder frei). Hier wurde späte Bindung gewählt, die frühe Bindung könnte jedoch genausogut implementiert werden. Sie werden gleich sehen, warum ich für dieses Beispiel einen EXE-Server vorgesehen habe.

```
Dim TimerObj As Object

Private Sub Form_Load()
   ' Sicherstellen, daß Tick1 läuft
   Set TimerObj = CreateObject("Tick1.clsTick1")
End Sub

Private Sub Form_Unload(Cancel As Integer)
   Set TimerObj = Nothing
End Sub
```

Das Objekt `Tick1.clsTick1` ist recht einfach gehalten. Es setzt einen Timer auf eine bestimmte Verzögerung und benachrichtigt Ihr Formular, wenn die eingestellte Zeit verstrichen ist, indem die `DelayedCall`-Methode des Formulars aufgerufen wird. Wie kann denn die Methode `DelayedCall` in Ihrem Formular aufgerufen werden? Offensichtlich müssen Sie dem Objekt eine Referenz auf das Formular übergeben, die es solange behält, bis der Timer abgelaufen ist. Dies geschieht über die Methode `TimerObj.TriggerDelay`, der eine Referenz auf das Formular als Parameter übergeben wird.

Die Schaltfläche `cmdTest` bleibt derweil gesperrt, damit die Methode `TriggerDelay` nicht erneut aufgerufen werden kann, während die vorhergehende Operation noch in Bearbeitung ist. Das Beispiel `Tick1` – ein sehr simples Programm –, ist nicht darauf vorbereitet, einen zwischenzeitlichen erneuten Aufruf zu verkraften. Die Funktion `cmdTest_Click` und den zugeordneten Rückruf sehen Sie hier:

```
Private Sub cmdTest_Click()
    ' 2000 ms Verzögerung
    TimerObj.TriggerDelay Me, 2000
    cmdTest.Enabled = False
End Sub

Public Sub DelayedCall()
    MsgBox "DelayedCall 'Event' received"
    cmdTest.Enabled = True
End Sub
```

Schauen Sie noch einmal auf Abbildung 11.1. Dort sehen Sie die beiden Objekte. Das Formular `TickTst1` enthält die `DelayedCall`-Methode in ihrem Standard-Interface. Das Objekt `clsTick1` enthält in seinem Standard-Interface die Methode `TriggerDelay`. Das `Client`-Objekt `TickTst1` verfügt über eine Referenz auf das `clsTick1`-Objekt und ruft dessen `TriggerDelay`-Methode auf. Bei diesem Aufruf wird eine Referenz auf das Formular `TickTst1` als einer der Parameter übergeben. Das Objekt hält diese Referenz in einer privaten Variablen fest. Hat es seine Arbeit abgeschlossen, kann es somit die Methode `DelayedCall` des Formulars aufrufen.

Wo ist nun das Ereignis? Beide Komponenten führen die gleiche Art von Operation aus – den Aufruf einer Methode. Jedoch entspricht der Aufruf von `DelayedCall` einem Ereignis, da er als Folge eines Vorkommnisses im Server erfolgt. Sie sehen, es ist alles nur eine Frage der Perspektive.

Schauen wir uns nun das Objekt `clsTick1` näher an. Ich mache Sie darauf aufmerksam: Dieses Objekt, wenngleich es nur einen geringen Umfang hat, nutzt eine ganze Reihe fortgeschrittener Techniken. Einige davon haben wir bereits besprochen, auf einige werden wir später noch eingehen.

Das Projekt Tick1

Dieses Projekt stellt eine einfache Zeitgeber-Anwendung dar. Sie übergeben den Wert einer Zeitverzögerung, und nachdem diese Zeit verstrichen ist, wird Ihr `Client`-Objekt benachrichtigt. In der Praxis würden Sie diese Funktionalität vielleicht direkt in Ihrer Anwendung implementieren. Da ein Zeit-Ereignis auf ideale Weise ein externes Ereignis repräsentiert, ist dies jedoch ein ganz gutes Beispiel. Die hier gezeigten Prinzipien können Sie direkt in Anwendungen wie folgenden einsetzen:

- Warten auf Antwort auf eine Fernabfrage
- Warten auf das Terminieren eines Kind-Prozesses
- Warten auf ein System-Ereignis
- Warten auf den Abschluß eines Downloads über das Internet

Dieses Projekt enthält eine einzige Klasse (`clsTick1.cls`) und ein Standard-Modul (`modTick1.bas`). Das Klassen-Modul enthält zwei private Variablen. `DelayToUse` nimmt den Verzögerungswert auf. Die Variable `CallbackObject` nimmt die Referenz auf das `Client`-Objekt auf. Dieses `Client`-Objekt muß über eine Methode namens `DelayedCall` verfügen. Sollte dies nicht der Fall sein, wird ein Fehler ausgelöst.

Beachten Sie, daß diese Referenz spät gebunden wird, da das Objekt von jedem beliebigen `Client`-Objekt verwendet werden soll, das die Methode `DelayedCall` implementiert hat. Könnte man das in eine frühe Bindung umwandeln? Ja, indem Sie ein Interface definieren, das die Methode `DelayedCall` enthält. Sie können dann in jeder Client-Anwendung, die diesen Server verwenden soll, dieses Interface im aufrufenden Objekt über die Anweisung `Implements` implementieren. Hier ist das betreffende Objekt jedoch in einem EXE-Server angelegt, so daß der Performance-Vorteil der frühen Bindung gegenüber dem Out-of-Process-Overhead vernachlässigbar ist.

Die Methode TriggerDelay speichert die Referenz auf das `Client`-Objekt in der Variablen `CallbackObject`. Dann ruft sie die Funktion `StartTimer` im Modul `modTick1` (mehr dazu gleich) auf. Die Timer-Geschichte ist in diesem Modul implementiert – dort wird nun noch eine Möglichkeit benötigt, das Objekt `clsTick1` zu benachrichtigen, wenn die eingestellte Zeit abgelaufen ist. Sie wissen nun schon, wie das geht: mit einem weiteren OLE-Rückruf! Die Methode `StartTimer` übernimmt also eine Referenz auf das Objekt `clsTick1` und speichert sie im Modul. Wenn die Zeit abgelaufen ist, ruft das Modul die Methode `TimerExpired` des Klassen-Objekts auf. Die Methode `TimerExpired` wiederum ruft die Methode `DelayedCall` des Objekts in `CallbackObject` auf.

