Kapitel 6

Prozeduren starten

In diesem Kapitel:

Bislang wurden alle Makros direkt aus dem Visual Basic-Editor gestartet. Dies ist zu Testzwecken legitim, allerdings keine praktikable Lösung für den Anwender. Visio 2010 stellt mehrere Varianten bereit, wie Prozeduren gestartet werden können. Sie werden in diesem Kapitel besprochen.

Prozeduraufruf über Entwicklertools/Code/Makros

Der Befehl *Ansicht/Makros/Makros* (Tastenkombination Alt+F8) öffnet ein Dialogfeld, das sämtliche als *Public* gespeicherten Makros in einem Listenfeld anzeigt. Sie werden über die Schaltfläche *Ausführen* gestartet, wie Sie in Abbildung 6.1 sehen. Alternativ finden Sie diesen Befehl in der Registerkarte *Entwicklertools/Code*, wenn Sie die Registerkarte *Entwicklertools* über die Symbolleiste für den Schnellzugriff aktiviert haben.

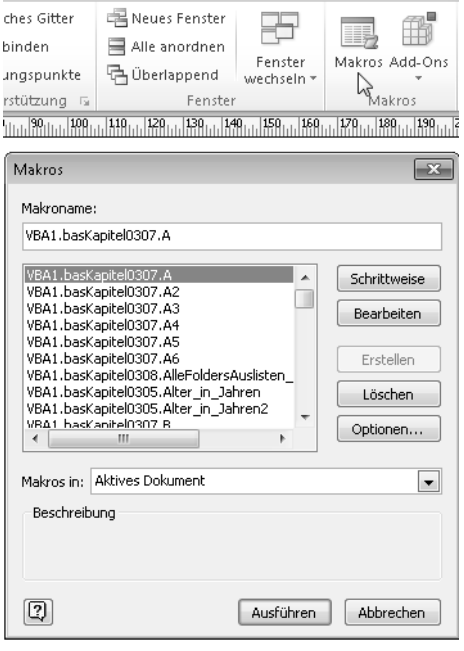

Abbildung 6.1 Die Makros des Kapitels 3

Makros per Doppelklick auf ein Shape starten

Wenn Sie auf ein Shape doppelklicken, wird normalerweise der Textmodus geöffnet. Dieses Doppelklickverhalten kann geändert werden. Markieren Sie das Shape und wählen Sie den Befehl *Entwicklertools/Shape-Design/Verhalten* und dort in der Registerkarte *Doppelklicken* das korrespondierende Makro, das gestartet werden soll, wenn der Benutzer auf das Shape doppelklickt. Das Shape kann per Drag & Drop in eine neue oder eine vorhandene Schablone gezogen werden, sodass diese Funktion für alle Shapes zur Verfügung steht, die aus diesem Master-Shape generiert werden. Ebenso kann das Doppelklickverhalten

bei zweidimensionalen und eindimensionalen Shapes, bei eingefügten Bildern, bei Führungslinien und Führungspunkten aktiviert werden. Obwohl Zeichenblätter auch Shapes sind, das heißt, auch sie besitzen ein ShapeSheet, verfügen sie jedoch über kein Doppelklickverhalten.

ACHTUNG Die Zeichnung muss bereits gespeichert sein, bevor Sie das Doppelklickverhalten aktivieren. Wird zuerst das Makro ausgewählt und anschließend die Zeichnung gespeichert, stimmt der vollständige Pfad nicht mehr, und das Makro kann nicht aufgerufen werden. Ebenso dürfen Modulname und Makroname nicht mehr geändert werden, da Visio diese Änderung nicht aktualisiert. Wenn Sie einen der drei Namen ändern – Dateiname, Modulname oder Makroname –, müssen Sie über den Befehl *Entwicklertools/Shape-Design/Verhalten* das Doppelklickverhalten, das ein Makro ausführt, neu einstellen. Sie sehen dieses Dialogfeld in Abbildung 6.2.

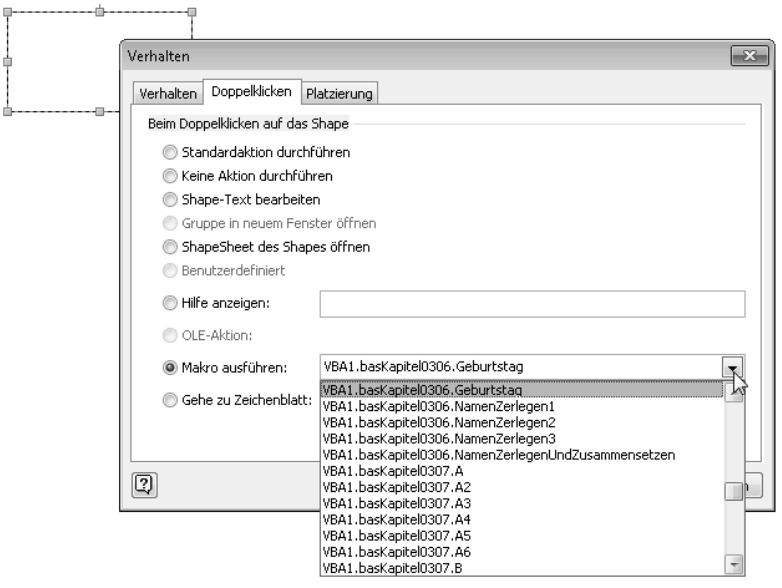

Abbildung 6.2 Das Doppelklickverhalten von Shapes anpassen

Diese Variante hat jedoch den Nachteil, dass der Benutzer nicht sieht, dass er ein Makro starten und wie er es aufrufen kann. Sie könnten dies im Shape mithilfe eines QuickInfos oder eines Kommentars mitteilen. Das Verändern des Doppelklickverhaltens hat einen weiteren Nachteil: Viele Benutzer kennen die Möglichkeit, dass Text über einen Doppelklick editiert werden kann. Ändert man nun diese Eigenschaft, kann dies leicht zu Verwirrung führen.

Der Abschnitt Events im ShapeSheet

Wird das Doppelklickverhalten geändert, erscheint dies in den ShapeSheets im Abschnitt *Events* in der Zelle *EventDblClick*. Dort steht beispielsweise die Formel:

```
=RUNADDON("Zeichnung1.Modul1.Test")
```

Sollen zwei Prozeduren hintereinander aufgerufen werden, können sie mit einem Pluszeichen verkettet werden:

```
=RUNADDON ("Zeichnung1.Modul1.Test")+RUNADDON("Zeichnung1.Modul1.Test2")
```

Funktionsname	Bedeutung
RUNADDON	Startet ein Makro
CALLTHIS	Startet ein Makro

Tabelle 6.1 Diese Funktionen starten ein Makro

Während die Funktion *RUNADDON* nur eine Prozedur starten kann, kann mit *CALLTHIS* eine Prozedur gestartet werden, an die ein oder mehrere Parameter übergeben wird. Die Syntax lautet:

```
CALLTHIS("Prozedur", ["Projekt"], [Arg1, Arg2, ...])
```

Im Abschnitt *Events* finden Sie einige weitere Ereignisse:

Zellname	Bedeutung
TheData	Nicht belegt
TheText	Dieses Ereignis wird aufgerufen, wenn Text eingefügt, geändert, gelöscht oder editiert wird
EventDblClick	Doppelklicken
EventXFMod	Dieses Ereignis wird aufgerufen, wenn Größe, Position oder Ausrichtung geändert wird
EventDrop	Das Shape wird aus der Schablone auf das Zeichenblatt gezogen
EventMultiDrop	Mehrere Shapes werden als Instanzen (beispielsweise als Gruppe) auf dem Zeichenblatt abgelegt, dupliziert oder eingefügt

Tabelle 6.2 Die Zellen im Abschnitt *Ereignisse*

Die Action-Zelle im ShapeSheet

Jedem Shape können neue Kontextmenüeinträge in einer Zeile im Abschnitt *Action* hinzugefügt werden. Mit ihrer Hilfe lassen sich andere Zellen des ShapeSheets steuern. Beispielsweise wird ein Text angezeigt oder ausgeblendet:

```
=SETF("HideText",NOT(HideText))
```

Die Funktion *DOCMD* ruft jedes interne Visio-Dialogfeld auf, beispielsweise das Dialogfeld *Drucken*:

```
=DOCMD(1010)
```

Wurde eine Prozedur mit dem Namen *Datenexport* erstellt, kann sie mithilfe des Kontextmenüs beziehungsweise mithilfe der Funktion *RUNADDON* aufgerufen werden:

```
=RUNADDON("Datenexport")
```

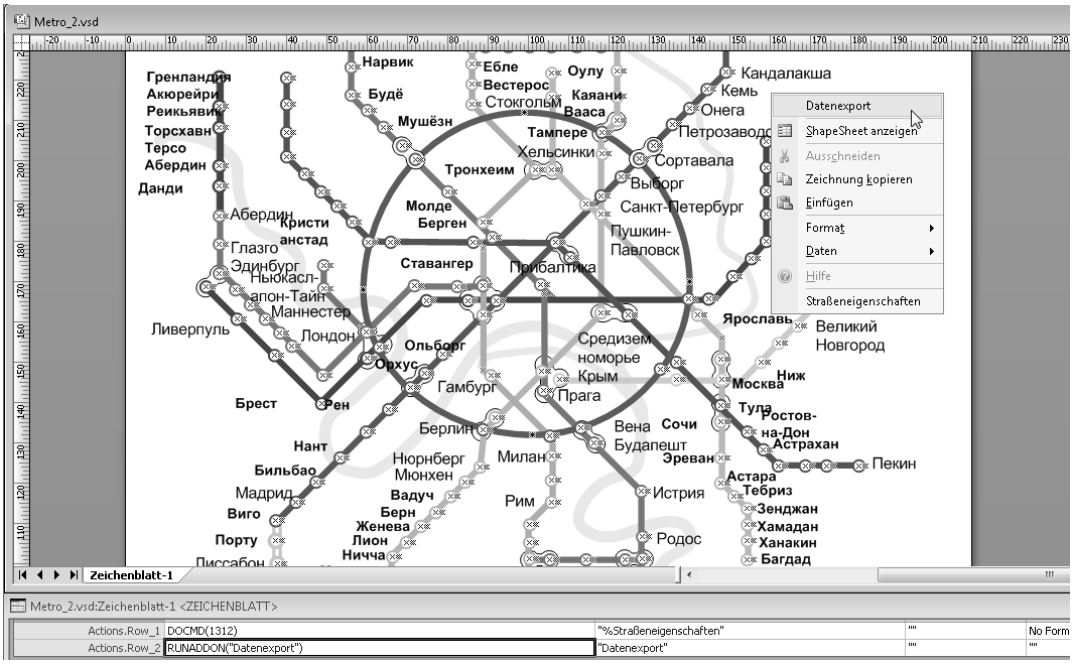

Abbildung 6.3 Über das Kontextmenü können Funktionen aufgerufen werden, die andere Zellen steuern, interne Dialogfelder anzeigen oder Prozeduren aufrufen