Woher weiß das Modul, welche Zeitverzögerung gewünscht ist? Es gibt zwei naheliegende Möglichkeiten. Sie können den Verzögerungswert als Parameter der `StartTimer`-Methode übergeben. Doch hier ergibt sich die Gelegenheit, einmal den Wert der `Friend`-Deklaration zu demonstrieren. Indem die Eigenschaft `Delay` als `Friend` deklariert wird, können andere Module in diesem Projekt den Wert der privaten Variablen `DelayToUse` abfragen, ohne daß dieser der übrigen Welt gegenüber offengelegt wird. In Listing 11.1 sehen Sie das Modul `clsTick1`.

```
' Guide to Perplexed: Tick1
' Copyright (c) 1997 by Desaware Inc. All Rights Reserved
'
Option Explicit

Private DelayToUse As Long
Private CallbackObject As Object

' Callbackparam ist ein Objekt, das über die
' Methode 'DelayedCall' verfügt
Public Sub TriggerDelay(Callbackparam As Object, _
 delayval As Long)
```

```
    DelayToUse = delayval
    ' Dies ist das Objekt, das wir zurückrufen wollen
    Set CallbackObject = Callbackparam
    StartTimer Me
End Sub

' Dem Standard-Modul den Zugriff auf den
' Verzögerungswert erlauben.
' Der Zugriff ist jedoch nicht öffentlich
Friend Property Get Delay() As Long
    Delay = DelayToUse
End Property

' Dies ist die Nachricht über den Ablauf des Timers
' im Standard-Modul
Friend Sub TimerExpired()
    ' Späte Bindung
    CallbackObject.DelayedCall
    ' Die Referenz auf das Objekt nicht unnötig behalten
    Set CallbackObject = Nothing
End Sub
```

Listing 11.1: Die Klasse clsTick1

Wie arbeitet nun der Timer selbst? Er greift auf die Timer-Fähigkeiten von Windows zurück. Dies sehen Sie in Listing 11.2. Zwei API-Funktionen sind dort deklariert: `SetTimer` und `KillTimer`. `SetTimer` legt ein `Timer`-Objekt mit einer angegebenen Verzögerung an und gibt eine Kennung für diesen Timer zurück. `KillTimer` zerstört das zu der Kennung gehörende `Timer`-Objekt.

```
' Guide to Perplexed: Tick1
' Copyright (c) 1997 by Desaware Inc. All Rights Reserved

Option Explicit

' Timer-Kennung
Dim TimerID&

' Objekt für diesen Timer
Dim TimerObject As clsTick1

Declare Function SetTimer Lib "user32" (ByVal hwnd _
 As Long, ByVal nIDEvent As Long, ByVal uElapse _
 As Long, ByVal lpTimerFunc As Long) As Long
Declare Function KillTimer Lib "user32" (ByVal hwnd _
 As Long, ByVal nIDEvent As Long) As Long
```

```
Public Sub StartTimer(callingobject As clsTick1)
   Set TimerObject = callingobject
   TimerID = SetTimer(0, 0, callingobject.Delay, _
    AddressOf TimerProc)
End Sub

' Windows-Callback-Funktion
Public Sub TimerProc(ByVal hwnd&, ByVal msg&, _
 ByVal id&, ByVal currentime&)
   Call KillTimer(0, TimerID)
   TimerID = 0
   TimerObject.TimerExpired
   ' Objekt-Referenz freigeben, damit es terminieren
   ' kann
   Set TimerObject = Nothing
End Sub
```

Listing 11.2: Das Modul modTick1.bas

Die Methode `StartTimer` speichert eine Referenz auf das aufrufende Objekt (das über die Methode `TimerExpired` verfügt) in der Variablen `TimerObject`. In diesem Fall ist das Rückruf-Objekt (`TimerObject`) früh gebunden, da wir hier ja wissen, daß es den einzigen Objekt-Typ darstellt, der die Methode `StartTimer` aufrufen kann – hier besteht keine Notwendigkeit, ein eigenes Interface anzulegen.

Die Funktion `SetTimer` erwartet einen Zeiger auf eine Funktion, die aufgerufen werden soll, wenn der Timer abgelaufen ist. Eine Funktions-Adresse einer Funktion in einem Standard-Modul kann über den `AddressOf`-Operator ermittelt werden. Es ist wichtig, daß diese Modul-Funktion mit den richtigen Parametern, Parameter-Typen und Rückgabewerten angelegt wird, damit keine Speicherausnahme auftritt. Dies schließt auch die korrekten Qualifizierer `ByVal` und `ByRef` für jeden Parameter ein.

Windows hält seinerseits die von der `SetTimer`-Funktion übergebene Funktions-Adresse fest und ruft diese Funktion (hier die Funktion `TimerProc`) auf, wenn der Timer abgelaufen ist. Die Funktion `TimerProc` löscht den Timer, ruft die Methode `TimerExpired` des aufrufenden Objekts auf und gibt die Referenz in der Variablen `TimerObject` wieder frei, damit das Objekt sauber terminieren kann.

Bei näherer Betrachtung zeigt sich, daß das Konzept des Festhaltens einer Funktions-Adresse und deren späterer Aufruf vom Konzept dasselbe ist wie das Festhalten einer Objekt-Referenz und des Aufrufs einer Methode dieses Objekts zu einem späteren Zeitpunkt. Tatsächlich wird dieser Typ einer Operation, bei dem eine Funktions-Adresse an Windows bzw. eine DLL für einen späteren Aufruf übergeben wird, »Callback« (deutsch »Rückruf«) genannt. Die Funktion, deren Adresse übergeben wurde, wird daher »Rückruf-Funktion« genannt. Allerdings muß deutlich zwischen einem API-Rückruf dieses Typs und einem OLE-Rückruf

unterschieden werden. Sie mögen sich zwar konzeptionell ähneln, werden aber auf völlig andere Art und Weise implementiert.

Das hier gezeigte Szenario können Sie ausprobieren, indem Sie die Projekte `Tick1` und `TickTst1` in jeweils einer eigenen Instanz der Visual Basic-Entwicklungsumgebung laden. Starten Sie das Projekt `Tick1`, um dessen Objekt verfügbar werden zu lassen. Vergewissern Sie sich, daß das Projekt `TickTst1` über einen Verweis auf das `Tick1`-Projekt verfügt. Starten Sie nun die Anwendung `TickTst1` und klicken Sie auf die TEST-Schaltfläche. Das in Abbildung 11.2 dargestellte Szenario wird ablaufen und es wird eine MessageBox nach etwa 2 Sekunden geöffnet.

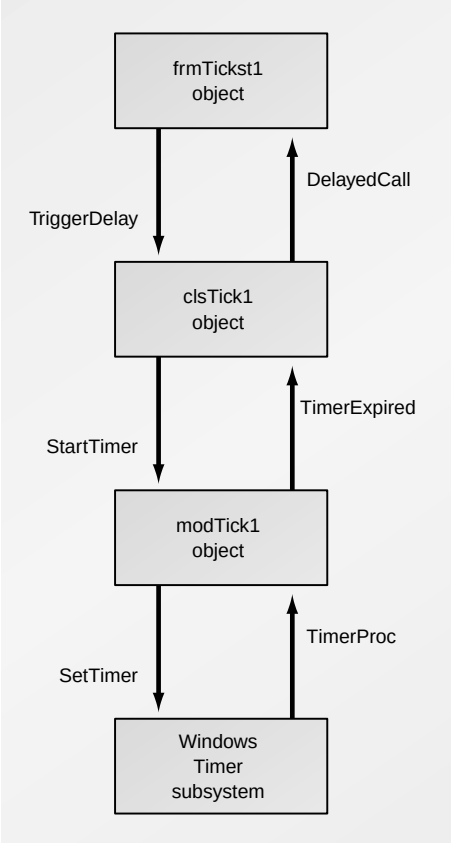

Abb. 11.2: Ablauf des Szenarios der Objekt- und Methoden-Aufrufe im Beispiel TickTst1/Tick1

Warum haben wir das Objekt `Tick1` als EXE-Server mit der Einstellung der `Instancing`-Eigenschaft auf `SingleUse` angelegt?

Ein Windows-Timer erfordert die Verwendung eines Standard-Moduls für die API-Rückruf-Funktion. Würde die Komponente das Objekt `Tick1` mehrfach anlegen, würde es erforderlich werden, in dem Standard-Modul die verschiedenen Instanzen des Objekts nachzuhalten und mehrere Timer zu verwalten. Dies ist natürlich möglich – ich werde Ihnen später noch zeigen, wie das genau geht. Doch für dieses Beispiel wäre dies ein wenig zu komplex geworden. Da also das Objekt als `SingleUse` gesetzt ist, wird jede Instanz des Objekts in ihrem eigenen Prozeßraum laufen und seine eigenen globalen Variablen haben. Damit erübrigen sich die Sorgen über die Verwaltung mehrerer Objekte – das geschieht automatisch. Die Möglichkeit, ein `SingleUse`-Objekt anzulegen, gibt es nur in einem EXE-Server.

Wenn Sie mit den Projekten `TickTst1` und `Tick1` experimentieren, müssen Sie das `Tick1`-Projekt jedesmal stoppen und neu starten, wenn Sie das Projekt `TickTst1` gestartet haben und `Tick1` in einer Visual Basic-Entwicklungsumgebung läuft. Visual Basic kann nur ein einziges Objekt einer Klasse anlegen, die als `SingleUse` gesetzt ist, und dies auch nur einmal nach einem Start. Sie können `Tick1` in eine ausführbare Datei kompilieren, um das Problem zu umgehen.

Wenn Sie aber tatsächlich in der Praxis ein `Timer`-Objekt erstellen wollen, wäre der hier gezeigte Ansatz der letzte, den Sie verfolgen sollten. Es ist schließlich sehr ineffizient, jedesmal einen neuen Prozeß zu starten, um eine Zeitgeber-Funktion zu starten. Aber wir wollten ja mit der Zeitverzögerung eigentlich andere, Zeit konsumierende Hintergrundprozesse simulieren. Die oben angeführten derartigen Prozesse werden dagegen häufig auf diese Weise implementiert. In den nachfolgenden Kapiteln, insbesondere in Kapitel 14, »Multithreading«, werden Sie noch einiges mehr über die Vor- und Nachteile dieser Technik erfahren. Wenden wir uns nun aber wieder dem eigentlichen Thema zu, den Ereignissen.

Noch einmal: Ereignisse

OLE-Rückrufe bieten eine Möglichkeit, ein Objekt darüber zu benachrichtigen, wenn ein Ereignis in einem anderen Objekt eingetreten ist. Der Begriff *Ereignis* bezieht sich in diesem Zusammenhang auf nichts anderes als auf den Grundgedanken der Benachrichtigung eines Clients durch einen Server, wenn irgend etwas geschehen ist.

Wenn Sie jedoch in der ActiveX- oder in der Visual-Basic-Dokumentation blättern, werden Sie auf die Verwendung des Begriffs *Ereignis* in einem anderen Zusammenhang der Benachrichtigung eines Clients durch einen Server stoßen. Diese Technik wird in den Beispiel-Projekten `Tick2.vbp` und `TickTst2.vbp` demonstriert. Diese beiden Projekte sind zunächst durch schlichtes Kopieren der Projekte `Tick1` und `TickTst1` samt der dazugehörenden Module entstanden. Dann wurden die Module umbenannt (aus `clsTick1` wurde `clsTick2`) und anschließend wurde der Code geändert. Dies geschah zur besseren Unterscheidung der beiden Beispiele und zur Vermeidung von Konfusion bei der Wiederverwendung der Klassen-Namen.

Um Tick2 ereignisfähig zu machen, brauchen Sie lediglich die folgende Zeile in den Allgemein-Bereich des Moduls clsTick2 einzugeben:

```
Event DelayedCall()
```

Weiterhin ist die Funktion TimerExpired wie folgt zu modifizieren:

```
Friend Sub TimerExpired()
   ' Späte Bindung
   If Not CallbackObject Is Nothing Then
      CallbackObject.DelayedCall
      ' Referenz auf das Objekt freigeben
      Set CallbackObject = Nothing
   Else
      RaiseEvent DelayedCall
   End If
End Sub
```

Wie Sie sehen können, unterstützt das Objekt clsTick2 weiterhin OLE-Rückrufe. Sie brauchen lediglich den »Wert« Nothing als Referenz im Parameter an die Methode TriggerDelay übergeben. Dies weist die Komponente an, das Ereignis DelayedCall auszulösen, anstatt die Methode DelayedCall irgendwo anders aufzurufen.