Steuerelemente

Man kann Makros auch über Befehlsschaltflächen starten, die sich auf einem Zeichenblatt befinden. Dazu muss die Registerkarte *Entwicklertools* aktiviert sein. Dort finden Sie in der Gruppe *Steuerelemente* den Befehl *Einfügen/Befehlsschaltfläche*. Ebenso finden sich dort weitere ActiveX-Steuerelemente (Abbildung 6.4).

Abbildung 6.4 Die ActiveX-Steuerelemente

Wird die Befehlsschaltfläche angeklickt, erscheint sie in der Mitte des Zeichenblatts. Selbstverständlich kann sie verschoben werden. Im Kontextmenü finden Sie den Befehl *Befehlsschaltfläche-Objekt/Steuerelementeigenschaften* (Abbildung 6.5). Er öffnet eine Liste der verschiedenen Eigenschaften, die für das Steuerelement eingestellt werden können – wahlweise alphabetisch oder nach Kategorien sortiert.

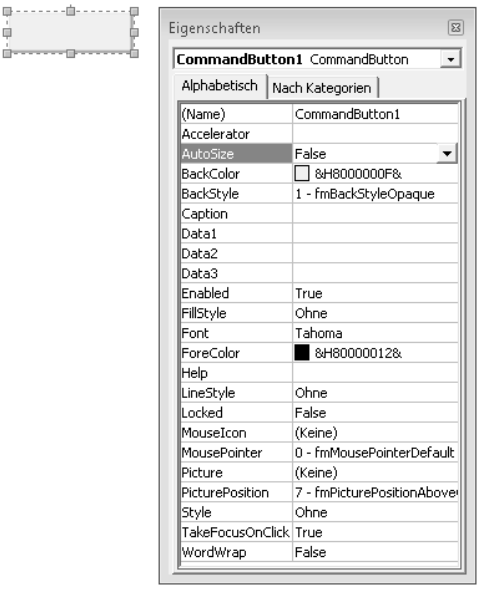

Abbildung 6.5 Die Liste der Eigenschaften

Die wichtigste Eigenschaft ist der *Name*. Dort wird der interne Name vergeben.

HINWEIS Verwechseln Sie nicht *Name* und *Caption*. *Caption* ist lediglich die Beschriftung der Befehlsschaltfläche; *Name* ist der interne Name der Befehlsschaltfläche, den der Benutzer normalerweise nicht zu Gesicht bekommt.

Sollte die Schaltfläche nicht aktiviert sein, genügt es, sie anzuklicken. Ein weiterer Klick führt dazu, dass sich der Cursor mitten in der Beschriftung befindet, wo man die Beschriftung löschen und verändern kann.

Im Kontextmenü der Befehlsschaltfläche finden Sie den Befehl *Befehlsschaltfläche-Objekt/Code* anzeigen. Wird dieser Befehl gewählt, wird Visual Basic geöffnet. Der Cursor befindet sich dort im Click-Ereignis des ActiveX-Objekts. Dort wird der Code eingegeben. Damit dieses Ereignis funktioniert, müssen Sie den Entwurfsmodus über den entsprechenden Befehl in *Entwicklertools/Steuerelemente* deaktivieren.

Beispiel: Ein Zeichenblatt besitzt ein Datenfeld *TotalArea*. Die Zahl, die sich darin befindet, wird auf 0 gesetzt, wenn eine Befehlsschaltfläche angeklickt wird. Der Name der Befehlsschaltfläche lautet *cmdReset*. Der folgende Code löst die Aufgabe:

```
Private Sub cmdZuruecksetzen_Click()
   ActivePage.PageSheet.Cells("Prop.TotalArea").ResultIU = 0
End Sub
```

Listing 6.1 Die angezeigte Größe der Fläche wird zurückgesetzt

Das Shape zeigt diese Zahl als Feld an:

```
=ThePage!Prop.TotalArea
```

Wird nun ein weiteres Shape auf das Blatt gezogen, berechnet eine Prozedur beim Erzeugen eines neuen Shapes die Gesamtfläche des neuen Shapes und addiert sie zur alten Zahl der benutzerdefinierten Zelle des Zeichenblatts hinzu (Abbildung 6.6):

```
Private Sub Document_ShapeAdded(ByVal Shape As IVShape)
   If Shape.Type = visTypeShape Then
      With ActivePage.PageSheet.Cells("Prop.TotalArea")
         .ResultIU = .ResultIU + Shape.AreaIU
      End With
   End If
End Sub
```

Listing 6.2 Jedes neu eingefügte Shape bewirkt ein Heraufzählen der Größe

Beim Löschen wird die Fläche des Shapes wieder abgezogen:

```
Private Sub Document_BeforeSelectionDelete(ByVal Selection As IVSelection)
   Dim i As Integer

   For i = 1 To Selection.Count
      If Selection(i).Type = visTypeShape Then
         With ActivePage.PageSheet.Cells("Prop.TotalArea")
            .ResultIU = .ResultIU - Selection(i).AreaIU
         End With
      End If
   Next
End Sub
```

Listing 6.3 Umgekehrt muss beim Löschen eines Shapes die Größenangabe verringert werden

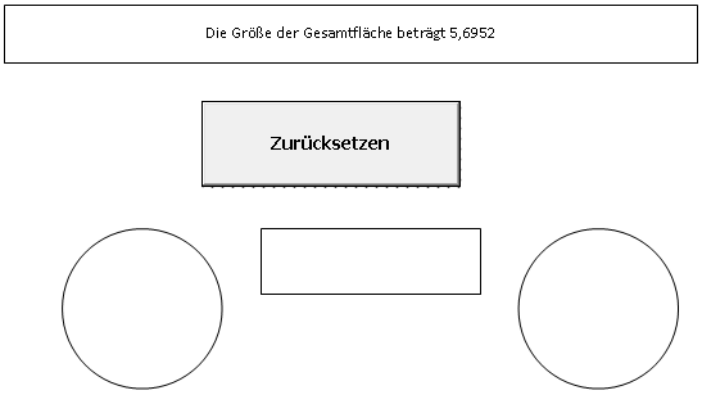

Abbildung 6.6 Die Schaltfläche setzt die angezeigte Zahl zurück.

Visio-Ereignisse

Häufig starten die Makros, wenn ein spezifisches Ereignis ausgelöst wird. Dies kann an die Zelle gebunden sein. Wie im vorherigen Abschnitt beschrieben, stehen im Abschnitt *Events* sechs (genauer: fünf) Ereignisse zur Verfügung: *TheData* (nicht belegt), *TheText*, *EventDblClick*, *EventXFMod*, *EventDrop* und *EventMultiDrop*.

Diese fünf Ereignisse sind nur an ein Shape gebunden. Im Visual Basic-Editor findet sich im Projekt-Explorer das Visio-Objekt *ThisDocument*. Dieses verfügt über 44 Ereignisse, die Sie im Projekt-Explorer einsehen können:

- *AfterRemoveHiddenInformation*
- *BeforeDataRecordsetDelete*
- *BeforeDocumentClose*
- *BeforeDocumentSave*
- …
- *UngroupCanceled*

Diese Ereignisse werden normalerweise an Zeichnungen oder an Vorlagen gebunden. Damit kann auf bestimmte Ereignisse reagiert werden, die vom Benutzer ausgelöst werden. Die Wichtigsten werden im Folgenden an einigen Beispielen beschrieben.