Die Änderungen in der Client-Test-Anwendung sind grundlegender, aber auch nur minimal. Den Code des Formulars sehen Sie in Listing 11.3.

```
 ' Guide to the Perplexed: TickTst2
' Copyright (c) 1997 by Desaware Inc. All Rights Reserved

Option Explicit

Dim WithEvents TimerObj As Tick2.clsTick2

Private Sub cmdTest_Click()
   ' 2000 ms Verzögerung
   TimerObj.TriggerDelay Nothing, 2000
   cmdTest.Enabled = False
End Sub

Private Sub Form_Load()
   Set TimerObj = New clsTick2
End Sub

Private Sub Form_Unload(Cancel As Integer)
   Set TimerObj = Nothing
End Sub

Private Sub TimerObj_DelayedCall()
```

```
    MsgBox "DelayedCall 'Event' received"
    cmdTest.Enabled = True
End Sub
```

Listing 11.3: Der Code des Formulars frmTickTst2

Der größte Unterschied besteht darin, daß Sie bei der Verwendung von Ereignissen frühe Bindung benutzen müssen. Daher müssen Sie vor dem Start der Test-Anwendung zunächst `Tick2` in einer Instanz einer Visual Basic-Entwicklungsumgebung starten (entfällt, wenn Sie `Tick2` bereits kompiliert haben) und über den VERWEISE-Dialog einen Verweis auf `Tick2` in das `TickTst2`-Projekt aufnehmen. Sie müssen dies auch tun, wenn Sie den auf der Buch-CD enthaltenen Beispiel-Code starten möchten. Nach der Aufnahme des Verweises in das Projekt können Sie ein Objekt der Klasse mit der Anweisung `Dim WithEvents` deklarieren.

Frage: Warum ist in Listing 11.3 der qualifizierte Klassen-Name `Tick2.clsTick2` angegeben, anstatt nur `clsTick2`?

Antwort: Es gibt keinen funktional bedingten Grund dafür. Es dient lediglich der Unterscheidung und der Erhöhung der Lesbarkeit dieses Beispiels – wir waren ja gerade eben noch mit dem Projekt `Tick1` befaßt.

Im `Form_Load`-Ereignis wird das Objekt angelegt. Sie können bei der Deklaration die `New`-Anweisung nicht in Kombination mit `WithEvents` verwenden. Ich weiß nicht, warum das so ist – vielleicht wird das in künftigen Versionen noch geändert.

Wenn das Objekt `clsTick2` ein Ereignis über die Anweisung `RaiseEvent` auslöst, wird dabei die Methode `DelayedCall` in einem speziellen Ereignis-Interface aufgerufen, das in dem Formular zusammen mit dem Namen der deklarierten Variablen (hier: `TimerObj`) erscheint.

Was hat sich hier nun eigentlich geändert?

- Wir verwenden frühe statt späte Bindung. Dies ist jedoch keine wesentliche Änderung, da wir auch bereits in `Tick1` frühe Bindung hätten verwenden können.
- Wir haben eine Objekt-Variable mit `WithEvents` deklariert.
- Wir verwenden im Objekt `Tick2` die Anweisung `RaiseEvent` anstelle des direkten Methoden-Aufrufs.
- Die Syntax des Ereignisses lautet etwas anders.

Wenn die beiden Ansätze ziemlich ähnlich erscheinen mögen, liegt das daran, daß sie auch tatsächlich ähnlich sind. ActiveX-Ereignisse sind tatsächlich nichts anderes als ein Weg, OLE einiges der Arbeit beim Implementieren eines Rückruf für Sie erledigen zu lassen. Sie werden gleich sehen, was hinter den Kulissen geschieht. Kurz gesagt nur soviel: Das Server-Objekt ruft eine Methode des

Client-Objekts auf. Ob es ein OLE-Rückruf oder ein OLE-Ereignis ist, tut nichts zur Sache – es handelt sich in jedem Fall um einen Aufruf einer Methode. Und Methoden-Aufrufe kennen Sie ja schon längst.

Sie sehen, ich habe nicht gescherzt, als ich behauptete, daß Sie eigentlich schon alles über Ereignisse wissen müßten.

11.2 Und wieder COM

Was tut nun OLE tatsächlich hinter den Kulissen, damit die Verwendung von OLE-Ereignissen so einfach wird? Schauen wir uns zunächst in Tabelle 11.1 die Unterschiede zwischen Ereignissen und Rückrufen bezüglich ihrer Verwendung an.

OLE-Ereignisse	**OLE-Rückrufe**
Frühe Bindung[1]	Späte oder frühe Bindung[1]
Ausdrückliche Übergabe einer Objekt-Referenz an das Server-Objekt ist nicht notwendig	Objekt-Referenz muß an den Server für den Rückruf übergeben werden
Visual Basic legt automatisch Ereignis-Prozedur-Rümpfe im Code-Fenster an, wenn das Ereignis ausgewählt wird	Rückruf-Methoden müssen manuell angelegt werden
Die `AutoList`-Funktionalität in Visual Basic erleichtert die Wahl von auszulösenden Ereignissen im Client-Objekt	Der Server hat keinen Zugriff auf die Typ-Informationen des Clients
Separate Ereignis-Prozedur für jedes mit `WithEvents` deklarierte Objekt	Alle Objekte des gleichen Typs werden die gleiche Rückruf-Methode aufrufen. Sie können zur Unterscheidung zusätzliche Parameter zusammen mit dem Ereignis übergeben (z.B. Sie können die dem Server-Objekt übergebene Referenz des Client-Objekts wiederum mit dem Rückruf zurückgeben und mit `If obj Is Me` prüfen)

Tab. 11.1: *Merkmale von OLE-Rückrufen und OLE-Ereignissen*

1. Frühe Bindung meint in diesem Zusammenhang, daß ein Verweis auf das Server-Objekt in die Client-Anwendung aufgenommen wird, und daß auf Methoden und Eigenschaft des Objekts über frühe Bindung zugegriffen werden kann. Es bedeutet NICHT, daß die Ereignisse selbst früh gebunden sind. Ob die Anweisung `RaiseEvent` frühe oder späte Bindung zur Auslösung des Ereignisses verwendet, ist ein Geheimnis der internen Implementierung seitens Visual Basic.

11.2.1 Hinter den Kulissen

Wie implementiert OLE Ereignisse? Folgende Schritte müssen auf jeden Fall abgearbeitet werden:

- Der Client fragt den Server, ob er Ereignisse unterstützt.
- Das Client-Objekt muß vom Server-Objekt eine Liste der unterstützten Ereignisse samt deren Parameter erhalten.
- Das Client-Objekt muß ein neues Interface anlegen, das die Methoden enthält, die den Ereignissen des Servers entsprechen.
- Das Client-Objekt muß dem Server eine Referenz auf das neue Interface übergeben.