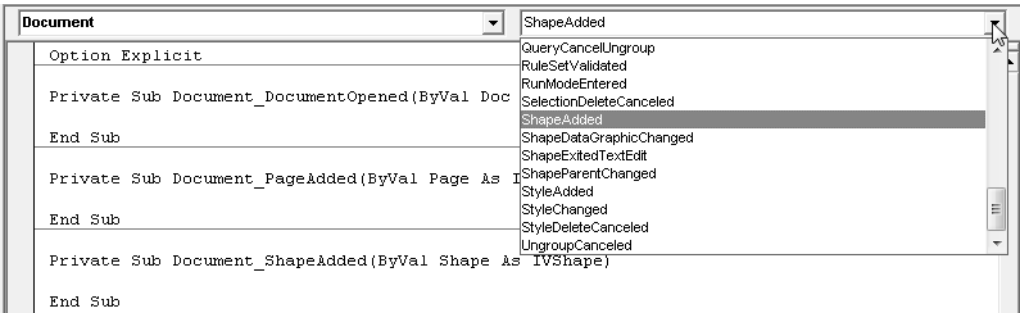

Abbildung 6.7 Die Visio-Ereignisse

Beispiel: Eine große Zeichnung, die regelmäßig von einem Benutzer bearbeitet wird, soll beim Öffnen auf einem Server gespeichert werden. Dazu wird das Ereignis *DocumentOpened* mit folgendem Code aufgerufen:

```
Private Sub Document_DocumentOpened(ByVal doc As IVDocument)
   Dim strDateiname As String
   Dim strDateinameMitPfad As String
   On Error Resume Next
   strDateiname = ActiveDocument.Name
```

Listing 6.4 Das Dokument wird beim Öffnen als Kopie an eine andere Stelle gespeichert

```
   strDateinameMitPfad = ActiveDocument.FullName
   Application.ActiveDocument.SaveAs "U:\Sicherung\" & strDateiname
   Application.ActiveDocument.SaveAs strDateinameMitPfad
End Sub
```

Listing 6.4 Das Dokument wird beim Öffnen als Kopie an eine andere Stelle gespeichert *(Fortsetzung)*

Die Zeile

```
On Error Resume Next
```

ist zwar nicht elegant, aber sehr simpel und effektiv: Besteht kein Netzwerkzugriff oder kann aus irgendeinem Grund nicht gespeichert werden, wird die Zeile

```
   Application.ActiveDocument.SaveAs "U:\Sicherung\" & strDateiname
```

schlicht übergangen. Das Ereignis *Opened* erhält den Parameter *doc*. Dies ist das Objekt, das einen Verweis auf das Dokument herstellt, das gerade geöffnet wird. Statt *ActiveDocument* kann mit dem Objekt *doc* gearbeitet werden:

```
Private Sub Document_DocumentOpened(ByVal doc As IVDocument)
   Dim strDateiname As String
   Dim strDateinameMitPfad As String
   On Error Resume Next
   strDateiname = doc.Name
   strDateinameMitPfad = doc.FullName
   doc.SaveAs "U:\Sicherung\" & strDateiname
   doc.SaveAs strDateinameMitPfad
End Sub
```

Listing 6.5 *ActiveDocument* oder der Parameter *Doc*

Diese Variante ist dann interessant, wenn mit mehreren Dokumenten gearbeitet wird. Dann muss *ActiveDocument* nicht zwangsläufig dasjenige sein, das vom Benutzer geöffnet wurde.

Dem Befehl *Datei/Neu* entspricht das Ereignis *DocumentCreated*. Das Schließen wird mit dem Ereignis *BeforeDocumentClose* abgefangen. Bricht der Benutzer das Schließen ab, kann mit dem Ereignis *DocumentCloseCanceled* darauf reagiert werden.

Speichern und *Speichern unter* werden mit *BeforeDocumentSave* und *BeforeDocumentSaveAs* abgefangen, bevor gespeichert wurde, und mit *DocumentSaved* und *DocumentSavedAs*, nachdem die Datei gespeichert wurde.

Arbeitet ein Benutzer regelmäßig mit einer bestimmten Vorlage, aus der er Zeichnungen erstellt, vergisst sie allerdings häufig zu speichern, könnten Sie ihm diese Aufgabe über das Ereignis *BeforeDocumentClose* abnehmen:

```
Private Sub Document_BeforeDocumentClose(ByVal doc As IVDocument)
   Dim strDateiname As String
   Dim i As Integer
   i = 1
   strDateiname = "C:\Eigene Dateien\Test" & _
      Format(Day(Date), "00") & _
      Format(Month(Date), "00") & _
      Format(Year(Date), "0000") & i & ".vsd"
   If doc.Saved = False Then
      Do Until Dir(strDateiname) = ""
         i = i + 1
         strDateiname = "C:\Eigene Dateien\Test" & _
            Format(Day(Date), "00") & _
            Format(Month(Date), "00") & _
            Format(Year(Date), "0000") & i & ".vsd"
      Loop
      doc.SaveAs strDateiname
   End If
End Sub
```

Listing 6.6 Jedes Dokument muss gespeichert werden

In diesem Beispiel wird überprüft, ob ein Dateiname, der sich aus *Test*, dem Datum in der Form *01042011* einer *Versionsnummer* zusammensetzt, beispielsweise 1 und der Endung *.vsd*, bereits im Ordner *C:\Eigene Dateien* existiert. Falls ja, wird die Nummer um 1 erhöht. Falls nein, wird die Datei unter diesem Namen gespeichert. Vorausgesetzt, der Benutzer hat sie nicht selbst gespeichert (*doc.Saved = False*).

Analog dem Objektmodell stehen auch auf der Ebene der Zeichenblätter Ereignisse zur Verfügung: *PageAdded*, *PageChanged*, *PageDeleteCanceled* und *BeforePageDelete*:

```
Private Sub Document_PageAdded(ByVal Page As IVPage)
   MsgBox "Ich begrüße Sie auf dem neuen Blatt """ & Page.Name & """"
End Sub

Private Sub Document_BeforePageDelete(ByVal Page As IVPage)
   MsgBox "Nun wird das Blatt """ & Page.Name & """ gelöscht."
End Sub
```

Listing 6.7 Ein Shape wird beim Einfügen gelöscht

Diese Ereignisse können interessant sein, wenn der aktuelle Stand der Zeichenblätter in einer Datenbank gespeichert werden soll. Beim Erstellen einer neuen Datei wird für diese Datei eine neue Datenbank angelegt. Für jedes neue Zeichenblatt wird eine Tabelle generiert; wird ein Zeichenblatt gelöscht, wird auch die Tabelle gelöscht.

Für Zeichenblätter kann es interessant sein zu überprüfen, ob ein neues Master-Shape auf das Zeichenblatt gezogen wird, das sich noch nicht auf dem Blatt befindet. Dies wird vom Ereignis *MasterAdded* abgefangen. Analog stehen die Ereignisse *BeforeMasterDelete*, *MasterChanged* und *MasterDeleteCanceled* zur Verfügung. Besonders interessant ist es, wenn diese Ereignisse an eine Schablone gebunden werden. Wird dort ein neues Master-Shape erzeugt, ein vorhandenes geändert oder gelöscht, kann mit *MasterAdded*, *BeforeMasterDelete* und *MasterChanged* darauf reagiert werden. Jedes dieser vier Ereignisse übergibt das Objekt *Master*.

Auch Formatvorlagen werden als Objekte behandelt. Analog zu den Mastern werden folgende Ereignisse bereitgestellt: *BeforeStyleDelete*, *StyleAdded*, *StyleChanged* und *StyleDeleteCanceled*. Dabei kann mit dem Objekt *Styles* gearbeitet werden.

Interessanter sind die Ereignisse, mit denen auf Benutzeraktionen, die Shapes betreffen, reagiert werden kann. Mit dem Ereignis *ShapeAdded* kann sowohl abgefangen werden, ob der Benutzer ein neues Rechteck, eine Linie, Kurve oder Ellipse zeichnet oder ob er ein Master-Shape aus der Schablone auf das Zeichenblatt zieht. Dieses Ereignis reagiert auch auf das Generieren von neuen Hilfslinien und Grafiken, da diese schließlich auch Shapes sind. Das Objekt, das hierbei übergeben wird, heißt *Shape*.

Ein kleiner Scherz: Es wird überprüft, ob der Anwender eine *Cafeteria* auf das Zeichenblatt zieht. Falls ja, erhält er einen Hinweis, und die Cafeteria wird gelöscht:

```
Private Sub Document_ShapeAdded(ByVal Shape As IVShape)
   If InStr(1, Shape.Name, "Cafeteria") > 0 Then
      MsgBox "Bitte keine Pause!"
      Shape.Delete
   End If
End Sub
```

Oder noch besser: Die Cafeteria wird durch das Shape *Überwachung* ersetzt, wie Abbildung 6.8 zeigt:

```
Private Sub Document_ShapeAdded(ByVal Shape As IVShape)
   Dim vsSchablone As Document
   Dim vsMastTV As Master
   Dim dblX As Double
   Dim dblY As Double

   If InStr(1, Shape.Name, "Cafeteria") > 0 Then
      Set vsSchablone = Application.Documents("WFDEP_M.vss")
      Set vsMastTV = schabSchablone.Masters("Überwachung")
      dblX = Shape.Cells("DrehBezX").Result(visInches)
      dblY = Shape.Cells("DrehBezY").Result(visInches)
      Shape.Delete
      ActivePage.Drop vsMastTV, dblX, dblY
   End If
End Sub
```

Listing 6.8 Ein Shape ersetzt ein anderes

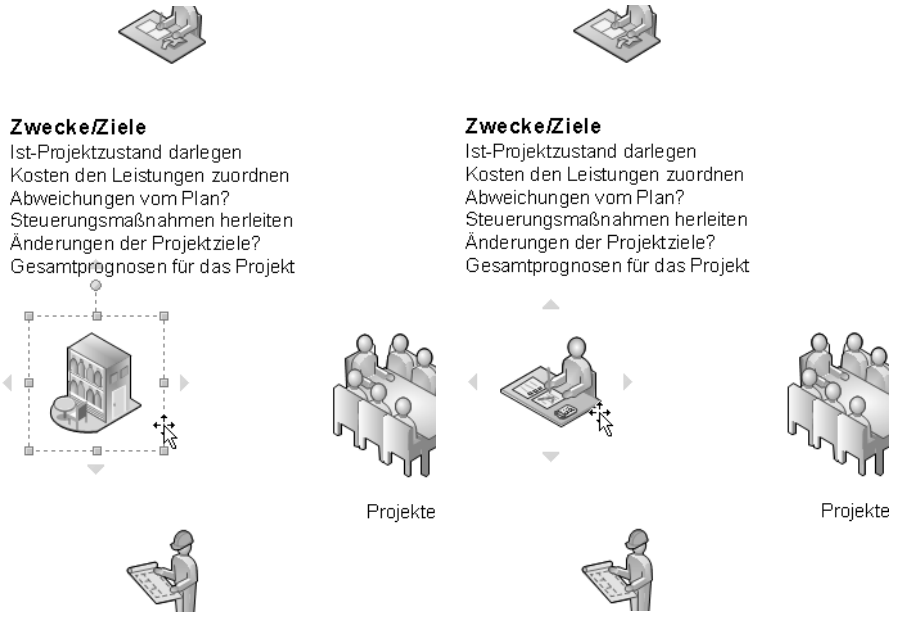

Abbildung 6.8 Überwachung statt Cafeteria!