Nun werden Sie vielleicht ahnen, warum es sieben Kapitel gedauert hat, um über diesen Punkt sprechen zu können. Sie wissen, daß COM mehr als nur ein Interface unterstützen kann. Sie wissen, daß Sie mehrere Interfaces für ein Objekt mit Hilfe der `Implements`-Anweisung anlegen können. Alles was hier geschieht ist, daß Visual Basic für jedes mit `WithEvents` deklarierte Objekt ein neues Interface anlegt. Die Methoden werden auf der Basis der vom Server übermittelten Informationen angelegt (daher gibt der Server die Namen und Parameter der Ereignisse vor – was natürlich Sinn macht, da der Server ja diese Ereignisse aufrufen soll). Visual Basic gibt eine Referenz auf dieses Interface an das Server-Objekt zurück, das nun diese Methoden aufrufen kann.

In Abbildung 11.3 sehen Sie, wie OLE-Objekte, die Ereignisse unterstützen, häufig dargestellt werden. Ein abgehender Pfeil stellt ein »Outgoing Interface« dar und zeigt, daß das Objekt Methoden eines Interfaces aufrufen kann, das von ihm definiert wird. Das Server-Objekt legt jedoch dieses nicht selbst an. Es stellt lediglich eine Spezifikation des Interfaces dar, das das Client-Objekt anlegen muß, um Ereignisse empfangen zu können.

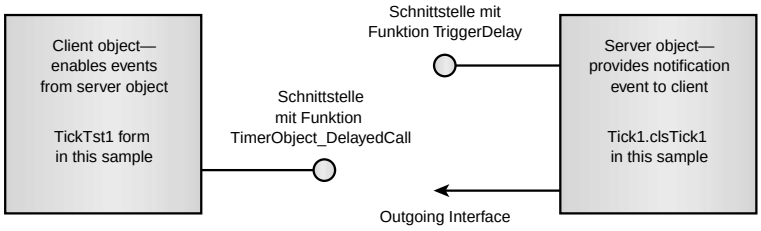

Abb. 11.3: Implementierung von OLE-Ereignissen

Wie erhält der Client diese Interface-Spezifikation? Er ruft `QueryInterface` auf, um vom Server eine Referenz auf ein Interface namens `IConnectionPointContainer` zu erhalten. Dieses Interface bietet Methoden an, über die ein Objekt wiederum Objekte mit einem Interface namens `IConnectionPoint` erhalten kann.

IConnectionPoint-Objekte verfügen über Methoden, über die ein Objekt Informationen über Rückgabewert und Parameter einer Methode erfragen kann. Das Client-Objekt verwendet diese Informationen zum Anlegen des Ereignis-Interfaces. Natürlich brauchen Sie sich in Visual Basic um all diese Dinge nicht zu kümmern. Es geschieht alles hinter den Kulissen. Wenn Sie mehr darüber erfahren möchten, finden Sie Informationen darüber auf den Microsoft Developer's Network-CDs (MSDN).

Ist es nicht erstaunlich, wie viele Interfaces ein einfaches Objekt haben kann? Ob Sie es glauben oder nicht – noch vor wenigen Jahren mußte man all dies noch von Hand in C oder C++ codieren, um OLE-Objekte verwenden zu können! Kein Wunder, daß OLE als kompliziert verschrien war ...

Im vorangegangenen Abschnitt sahen Sie, daß das Rückruf-Objekt als einfache Objekt-Variable deklariert worden war, damit der Server mit jedem beliebigen Objekt umgehen konnte. Ich habe bereits angedeutet, daß Sie auch ein eigenes Interface definieren können, das von jedem Client unterstützt werden kann, der diesen Server verwenden möchte. Wenn Sie lediglich ein einziges Objekt verwenden, ergibt das den gleichen Effekt wie der OLE-Ereignis-Mechanismus. Doch das ändert sich, wenn Sie mehrere Objekte des gleichen Typs verwenden wollen.

Überlegen Sie einmal, was geschehen würde, wenn im Beispiel TickTst2 ein zweites clsTick2-Objekt namens TimerObject2 angelegt würden Sie erhielten eine zweite Ereignis-Prozedur mit dem Namen TimerObject2_DelayedCall. Dies sehen Sie in Abbildung 11.4. Der Client TickTst2 legt für jedes mit WithEvents deklarierte Objekt ein eigenes Interface an.

Bei OLE-Rückrufen haben Sie die Wahl. Sie können im Server die Variable für das Rückruf-Objekt als As Object deklarieren und eine Methode des Hauptinterfaces des Clients aufrufen. Sie können auch ein eigenes Interface definieren, das von jedem Client-Objekt implementiert werden kann, das den Server verwendet, und dann auf dieses Interface zugreifen. Wie auch immer – es gibt jedoch keine Möglichkeit für den Client, für jedes Server-Objekt ein separates Interface anzulegen. Egal wie viele Server-Objekte Sie anlegen, die Server-Objekte werden alle dieselbe Methode aufrufen. Dies sehen Sie in Abbildung 11.5.

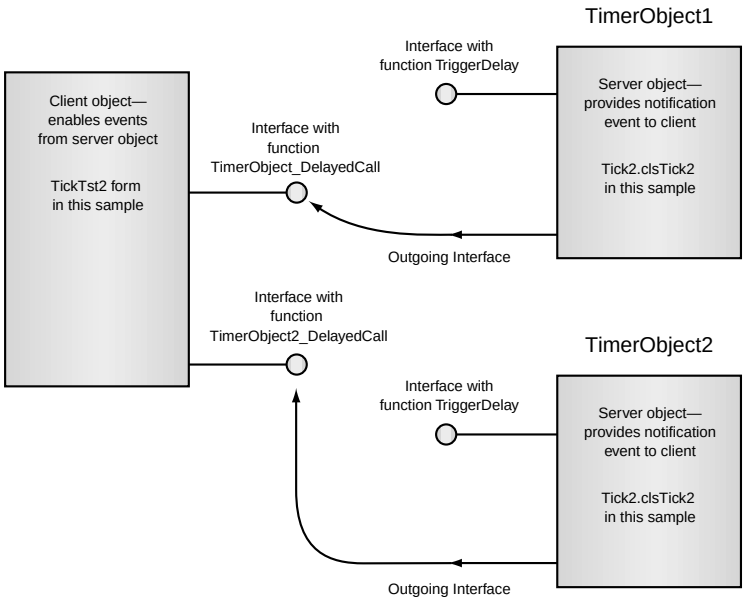

Abb. 11.4: OLE-Ereignisse bei zwei Objekten

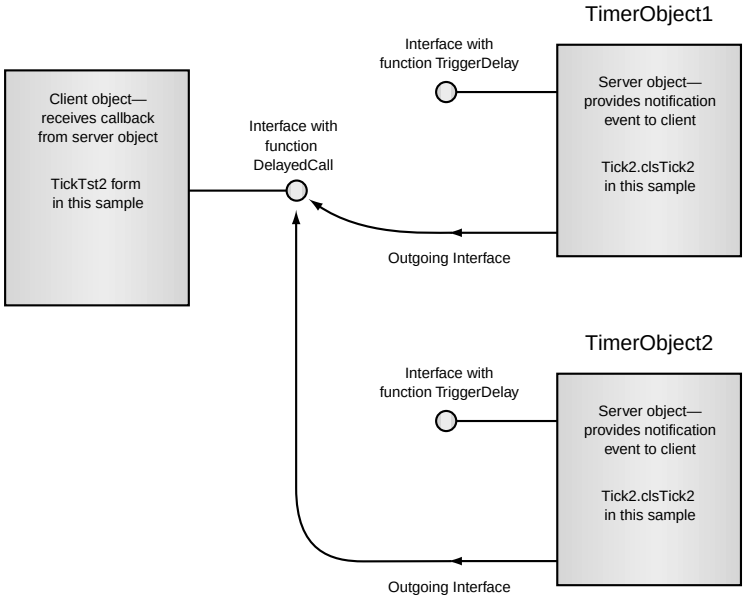

Abb. 11.5: OLE-Rückrufe mit zwei Objekten

11.2.2 Separate Ereignisse mit OLE-Rückrufen

Am einfachsten kann zwischen Server-Objekten unterschieden werden, wenn der Server selbst Informationen als Parameter an die Client-Methode zurückgibt. Aber auch wenn Sie ein Objekt verwenden, das diesen Service nicht bietet, können Sie trotzdem mehrere Objekte verwenden und zwischen diesen unterscheiden.

In der Projektgruppe `callbk1.vbg` wird gezeigt, wie das geht. Diese Projektgruppe enthält zwei Projekte, einen ActiveX-DLL-Server und ein Test-Projekt.

Das Server-Projekt, `Callback1Server` (`Callbk1s.vbp`), enthält eine einfache Klasse, die einen OLE-Rückruf verwendet, um ein Ereignis zu signalisieren. Die Funktion `SetCallback` übernimmt die Objekt-Referenz des Clients als Parameter und speichert diese in einer privaten Variablen. Die Funktion `EventDemo` ist eine einfache Test-Funktion, die das Ereignis über den Rückruf auslöst. Den Code des Objekts sehen Sie hier:

```
' Guide to the Perplexed: Callback1 Sample
' Copyright (c) 1997 by Desaware Inc.  All Rights Reserved

Option Explicit

Private CallbackObject As Object

' Beispiel-Funktion, die den Rückruf setzt
Public Sub SetCallback(myobject As Object)
   Set CallbackObject = myobject
End Sub

' Beispiel-Funktion, die den Rückruf auslöst
Public Sub EventDemo()
   CallbackObject.SampleEvent "Received an event"
End Sub
```

Die `Instancing`-Eigenschaft der in einer ActiveX-DLL implementierten Klasse ist auf `MultiUse` gesetzt. Dies bedeutet, daß der Client beliebig viele dieser Objekte anlegen kann. Jedes dieser Objekte ruft als Ereignis-Meldung die Methode `SampleEvent` im Client-Objekt auf. Da die Methode SampleEvent keine Parameter hat, kann der Client nicht unterscheiden, welches Objekt den Rückruf vorgenommen hat.

Die einzige Lösung zur Unterscheidung zwischen verschiedenen Rückruf-Objekten ist naheliegend: Man nehme einen separaten Client für jedes Rückruf-Objekt. Dies wird im Projekt `Callback1` gezeigt. Dort wird eine Klasse namens `clsCallbackSink` definiert, deren einziger Zweck darin besteht, als Empfänger für Aufrufe der `SampleEvent`-Methode durch den Server zu dienen. Der Code dieser Klasse sieht folgendermaßen aus:

```
Option Explicit

    ' Dieses Objekt
    Public Name As String

    ' Beispiel-Ereignis wird vom Server aufgerufen
    Public Sub SampleEvent(ByVal msg As String)
       MsgBox Name & " " & msg
    End Sub
```

Die Einrichtung der OLE-Rückrufe wird im Formular `frmCallback` gezeigt, dessen Code Sie in Listing 11.4 sehen. Das Formular enthält die zwei Schaltflächen `cmdServer1` und `cmdServer2`.

Es werden zwei `clsCallback`-Server-Objekte definiert, genannt `server1` und `server2`. Zu beiden gibt es ein entsprechendes `clsCallbackSink`-Objekt, das jeweils den OLE-Rückruf entgegennimmt. Anstelle der Referenz auf das Formular werden nun die Referenzen auf diese `clsCallbackSink`-Objekte an die `SetCallback`-Funktion der Server übergeben.

Die zwei Schaltflächen simulieren das Auftreten eines Ereignisses. Wenn Sie eine davon anklicken, wird schließlich die `SampleEvent`-Methode in einem der `clsCallbackSink`-Objekte aufgerufen. Auf diese Weise kann Ihr Projekt zwischen beliebig vielen rückrufenden Server-Objekten unterscheiden.

```
    ' Guide to the Perplexed: Callback1 Sample
    ' Copyright (c) 1997 by Desaware Inc.  All Rights Reserved

    Option Explicit

    ' Dimensionierung von zwei separaten Objekten
    Dim obj1 As New clsCallbackSink
    Dim obj2 As New clsCallbackSink

    ' Hier sind die Server-Objekte
    Dim server1 As New clsCallback
    Dim server2 As New clsCallback

    Private Sub cmdServer1_Click()
       server1.EventDemo
    End Sub

    Private Sub cmdServer2_Click()
       server2.EventDemo
    End Sub

    Private Sub Form_Load()
       obj1.Name = "Object 1"
```

```
    obj2.Name = "Object 2"
    ' Jedem Server-Objekt wird ein eigener Client
    ' übergeben
    ' object reference
    server1.SetCallback obj1
    server2.SetCallback obj2
End Sub
```

Listing 11.4: Code des Formulars frmCallback

Dieser Ansatz ähnelt sehr dem vorangegangenen mit OLE-Ereignissen; er macht nur ein wenig mehr Arbeit.