Es gibt eine Reihe von Anwendungen, in denen dieses Ereignis sinnvoll eingesetzt werden kann. Zieht der Benutzer ein bestimmtes Master-Shape aus der Schablone und lässt es auf einem anderen Shape fallen, welches das gleiche Master-Shape als Grundlage hat, erhält er die Frage gestellt, ob das alte durch das neue ersetzt werden soll. Bejaht er diese Frage, wird das neue Shape auf die gleiche Position wie das alte platziert, übernimmt eine benutzerdefinierte Eigenschaft, klebt alte Verbinder an sich und löscht das alte Shape. Es handelt sich dabei nur um das Shape mit Namen *Auslöser*. Das Datenfeld lautet *Kosten*; es wird ebenso wie der Master-Shape-Name in einer Konstanten abgelegt. Zuerst werden alle Shapes des Zeichenblatts durchlaufen und überprüft, ob ihre Entfernung zu dem neuen Shape geringer ist als die Breite des Master-Shapes. Wird ein solches gefunden, erfolgt eine Meldung. Fällt die Antwort positiv aus, werden die alten Formeln, das heißt Werte, aus den Zellen des Abschnitts *Shape Data*: *Kosten*, *X-Position* und *Y-Position* vom alten Shape auf das neue übertragen.

```
Const SHAPENAME = "Auslöser"
Const DATENFELDNAME = "Kosten"
Private Sub Document_ShapeAdded(ByVal Shape As IVShape)
Dim shp As Shape
Dim shpVerbinder As Shape
On Error Resume Next
   If InStr(1, Shape.Name, SHAPENAME) > 0 Then
      For Each shp In ActivePage.Shapes
         If shp.DistanceFrom(Shape, 1) < _
         Shape.Cells("Width").Result("in") And _
         shp <> Shape _
         And InStr(1, shp.Name, SHAPENAME) > 0 Then
```

Listing 6.9 Ein Shape ersetzt ein anderes und übernimmt dessen Informationen

```
            If MsgBox("Soll das alte Shape durch dieses " _
            & "ersetzt werden?", vbYesNo) = vbYes Then

               Shape.Cells("Prop.Kosten").Formula = _
                  shp.Cells("Prop.Kosten").Formula
               Shape.Cells("PinX").Formula = shp.Cells("PinX").Formula
               Shape.Cells("PinY").Formula = shp.Cells("PinY").Formula

            Shape.Cells("Prop." & DATENFELDNAME).Formula = _
               shp.Cells("Prop." & DATENFELDNAME).Formula

               Call Neu_Kleben(shp, Shape)
               shp.Delete
               Exit Sub
            End If
         End If
      Next
   End If
End Sub
```

Listing 6.9 Ein Shape ersetzt ein anderes und übernimmt dessen Informationen *(Fortsetzung)*

Für das neue Kleben des Verbinders wurde eine zweite Prozedur geschrieben: *Neu_Kleben*. Darin wird für jedes Shape des Zeichenblatts überprüft, ob es sich um einen Verbinder handelt. Falls ja, werden die vier Koordinaten der Anfangs- und Endpunkte daraufhin untersucht, ob sie mit dem alten Shape kleben, das heißt, ob der Name des alten Shapes in einer der vier Zellen steht; beispielsweise so:

```
=PAR(PNT(Document!Connections.X2,Document!Connections.Y2))
```

Falls er als Zeichenkette darin zu finden ist, wird er mit der Funktion *Replace* durch den Namen des neuen Shapes ersetzt (Abbildung 6.9).

```
Sub Neu_Kleben(shpAltesShape As Shape, _
   shpNeuesShape As Shape)
   Dim shp As Shape
   On Error Resume Next

   For Each shp In ActivePage.Shapes
      If shp.CellExists("BeginX", True) = True Then
         If InStr(1, shp.Cells("BeginX").Formula, _
            shpAltesShape.Name) > 0 Then
            shp.Cells("BeginX").Formula = _
            Replace(shp.Cells("BeginX").Formula, _
            shpAltesShape.Name, shpNeuesShape.Name)
         End If

         If InStr(1, shp.Cells("BeginY").Formula, _
            shpAltesShape.Name) > 0 Then
            shp.Cells("BeginY").Formula = _
            Replace(shp.Cells("BeginY").Formula, _
            shpAltesShape.Name, shpNeuesShape.Name)
         End If
```

Listing 6.10 Das neue Shape wird an den Verbinder geklebt, an dem das alte Shape geklebt war

```
        If InStr(1, shp.Cells("EndX").Formula, _
           shpAltesShape.Name) > 0 Then
           shp.Cells("EndX").Formula = _
           Replace(shp.Cells("EndX").Formula, _
           shpAltesShape.Name, shpNeuesShape.Name)
        End If
        If InStr(1, shp.Cells("EndY").Formula, _
           shpAltesShape.Name) > 0 Then
           shp.Cells("EndY").Formula = _
           Replace(shp.Cells("EndY").Formula, _
           shpAltesShape.Name, shpNeuesShape.Name)
        End If
      End If
   Next
End Sub
```

Listing 6.10 Das neue Shape wird an den Verbinder geklebt, an dem das alte Shape geklebt war *(Fortsetzung)*

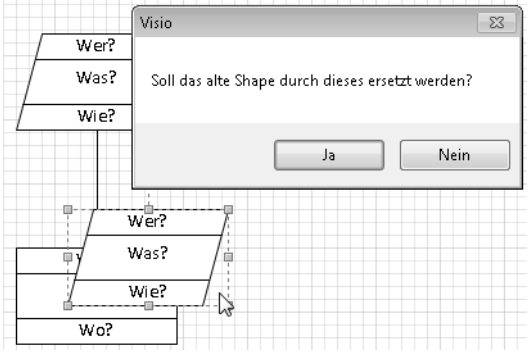

Abbildung 6.9 Ein neues Shape ersetzt ein altes

Das Ereignis, das ausgelöst wird, wenn ein Shape gelöscht wird, lautet: *BeforeSelectionDelete*. Im folgenden Beispiel wurde ein Shape in *Lösch-Shape* umbenannt (über den Befehl *Entwicklertools/Shape-Design/Shape-Name*). Nun werden die Namen aller gelöschten Shapes auf diesem Shape angezeigt:

```
Private Sub Document_BeforeSelectionDelete _
   (ByVal Selection As IVSelection)
   Dim shpShape As Shape
   Dim shpLöschshape As Shape
   Set shpLöschshape = ActivePage.Shapes("Lösch-Shape")
   For Each shpShape In Selection
      shpLöschshape.Text = shpLöschshape.Text & _
      vbLf & shpShape.Name
   Next
End Sub
```

Listing 6.11 Ein Shape protokolliert die Löschvorgänge

Im folgenden Beispiel wird dieses Ereignis verwendet. Es geht um folgendes Problem: Zieht der Benutzer aus einer Schablone ein bestimmtes Shape, wird er gefragt, wie viele Shapes dieser Sorte er insgesamt haben möchte, wie Sie in Abbildung 6.10 sehen können. Er kann eine Zahl von 1 bis 20

auswählen. Diese Shapes werden automatisch nebeneinander angeordnet. Das passiert auch, wenn er ein weiteres Shape auf das Zeichenblatt zieht, das zum gleichen Master-Shape gehört.

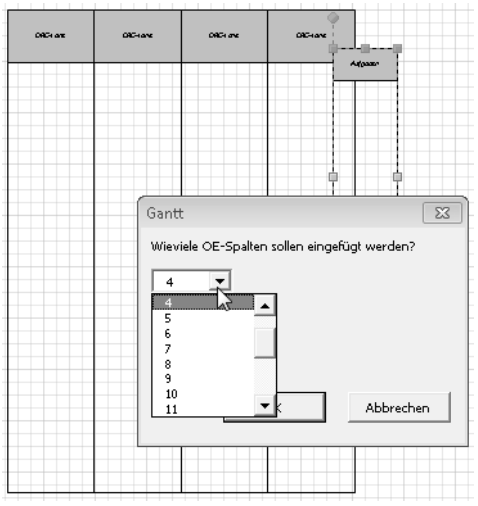

Abbildung 6.10 Es werden weitere Shapes hinzugefügt

Wenn nun der Benutzer ein Shape oder mehrere Shapes löscht, wie in Abbildung 6.11, werden alle Shapes, die sich rechts davon befinden, nach links versetzt, sodass alle Shapes wieder bündig in einer Reihe stehen.