11.3 OLE-Rückrufe und OLE-Ereignisse kombiniert

Um OLE-Ereignisse eines Objekts verwenden zu können, müssen Sie folgendendermaßen vorgehen:

- Dimensionieren des Objekts mit `WithEvents`:

  ```
  Dim WithEvents MeinObjekt As ObjektTyp
  ```

- Das Objekt anlegen:

  ```
  Set MeinObjekt = New ObjektTyp '(oder CreateObject verwenden)
  ```

- Die Ereignis-Prozedur anlegen:

  ```
  Sub MeinObjekt_MeinEreignis(ParameterListe)
  ```

Dies muß für jedes Objekt geschehen, dessen Ereignisse empfangen werden sollen.

Dieser Ansatz erlaubt natürlich nicht die dynamische Verwendung von Objekten mit Ereignissen. Wenn Sie Rückrufe von dynamisch angelegten Objekten erhalten wollen, müssen Sie OLE-Rückrufe verwenden, können aber die Rückrufe in einer separaten Klasse verbergen, um die Verwendung zu vereinfachen. Dabei können Sie auch den Overhead des Anlegens von separaten Klassen und Methoden für jedes Server-Objekt umgehen.

Die Projektgruppe `Callbk2.vbg` enthält zwei Projekte, die dies demonstrieren. Das Server-Projekt `Callback2server` enthält nun zwei Klassen. Die eine Klasse ist nahezu identisch mit der Klasse von `Callbackserver`, außer daß eine Eigenschaft für den Objekt-Namen hinzugekommen und sie zur Zusammenarbeit mit der Klasse `clsCallback2col`, der zweiten, neuen Klasse in diesem Projekt, vorbereitet ist. Beide Klassen sehen Sie in Listing 11.5.

```
' Guide to the Perplexed: Callback2 Sample
' Copyright (c) 1997 by Desaware Inc.  All Rights Reserved
```

```
Option Explicit

Public Name$

Private callbackobject As clsCallback2Col

' Beispiel-Funktion, die den Rückruf setzt
Public Sub SetCallback(myobject As Object)
   Set callbackobject = myobject
End Sub

' Beispiel-Funktion, die den Rückruf vornimmt
Public Sub EventDemo()
   callbackobject.SampleEvent Name$
End Sub

' Guide to the Perplexed: Callback2 Sample
' Copyright (c) 1997 by Desaware Inc.  All Rights Reserved
' Class clsCallback2Col

Option Explicit

' Dies ist die Definition des OLE-Ereignisses
' des Ereignis-Kombinations-Objekts.
Event SampleEvent(ByVal objectname$)

' Dies ist die OLE-Rückruf-Methode des Hauptinterfaces
' des Ereignis-Kombinations-Objekts
Sub SampleEvent(ByVal objectname$)
   RaiseEvent SampleEvent(objectname)
End Sub

' Eine Hilfsfunktion zur Initialisierung des
' OLE-Rückrufs
Public Sub SetCallbackServer(obj As clsCallback2)
   obj.SetCallback Me
End Sub
```

Listing 11.5: Die Klassen-Module clsCallback2 und clsCallback2Col

Die Klasse `clsCallback2col` dient gewissermaßen als Puffer für mehrere `clsCallback2`-Objekte. Die Methode `SetCallbackServer` dieser Klasse setzt den OLE-Rückruf für das Objekt zur Referenzierung des `clsCallback2col`-Objekts. Dieses Objekt enthält die Methode `SampleEvent`, die vom OLE-Rückruf im `clsCallback2`-Objekt aufgerufen wird. Das Objekt `clsCallback2col` löst dann das Ereignis im Client-Formular aus.

Kann ein Objekt ein Ereignis und eine Eigenschaft gleichen Namens haben? Ja, natürlich. Denken Sie daran, daß das Ereignis zu einer Methode eines Interfaces wird, das dem Client-Objekt gehört. Es existiert niemals im Server-Objekt, das das Ereignis auslöst. Daher ergibt sich auch kein Namenskonflikt.

Das Ergebnis dieses Ansatzes sehen Sie im Code des Formulars. Er unterscheidet sich stark von dem des Formulars `frmCallback` des vorangegangenen Beispiels, wie Sie in Listing 11.6 sehen können.

```
' Guide to the Perplexed: Callback2 Sample
' Copyright (c) 1997 by Desaware Inc.  All Rights Reserved

Option Explicit

Dim WithEvents multievent As clsCallback2Col

' Dies sind die Server-Objekte
Dim server1 As New clsCallback2
Dim server2 As New clsCallback2
Dim server3 As New clsCallback2

Private Sub cmdServer1_Click()
   server1.EventDemo
End Sub

Private Sub cmdServer2_Click()
   server2.EventDemo
End Sub

Private Sub cmdServer3_Click()
   server3.EventDemo
End Sub

Private Sub Form_Load()
   Set multievent = New clsCallback2Col
   server1.Name = "Object 1"
   server2.Name = "Object 2"
   server3.Name = "Object 3"

   multievent.SetCallbackServer server1
   multievent.SetCallbackServer server2
   multievent.SetCallbackServer server3

End Sub

' Alle Server-Objekte lösen letztlich dieses Ereignis
' aus
```

```
Private Sub multievent_SampleEvent(ByVal objectname _
 As String)
   MsgBox "Event received from " & objectname
End Sub
```

Listing 11.6: Code des Formulars frmCallback2

Schauen Sie sich die Prozeduren, die die Server-Objekte anlegen, näher an. Die Objekte werden dimensioniert, jedoch ohne das Schlüsselwort `WithEvents`. Dann wird die Eigenschaft `Name` gesetzt und das Objekt als Parameter an die Methode `SetCallbackServer` des `MultiEvent`-Objekts übergeben. Dieses bereitet die Server-Objekte auf den OLE-Rückruf der Methode `SampleEvent` des `MultiEvent`-Objekts vor. Sehen Sie einen Grund dafür, daß Sie diese Technik nicht auch für ein Array von `clsCallback2`-Server-Objekten verwenden könnten? Oder nicht mit Objekten verwenden könnten, die je nach Bedarf angelegt und wieder freigegeben werden? Ich hoffe nicht – es gibt nämlich keinen.

An dieser Stelle sind alle Server-Ereignisse (über OLE-Rückrufe ausgelöst) einem einzigen OLE-Ereignis im Client-Formular zugeordnet. Der Overhead von separaten Ereignis-Funktionen für jedes Objekt oder für dazwischengeschobene Objekte hat sich erübrigt.

11.4 Control-Ereignisse

Wenn Sie ein Control zur Design-Zeit in ein Formular einfügen, werden die Ereignisse des Controls bereits automatisch von dem Formular unterstützt. In Visual Basic 6 hat Microsoft die Möglichkeit hinzugefügt, Controls dynamisch zur Laufzeit anlegen zu können. Dieses Feature kann sehr hilfreich sein, wenn Sie beispielsweise Formulare zur Laufzeit konfigurieren möchten. Es wirft aber auch ein paar interessante Fragen im Zusammenhang mit der Ereignis-Behandlung auf. Da das Control zur Design-Zeit noch nicht existiert, gibt es auch keinen Mechanismus dafür, auf gewohnte Weise Code für die Ereignisse einzufügen. Tatsächlich bietet Visual Basic als einzige Möglichkeit der Unterstützung von Ereignissen von Objekten, die erst zur Laufzeit angelegt werden, die Deklaration mit `WithEvents` an. Wie bereits in diesem Kapitel erwähnt, muß jedoch bei der Verwendung des Schlüsselworts `WithEvents` der Objekt-Typ angegeben werden. Eine allgemeingültige Deklaration, etwa `As Object`, ist hier nicht möglich. Da Visual Basic die Interfaces der eingebauten Control-Typen kennt, können Sie folgenden Code zum dynamischen Anlegen eines Controls zur Laufzeit verwenden (wie auch im Beispiel-Programm `CtlEvent.vbp` zu sehen):

```
   Dim WithEvents cmdAddControl As CommandButton
'...
   Set cmdAddControl = _
    Me.Controls.Add("Vb.CommandButton", "cmdAddControl")
   Call cmdAddControl.Move(180, 1080, 1005, 465)
   cmdAddControl.Caption = "Add Control"
```

```
   cmdAddControl.Visible = True
'...

   Private Sub cmdAddControl_Click()
      If CtlTestExtender Is Nothing Then
         Set CtlTestExtender = _
          Me.Controls.Add("CtlEvent.CtlTest", "CtlTest1")
         CtlTestExtender.Move 1620, 540
         CtlTestExtender.Visible = True
      Else
         MsgBox "Control is already present"
      End If
   End Sub
```

Die Ereignis-Prozedur `cmdAddControl_Click` erscheint wie gewohnt, als wenn das Control zur Design-Zeit angelegt worden wäre. Etwas anders stellt sich die Situation dar, wenn nicht in Visual Basic eingebaute Controls zur Laufzeit hinzugefügt werden sollen. Visual Basic kann im voraus nicht wissen, wie das Ereignis-Interface des Controls aussehen wird. Auf den ersten Blick mag es erscheinen, als ob man das Schlüsselwort `WithEvents` nicht verwenden könnte. Das stimmt im Prinzip auch. Visual Basic umgeht diese Beschränkung jedoch mit einem eleganten Trick.

Anstelle ein Objekt des betreffenden Control-Typs hinzuzufügen, wird ein Objekt des Typs `VBControlExtender` hinzugefügt. Sie werden später noch erfahren, was es mit Extender-Objekten bezüglich Controls auf sich hat. Fürs erste sollte die Information reichen, daß es sich um ein zwischen Control und Client-Formular geschobenes Objekt handelt. Dieses Objekt kann direkt einige Standard-Ereignisse wie `GotFocus` oder `LostFocus` offenlegen. Es kann auch einige Standard-Methoden wie `Left`, `Top` oder `Name` offenlegen. Wenn Sie Ihr Control über die Methode `Controls.Add` anlegen, weisen Sie das Control-Objekt einer Variablen des Typs `VBControlExtender` zu, die so deklariert wird:

```
Dim WithEvents CtlTestExtender As VBControlExtender
```

Wie vermittelt nun dieses Objekt die Ereignisse, die von dem dahinterstehenden Control ausgelöst werden? Das Objekt verfügt selbst über das Ereignis `ObjectEvent`. Dieses Ereignis übergibt als Parameter das Objekt `EventInfo`. Dieses bietet Eigenschaften, die den Namen des Ereignisses (wie vom Control definiert) und dessen Parameter (in einer `Variant`-Collection) enthalten. Die Ereignis-Prozedur `CtlTestExtender_ObjectEvent` des Projekts `CtlEvent` zeigt, wie Ereignisse eines einfachen Controls mit zwei Ereignissen verarbeitet werden. Das `Click`-Ereignis hat keine Parameter. Das `DblClick`-Ereignis übergibt als Parameter die X- und Y-Position des Mauszeigers. Den Code sehen Sie hier:

```
 Private Sub CtlTestExtender_ObjectEvent(Info As EventInfo)
   Select Case Info.Name
      Case "Click"
```

```
         MsgBox "Control was clicked"
      Case "DblClick"
         MsgBox "Control was double clicked at " _
          & Info.EventParameters(0) & " , " _
          & Info.EventParameters(1)
   End Select
End Sub
```

Das Control `CtlTest` in dem Projekt ist einfach gehalten. Es gibt lediglich Ereignisse des `UserControl`-Objekts und eines Labels an den Container weiter. Das Interessante daran ist, daß hier gezeigt wird, wie Sie sowohl das `Click`- als auch das `DblClick`-Ereignis auswerten können – das ist etwas trickreich, da das `Click`-Ereignis in der Regel vor dem `DblClick`-Ereignis ausgelöst wird.

Dieser Ansatz befriedigt alle Anforderungen an das Einfügen eines Controls zur Laufzeit. Mit `WithEvents` dimensionierte Objekte sind nach wie vor früh gebunden. Das `VBControlExtender`-Objekt leitet alle Ereignisse eines Controls über das Ereignis `ObjectEvent` in dessen vordefinierten `EventInfo`-Parameter um.

Wie Sie sich sicher vorstellen können, gibt es viele Möglichkeiten, Ereignisse zu behandeln, sowohl als OLE-Rückrufe, als OLE-Ereignisse als auch als Kombination aus beidem. Wir werden später noch weitere Möglichkeiten kennenlernen. Das wichtigste, was Sie sich unbedingt merken sollten, ist, daß Sie es immer nur mit COM-Interfaces zu tun haben.