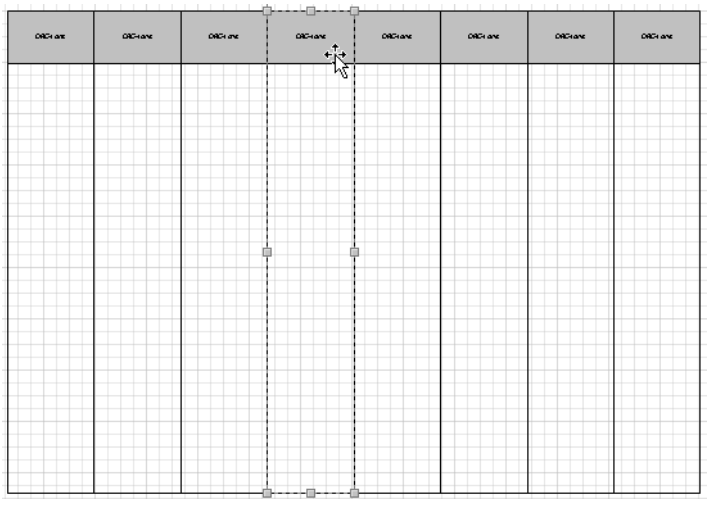

Abbildung 6.11 Shapes können gelöscht werden, ohne die Ordnung zu stören

Dazu sind einige Ereignisse nötig. Im Ereignis *ShapeAdded* wird überprüft, ob die Obergrenze von 20 schon erreicht ist. Falls nicht, wird das neue Shape positioniert. Die Anzahl der vorhandenen Shapes »merkt« sich das Programm, indem es den Wert in eine benutzerdefinierte Zelle (*LaneZahl*) des Zeichenblatts schreibt. Dieser Wert wird auch ausgelesen. Danach wird das Formular geöffnet, über das nach der Anzahl gefragt wird.

```
Const XPOS = 6
Const YPOS = 200
Private Sub Document_ShapeAdded(ByVal Shape As IVShape)
   Dim dblBreiteShp As Double
   Dim intLaneZahl As Integer
   Dim vsShape As Shape

   If Left(Shape.Name, 8) = "Org-Lane" Then
      intLaneZahl = _
         ActivePage.PageSheet.Cells("prop.LaneZahl"). _
         ResultInt(visNoCast, True)
      If intLaneZahl >= 20 Then
         MsgBox "Die Obergrenze ist erreicht!", _
            vbInformation
         Exit Sub
      End If

      dblBreiteShp = Shape.Cells("Width").Result(visMillimeters)

      ActivePage.PageSheet.Cells("prop.LaneZahl").Formula = _
      ActivePage.PageSheet.Cells("prop.LaneZahl").Formula + 1

      Shape.Cells("PinX").Result(visMillimeters) = _
      XPOS + intLaneZahl * dblBreiteShp
      Shape.Cells("PinY").Result(visMillimeters) = YPOS
      frmGantt.Show
   End If
End Sub
```

Listing 6.12 Benutzeraktionen werden im ShapeSheet des Zeichenblatts mitprotokolliert

Die UserForm ruft eine Prozedur auf, die erneut die Obergrenze überprüft, anschließend neue ORG-Lane-Shapes auf das Zeichenblatt zieht, korrekt positioniert und die Anzahl in den benutzerdefinierten Eigenschaften um 1 erhöht.

```
Sub ZeilenRein(intGewAnzahl As Integer)
   Dim vsApp As Application
   Dim vsSchablone As Document
   Dim vsMaster As Master
   Dim vsShape As Shape
   Dim dblBreiteShp As Double
   Dim intZähler As Integer
   Dim intLaneZahl As Integer
   intLaneZahl = ActivePage.PageSheet.Cells("Prop.LaneZahl").Formula
   If intLaneZahl >= 20 Then
      MsgBox "Die Obergrenze ist erreicht!", vbInformation
      Exit Sub
   ElseIf intLaneZahl + intGewAnzahl > 20 Then
      MsgBox "Es können leider nur noch " & _
         20 - intLaneZahl & " hinzugefügt werden."
   End If

   Set vsApp = Application
```

Listing 6.13 Die Obergrenze wird überprüft

```
   Set vsSchablone = vsApp.Documents(SCHABLONENNAME)

   If vsSchablone = Empty Then
      Set vsSchablone = _
         vsApp.Documents.OpenEx(SCHABLONENNAME, visOpenDocked)
   End If
   Set vsMaster = vsSchablone.Masters(LANENAME)

   For intZähler = 2 To intGewAnzahl
      Set vsShape = ActivePage.Drop(vsMaster, 1, 1)
      dblBreiteShp = vsShape.Cells("Width").Result(visMillimeters)

      intLaneZahl = _
      ActivePage.PageSheet.Cells("prop.LaneZahl").Formula
      vsShape.Cells("PinX").Result(visMillimeters) = _
         XPOS + intLaneZahl * dblBreiteShp
      vsShape.Cells("PinY").Result(visMillimeters) = YPOS
   ActivePage.PageSheet.Cells("Prop.LaneZahl").Formula = _
   ActivePage.PageSheet.Cells("Prop.LaneZahl").Formula + 1

   If ActivePage.PageSheet.Cells("Prop.LaneZahl").Formula >= 20 Then
         Exit Sub
      End If
   Next
End Sub
```

Listing 6.13 Die Obergrenze wird überprüft *(Fortsetzung)*

Übrigens sollten auch neue Zeichenblätter mit dieser benutzerdefinierten Eigenschaft *LaneZahl* versehen werden. Dazu wird das Ereignis *PageAdded* verwendet:

```
Private Sub Document_PageAdded(ByVal Page As IVPage)
      If ActivePage.PageSheet.SectionExists _
      (visSectionProp, True) = False Then
         ActivePage.PageSheet.AddSection visSectionProp
      End If

      If ActivePage.PageSheet.CellExists _
      ("prop.LaneZahl", False) = False Then
         ActivePage.PageSheet.AddNamedRow visSectionProp, "LaneZahl", 0
      End If

      If ActivePage.PageSheet.Cells("prop.LaneZahl").Formula = "" Then
         ActivePage.PageSheet.Cells("prop.LaneZahl.Type").Formula = 2
         ActivePage.PageSheet.Cells _
         ("prop.LaneZahl.Label").Formula = "=""LaneZahl"""
         ActivePage.PageSheet.Cells _
            ("prop.LaneZahl.Invisible").Formula = True
         ActivePage.PageSheet.Cells("prop.LaneZahl").Formula = 0
      End If
End Sub
```

Listing 6.14 Jedes neue Zeichenblatt erhält ein Datenfeld (benutzerdefinierte Eigenschaft)

Zugegeben: Das Überprüfen, ob bereits ein Abschnitt und eine Zeile existieren, ist überflüssig – sie sind natürlich noch nicht vorhanden. Und dann kann gelöscht werden. Da mehrere Shapes markiert werden können, da nicht jedes markierte Shape ein ORG-Lane-Shape ist, muss dies überprüft werden. Wird in der Auswahlmarkierung ein solches Shape gefunden, wird *LaneZahl* um 1 verringert, und alle ORG-Lane-Shapes, deren Position sich rechts von dem zu löschenden Shape befindet, werden um eine Position nach links verschoben.

```
Private Sub Document_BeforeSelectionDelete _
   (ByVal Selection As IVSelection)
   Dim vsShape As Shape
   Dim vsShapeInnen As Shape
   Dim dblXPos As Double

      For Each vsShape In Selection
         If Left(vsShape.Name, 8) = "Org-Lane" Then
            ActivePage.PageSheet.Cells _
               ("Prop.LaneZahl").Formula = _
               ActivePage.PageSheet.Cells _
               ("Prop.LaneZahl").Formula - 1

    dblXPos = vsShape.Cells("PinX").Result(visMillimeters)

        For Each vsShapeInnen In ActivePage.Shapes
           If Left(vsShapeInnen.Name, 8) = "Org-Lane" And _
              vsShapeInnen.Cells _
             ("PinX").Result(visMillimeters) > dblXPos Then
              vsShapeInnen.Cells _
              ("PinX").Result(visMillimeters) = _
               vsShapeInnen.Cells _
               ("PinX").Result(visMillimeters) - _
               vsShapeInnen.Cells _
               ("Width").Result(visMillimeters)
            End If
         Next
      End If
   Next
End Sub
```

Listing 6.15 Vorhandene Shapes werden beim Einfügen von neuen Shapes verschoben

Da mehrere Shapes zu einer Gruppe und damit zu einem neuen Shape zusammengefasst werden können, kann auch auf dieses Ereignis reagiert werden. Es heißt *ShapeParentChanged* und wird sowohl beim Gruppieren als auch beim Auflösen einer Gruppe aufgerufen. Wenn der Benutzer vorhandenen Text ändern oder neuen Text auf ein Shape schreiben möchte, wird *BeforeShapeTextEdit* ausgelöst, wenn er das Schreiben beendet (also mit der Maus aus das Shape klickt oder die Esc-Taste drückt), wird *BeforeShapeTextEdit* verwendet.

Im folgenden Beispiel wurde ein Shape *Textmodus* genannt. Es stellt ein schwarzes »T« dar. Schreibt der Benutzer etwas, leuchtet das »T« rot, wie man in Abbildung 6.12 erkennen kann. Hat der Benutzer das Schreiben beendet, erlischt die rote Farbe:

```
Private Sub Document_BeforeShapeTextEdit(ByVal Shape As IVShape)
   ActivePage.Shapes("Textmodus").Cells("FillForegnd").Formula = 2
End Sub
Private Sub Document_ShapeExitedTextEdit(ByVal Shape As IVShape)
   ActivePage.Shapes("Textmodus").Cells("FillForegnd").Formula = 0
End Sub
```

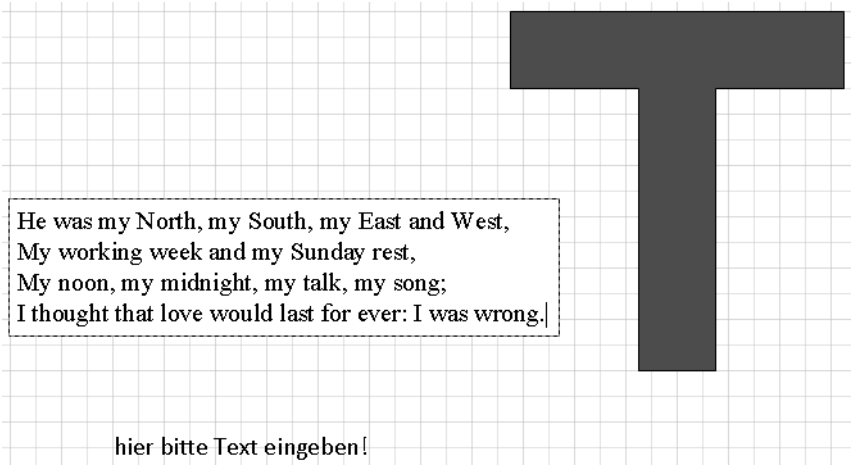

Abbildung 6.12 Ein Signal, das anzeigt, dass der Benutzer schreibt

In diesem Beispiel entspricht der Farbe Schwarz für *FillForegnd* 0, während 2 für Rot steht.

Mit diesem Wissen können nun kleinere Beispiele erstellt werden. Sie werden in Anlehnung an die Beispiele erstellt, die Visio 2000 mitlieferte. In der Datei *VBA Distance From.vsd* befindet sich ein Tankstellensymbol auf dem Zeichenblatt:

Dieses Shape trägt den Namen *Fuel*. Damit es bei allen Ereignissen eindeutig identifiziert werden kann, wird beim Öffnen der Datei mit einer Objektvariablen (*vsBaseShape*) ein Verweis auf dieses Shape hergestellt:

```
Private Sub Document_DocumentOpened(ByVal doc As IVDocument)
   Set objBaseShape = ActivePage.Shapes("Fuel")
End Sub
```

Damit das Makro funktioniert, muss die Objektvariable am Anfang prozedurübergreifend deklariert werden:

```
Private vsBaseShape As Shape
```

Wird nun ein (beliebiges) Shape aus einer Schablone auf das Zeichenblatt gezogen, tritt das Ereignis *ShapeAdded* ein. Mit ihm wird eine Objektvariable vom Typ *Shape* übergeben. Damit kann die Lage des neuen Shapes ausfindig gemacht werden:

```
Private Sub Document_ShapeAdded(ByVal Shape As IVShape)
   Dim strMsg As String
   Dim dblMinimumDistance As Double
   Dim dblDistance As Double

   dblMinimumDistance = 84

   If vsBaseShape.Parent = Shape.Parent Then
      dblDistance = vsBaseShape.DistanceFromPoint _
         (Shape.Cells("PinX"), Shape.Cells("PinY"), 0)
      If dblDistance < dblMinimumDistance Then
           strMsg = "Die Entfernung ist korrekt."
       Else
           strMsg = "Die Entfernung ist zu gering."
       End If
   Else
       strMsg = "Die Shapes liegen auf verschiedenen Zeichenblättern."
   End If

MsgBox strMsg
End Sub
```

Listing 6.16 Die Entfernung zwischen zwei Shapes ermittelt

Abbildung 6.13 Ein Shape wird auf das Zeichenblatt gezogen.

Einige Erläuterungen hierzu: Die wichtigste Funktion in diesem Beispiel ist die Eigenschaft *DistanceFromPoint* des *Shape*-Objekts. Mit ihr kann die Entfernung zu einem Punkt bestimmt werden. Mit der Eigenschaft *DistanceFrom* könnte die Entfernung zu einem Shape berechnet werden. Um zu überprüfen, ob die beiden Shapes auf dem gleichen Zeichenblatt liegen, wird die Eigenschaft *Parent* verwendet. Im Sinne der Objekthierarchie stellt das *Parent*-Objekt das Blatt dar, auf dem sich das Shape befindet. Erst wenn beide gleich sind, kann die Entfernung bestimmt werden.

Auch im folgenden Beispiel wird das Ereignis *ShapeAdded* verwendet, um den Ort des auf das Zeichenblatt gezogenen Shapes ausfindig zu machen:

```
Private Sub Document_ShapeAdded(ByVal Shape As IVShape)
   Dim ShapeOnPage As Shape
   Dim ReturnedSelection As Selection
   Dim strShapeText As String

   Set ReturnedSelection = _
      Shape.SpatialNeighbors(visSpatialContainedIn, 0#, 0)

   If ReturnedSelection.Count = 0 Then
      Shape.Text = Shape.Name & " befindet sich außerhalb."
   Else
      For Each ShapeOnPage In ReturnedSelection
         strShapeText = strShapeText & Shape.Name & _
         " befindet sich im " & ShapeOnPage.Name
      Next
   End If

   Shape.Text = strShapeText
End Sub
```

Listing 6.17 Befindet sich das neue Shape innerhalb eines vorhandenen Shapes?

Ein Shape wird aus der Schablone auf das Zeichenblatt gezogen. Zu diesem Shape werden alle Shapes ermittelt (*ReturnedSelection*), innerhalb derer sich das Shape befindet. Dabei hilft die Eigenschaft *SpatialNeighbors*, die ein *Selection*-Objekt zurückgibt. Die Anzahl der Shapes wird ermittelt. Beträgt sie 0, liegt das Shape außerhalb. Ist sie größer oder gleich 1, werden alle diese Shapes durchlaufen und als Text in dem neuen Shape aufgelistet.

Die Syntax der Eigenschaft *SpatialNeighbors* lautet:

```
objRet = Shape.SpatialNeighbors(Relation, Tolerance, Flags, [ResultRoot])
```

Mit Relation ist eine Objektkonstante gemeint, die folgende Werte annehmen kann:

Bezeichnung	Wert	Bedeutung
visSpatialOverlap	1	Zwei Shapes können sich überlappen
visSpatialContain	2	Ein Shape beinhaltet vollkommen das andere Shape
visSpatialContainedIn	4	Ein Shape wird vollkommen von einem anderen Shape eingeschlossen
visSpatialTouching	8	Ein Shape berührt ein anderes Shape, ohne es einzuschließen

Tabelle 6.3 Die Systemkonstanten der Beziehungsmöglichkeiten zweier Shapes

Sie könnten übrigens auch die Eigenschaft *SpatialRelation* verwenden. Mit ihrer Hilfe kann überprüft werden, ob sich zwei Shapes ineinander befinden. Sie gibt eine Integer-Zahl zurück. Oder *SpatialSearch*, der zum Toleranzwert zusätzlich zwei Koordinaten verlangt.

Wird nun ein Shape aus der Schablone auf die Zeichnung gezogen, erscheint beispielsweise der in Abbildung 6.14 dargestellte Text.

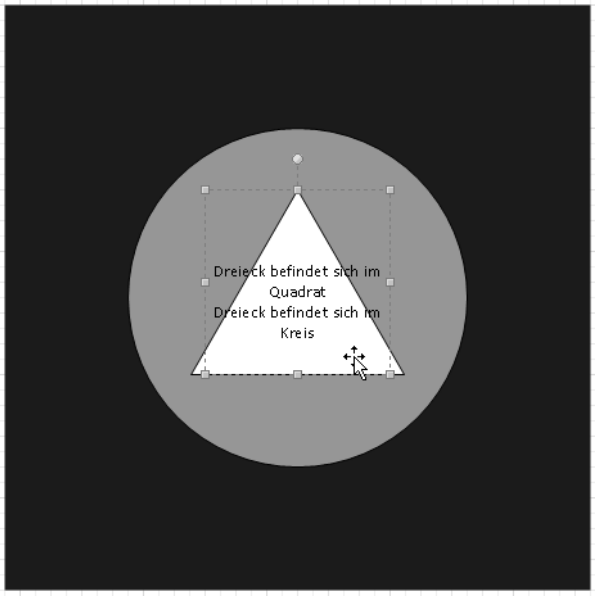

Abbildung 6.14 Das Dreieck befindet sich im Kreis und im Quadrat

Weitere Ereignisse

Natürlich ist das Objekt *Document* nicht das einzige Objekt, auf das verwiesen werden kann und das Ereignisse besitzt. Die Visio-Objekte mit Ergebnissen, die zur Verfügung stehen, finden Sie in folgender Liste:

- Application
- InvisibleApp
- Windows
- Window
- Documents
- Document
- Pages
- Page
- Masters
- Master
- Selection

- Shape
- Characters
- Styles
- Style
- Section
- Row
- Cell

VBA stellt das Schlüsselwort *WithEvents* zur Verfügung, mit dem eine Ereignisbearbeitung des Quellenobjekts möglich ist. Wird beispielsweise *ThePage* in *ThisDocument* als Visio-Zeichenblatt deklariert, kann damit gearbeitet werden:

```
Private WithEvents ThePage As Visio.Page
```

ThePage enthält neben den bekannten Ereignissen *BeforePageDelete*, *PageChanged*, *PageDeleteCanceled* die beiden Ereignisse *ConnectionsAdded* und *ConnectionsDeleted*. Mit ihnen kann überprüft werden, welcher Verbinder mit welchem Shape eine Verbindung eingeht oder löst:

```
Private Sub ThePage_ConnectionsAdded(ByVal Connects As Visio.IVConnects)
   Dim cnt As Connect

   For Each cnt In Connects
      With cnt
         MsgBox .FromCell.Name & " in " & _
            .FromSheet.Name & " wurde verbunden mit " & _
            .ToCell.Name & " in .ToSheet.Name, vbInformation
      End With
   Next
End Sub
Private Sub ThePage_ConnectionsDeleted(ByVal Connects _
   As Visio.IVConnects)
   Dim cnt As Connect

   For Each cnt In Connects
      With cnt
      MsgBox .FromCell.Name & " in " & .FromSheet.Name & _
         " wurde gelöst von " & .ToCell.Name & " in " & _
         .ToSheet.Name, vbInformation
      End With
   Next

End Sub
```

Listing 6.18 Welcher Verbinder wurde geklebt oder gelöst?

Damit dies funktioniert, muss beim Start *ThePage* festgelegt werden:

```
Private Sub Document_DocumentOpened(ByVal doc As Visio.IVDocument)
   Set ThePage = Visio.ActivePage
End Sub
```

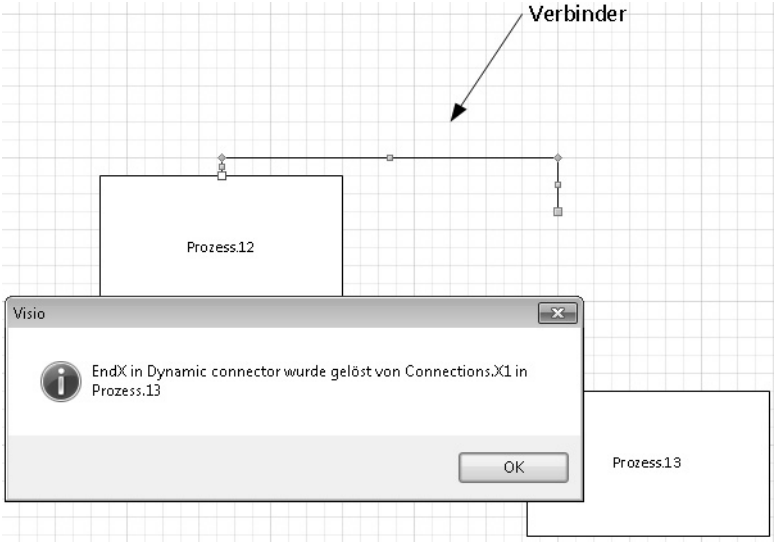

Abbildung 6.15 Ein weiteres Ereignis mit anderen Objekten

Analog funktioniert auch das folgende Beispiel. Ein Verbinder mit Namen *Verbinder.1* besitzt zwei benutzerdefinierte Datenfelder *Von* und *Zu*. Diese werden als Felder vor und nach dem Text » ist der Boss von « eingefügt. Per Programmierung wird nun beim Verknüpfen der Linie mit einem Shape der Name des Shapes (nicht der Text!) in die Datenfelder geschrieben, und diese werden angezeigt:

```
Dim WithEvents pagobj As Visio.Page

Private Sub Document_RunModeEntered(ByVal doc As Visio.IVDocument)
   Set pagobj = Visio.ActivePage
End Sub

Private Sub pagobj_ConnectionsAdded(ByVal Connects As Visio.IVConnects)
   Dim celobj As Visio.Cell
   Dim PosPeriod As Integer
   Dim strFromName As String
   strFromName = Connects.FromSheet.Name
   PosPeriod = InStr(1, strFromName, ".")
   If PosPeriod <> 0 Then
```

Listing 6.19 Text wird beim Verbinden geändert

```
      strFromName = Left(strFromName, PosPeriod - 1)
   End If

   If strFromName <> "Verbinder" Then Exit Sub

   If Connects(1).FromPart = visEnd Then
     Set celobj = Connects.FromSheet.Cells("Prop.Zu.Value")
   ElseIf Connects(1).FromPart = visBegin Then
    Set celobj = Connects.FromSheet.Cells("Prop.Von.Value")
   End If

   celobj.Formula = Chr$(34) & Connects.ToSheet.Name & Chr$(34)
End Sub

Private Sub pagobj_ConnectionsDeleted(ByVal Connects As Visio.IVConnects)
   Dim celobj As Visio.Cell
   Dim PosPeriod As Integer
   Dim strFromName As String

   strFromName = Connects.FromSheet.Name
   PosPeriod = InStr(1, strFromName, ".")
   If PosPeriod <> 0 Then
      strFromName = Left(strFromName, PosPeriod - 1)
   End If
   If strFromName <> "Verbinder" Then Exit Sub

   If Connects(1).FromPart = visEnd Then
      Set celobj = _
         Connects.FromSheet.Cells("Prop.Zu.Value")
   ElseIf Connects(1).FromPart = visBegin Then
      Set celobj = _
         Connects.FromSheet.Cells("Prop.Von.Value")
   End If
   celobj.Formula = Chr$(34) & Chr$(34)
End Sub
```

Listing 6.19 Text wird beim Verbinden geändert *(Fortsetzung)*

Im ersten Teil der beiden Prozeduren wird jeweils überprüft, ob es sich um einen Verbinder handelt. Wenn ja, wird überprüft, ob Anfang oder Ende geklebt oder gelöst wurde. Bei jeder der vier Aktionen wird in die Zelle *Von* oder *Zu* entweder "" geschrieben (das heißt, der Text wird gelöscht), oder es wird der Name des Shapes in Anführungszeichen hineingeschrieben. Drei Varianten werden in den Teilen von in Abbildung 6.16 gezeigt.

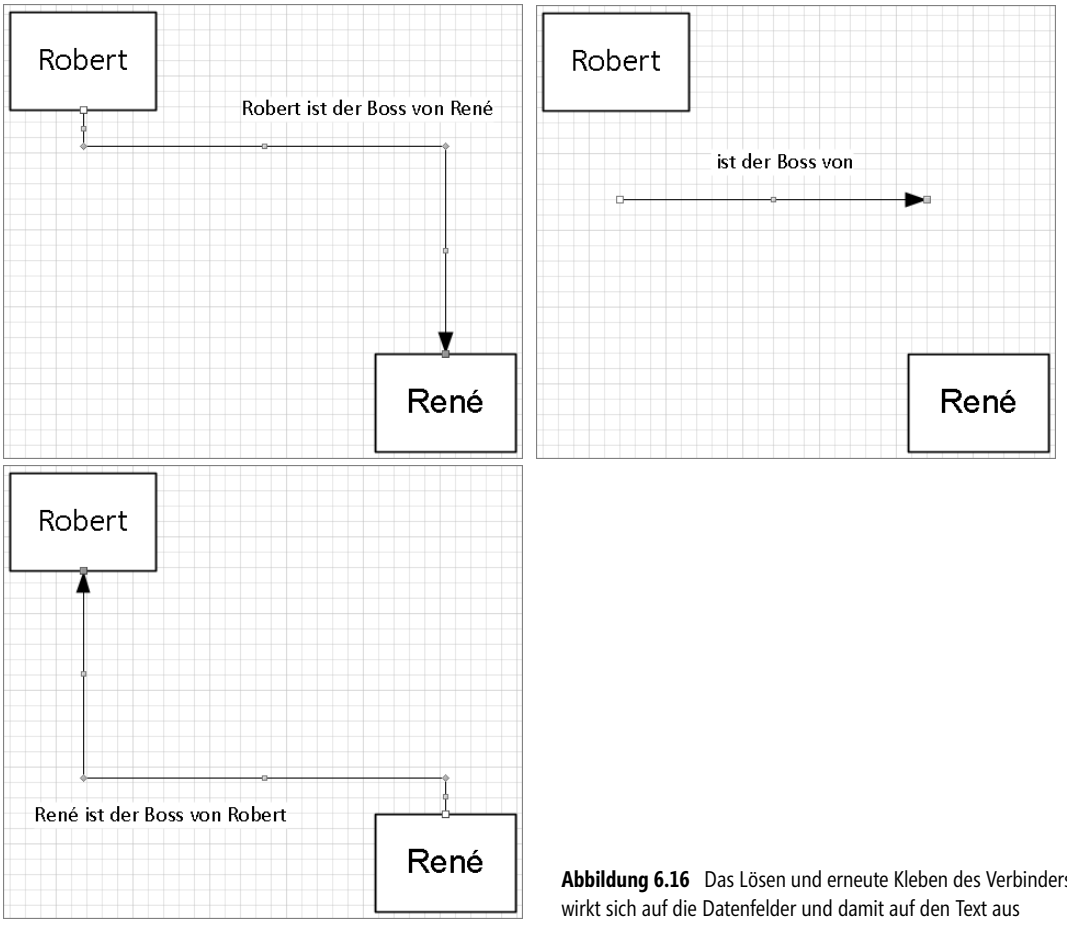

Abbildung 6.16 Das Lösen und erneute Kleben des Verbinders wirkt sich auf die Datenfelder und damit auf den Text aus

So angenehm einfach sich der Umgang mit den verschiedenen Ereignissen gestaltet, so schwierig ist es, wenn bestimmte Ereignisse gesucht werden. So soll beispielsweise abgefangen werden, ob der Benutzer ein Shape verändert, das heißt: verschiebt, formatiert oder in seiner Größe ändert. Das Ereignis *ShapeChanged* gilt erstaunlicherweise nur für Dinge, die nicht in den ShapeSheet-Zellen gespeichert werden, also Änderungen des Namens, der ID und der drei Werte, die im Befehl *Entwicklertools/Shape-Design/Shape-Name* in *Data1* bis *Data3* gespeichert werden.

Um das Ereignis-Modell vollständig verwenden zu können, muss ein *Sink*-Objekt erstellt werden. Dies ist ein Klassenmodul, für das die *visEvtProc*-Methode definiert wurde. Das Klassenmodul wird hinzugefügt (Abbildung 6.17); im Eigenschaftenfenster kann sein Name geändert werden.

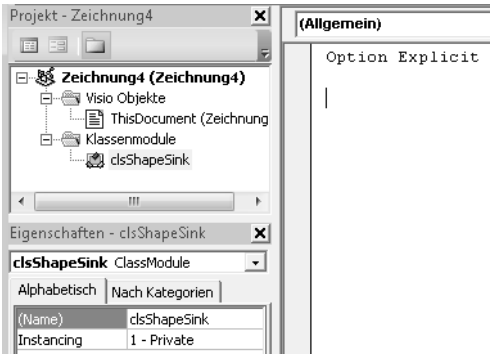

Abbildung 6.17 Die neue Klasse

Nachdem mit der Deklaration

```
Dim sinks As New Collection
```

in den *ThisDocument*-Objekten die neue Sammlung deklariert ist, kann in der neuen Klasse mit

```
Private WithEvents m_shpObj As Visio.Shape
```

eine Referenz auf ein Visio-Shape hergestellt werden. Damit jedes neue *Sink*-Objekt der Sammlung hinzugefügt werden kann, wird eine Startprozedur *InitWith* im Klassenmodul erstellt:

```
Public Sub InitWith(ByVal aShape As Visio.Shape)
   Set m_shpObj = aShape
End Sub
```

Sie wird aufgerufen, wenn ein neues Shape erzeugt wird. Der zugehörige Befehl lautet:

```
Private Sub Document_ShapeAdded(ByVal Shape As Visio.IVShape)
   Dim sinkObj As New ShapeSink
   sinkObj.InitWith Shape
   sinks.Add sinkObj, Shape.UniqueID(visGetOrMakeGUID)
End Sub
```

So wird ein *Sink*-Objekt zu Sammlung hinzugefügt. Damit das Objekt eindeutig identifiziert ist, wird die *UniqueID* verwendet.

Wird nun das Shape geändert, das heißt: verschoben, deformiert oder formatiert, kann das Ereignis *CellChanged* des Objekts *m_shpObj* verwendet werden:

```
Private Sub m_shpObj_CellChanged(ByVal Cell As Visio.IVCell)
   MsgBox Cell.Shape.Name & " " & Cell.Name & _
   " wurde geändert in: =" & Cell.Formula
End Sub
```

Da mit dem Ereignis *ShapeChanged* nur wenige Änderungen am Shape abgefangen werden können, benötigen Sie ein anderes, besseres Ereignis, um auf einen Benutzerzugriff reagieren zu können – nämlich *CellChanged*.

```
Private Sub m_shpObj_ShapeChanged(ByVal Shape As IVShape)
   MsgBox "Ich bin das neue Shape: " & Shape.Name
End Sub
```

Ein weiteres Beispiel zum Ereignis *CellChanged* finden Sie in Kapitel 4 im Abschnitt »Die Layer«.

Und beim Löschen? Dort können die Elemente der *Sink*-Sammlung wieder gelöscht werden. Beispielsweise so:

```
Private Sub Document_BeforeSelectionDelete(ByVal _
   Selection As Visio.IVSelection)
   Dim i As Integer
   On Error Resume Next
   For i = 1 To Selection.Count
      sinks.Remove Selection(i).UniqueID(visGetGUID)
   Next i
End Sub
```

Damit kann auf jedes einzelne Ereignis (wie beispielsweise in Abbildung 6.18) reagiert werden:

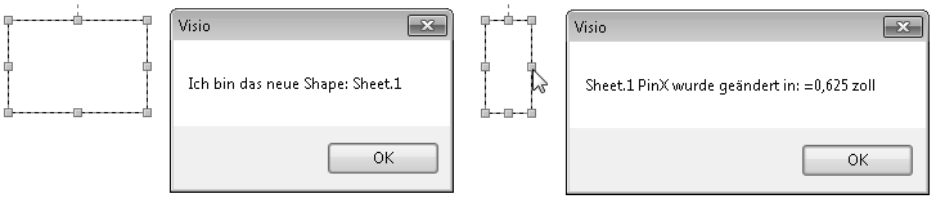

Abbildung 6.18 Jede Änderung wird angezeigt

Der Programmierer kann Ereignisobjekte der Ereignisliste hinzufügen, und zwar jedes Objekt, das von der Schnittstelle der Visio-Automation gefördert wird. Die Ereignisliste fördert zwei Methoden zum Anhängen von Vorfällen:

```
Event.Add
Event.AddAdvice
```

Während Erstere ein Ereignis hinzufügt, das gestartet wird, wenn das Ereignis eintritt, fügt *AddAdvice* ein Ereignis hinzu, das innerhalb des aufgerufenen Programms eine Handlungsfunktion startet, wenn das Ereignis eintritt. Die Adresse desjenigen, der das Ereignis bearbeitet, wird weitergeleitet. Konkret könnte dies so aussehen: Im Objekt *ThisDocument* wird beim Start folgende Prozedur aufgerufen:

```
Private Sub Document_RunModeEntered(ByVal doc As Visio.IVDocument)
   Dim appEvtList As Object
   Dim g_sink As New EventSink
   Set appEvtList = Application.EventList
   appEvtList.AddAdvise visEvtCodeDocSave, g_sink, "", _
   "Das Dokument wird gespeichert"
   appEvtList.AddAdvise visEvtCodeBefSelDel, g_sink, "", _
   "Vor dem Löschen der Shapes"
   appEvtList.AddAdvise &H8040, g_sink, "", "Ein Shape wird hinzugefügt"
   appEvtList.AddAdvise (visEvtDel + visEvtPage), g_sink, _
   "", "Seite wird gelöscht"
   appEvtList.AddAdvise (&H8010), g_sink, "", "Seite wird hinzugefügt"
End Sub
```

Der Text, der am Ende übergeben wird (»Seite wird hinzugefügt«), ist überflüssig. Er dient lediglich als Kommentar. Nun können die einzelnen Ereignisse abgefangen werden:

```
Public Sub VisEventProc(eventCode As Integer, _
   sourceObj As Object, eventID As Long, _
   seqNum As Long, subjectObj As Object, moreInfo As Variant)
   Dim i As Integer
   If eventCode = visEvtCodeBefSelDel Then
      For i = 1 To subjectObj.Count
         MsgBox subjectObj(i).Name & " wird nun gelöscht!"
      Next
   ElseIf eventCode = &H8010 Then
      MsgBox "Neue Seite"
   ElseIf eventCode = visEvtCodeDocSave Then
      MsgBox "Nun wird gespeichert"
   ElseIf eventCode = visEvtDel + visEvtPage Then
      MsgBox "Seite wird gelöscht: " & ActivePage.Name
   ElseIf eventCode = &H8040 Then
      MsgBox "Shape wird hinzugefügt: " & subjectObj.Name
   Else
      MsgBox "Unbekanntes Ereignis: " & Str$(eventCode)
   End If
End Sub
```

Listing 6.20 Die Ereignisse können abgefangen werden

Dabei haben die Parameter folgende Bedeutung: Der *eventCode* entspricht der übergebenen EventID. *SourceObj* ist die Referenz auf die Quelle des Objekts, *eventID* die ID des Ereignisses in der EventList der Quelle des Objekts. Mit *seqNum* ist die Event Sequence ID dieser Visio-Instanz gemeint, *subjectObj* bezeichnet das Objekt des Ereignisses, und *moreInfo* liefert weitere Informationen über das Ereignis. Im obigen Fall genügt der *eventCode*. Dafür stehen dem Benutzer die Ereignisse zur Verfügung:

Ereignis	EventCode	Zahlencode
AfterModal	visEvtApp+visEvtAfterModal	&H1040
AppActivated	visEvtApp+visEvtAppActivate	&H1001
AppDeactivated	visEvtApp+visEvtAppDeactivate	&H1002
AppObjectActivated	visEvtApp+visEvtObjActivate	&H1004
AppObjectDeactivated	visEvtApp+visEvtObjDeactivate	&H1008
BeforeDocumentClose	visEvtDel+visEvtDoc	&H4002
BeforeDocumentSave	visEvtCodeBefDocSave	&H0007 (7)
BeforeDocumentSaveAs	visEvtCodeBefDocSaveAs	&H0008 (8)
BeforeMasterDelete	visEvtDel+visEvtMaster	&H4008
BeforeModal	visEvtApp+visEvtBeforeModal	&H1020
BeforePageDelete	visEvtDel+visEvtPage	&H4010
BeforeQuit	visEvtApp+visEvtBeforeQuit	&H1010
BeforeSelectionDelete	visEvtCodeBefSelDel	&H0385 (901)
BeforeShapeDelete	visEvtDel+visEvtShape	&H4040
BeforeStyleDelete	visEvtDel+visEvtStyle	&H4004
BeforeWindowClosed	visEvtDel+visEvtWindow	&H4001
BeforeWindowPageTurn	visEvtCodeBefWinPageTurn	&H02BF (703)
BeforeWindowSelDelete	VisEvtCodeBefWinSelDel	&H02BE (702)
CellChanged	visEvtMod+visEvtCell	&H2800
ConnectionsAdded	visEvtAdd+visEvtConnect	&H8100
ConnectionsDeleted	visEvtDel+visEvtConnect	&H4100
DesignModeEntered	visEvtCodeDocDesign	&H0006 (6)
DocumentAdded	visEvtAdd+visEvtDoc	&H8002
DocumentChanged	visEvtMod+visEvtDoc	&H2002
DocumentCreated	visEvtCodeCreate	&H0001 (1)
DocumentOpened	visEvtCodeOpen	&H0002 (2)
DocumentSaved	visEvtCodeSave	&H0003 (3)
DocumentSavedAs	visEvtCodeSaveAs	&H0004 (4)
EnterScope	visEvtCodeEnterScope	&H00ca (202)
ExitScope	visEvtCodeExitScope	&H00cb (203)
FormulaChanged	visEvtMod+visEvtFormula	&H3000

Tabelle 6.4 Die Visio-Ereignisse

Ereignis	EventCode	Zahlencode
MarkerEvent	visEvtApp+visEvtMarker	&H1100
MasterAdded	visEvtAdd+visEvtMaster	&H8008
MasterChanged	visEvtMod+visEvtMaster	&H2008
MustFlushScopeBeginning	visEvtApp+visEvtCodeBefForcedFlush	&H00C8 (200)
MustFlushScopeEnded	visEvtApp+visEvtCodeAfterForcedFlush	&H00C9 (201)
NoEventsPending	visEvtApp+visEvtNonePending	&H1200
PageAdded	visEvtAdd+visEvtPage	&H8010
PageChanged	visEvtMod+visEvtPage	&H2010
RunModeEntered	visEvtCodeDocRunning	&H0005 (5)
SelectionAdded	visEvtCodeSelAdded	&H0386 (902)
SelectionChanged	visEvtCodeBefWinSelChange	&H02BD (701)
ShapeAdded	visEvtAdd+visEvtShape	&H8040
ShapeChanged	visEvtMod+visEvtShape	&H2040
ShapeParentChanged	visEvtCodeShapeParentChange	&H0322 (802)
ShapesDeleted	visEvtCodeShapeDelete	&H0321 (801)
StyleAdded	visEvtAdd+visEvtStyle	&H8004
StyleChanged	visEvtMod+visEvtStyle	&H2004
TextChanged	visEvtMod+visEvtText	&H2080
ViewChanged	visEvtCodeViewChanged	&H02c1 (705)
VisioIsIdle	visEvtApp+visEvtIdle	&H1400
WindowActivated	visEvtApp+visEvtWinActivate	&H1080
WindowOpened	visEvtAdd+visEvtWindow	&H8001
WindowChanged	visEvtMod+visEvtWindow	&H2001 (8193)
WindowTurnedToPage	visEvtCodeWinPageTurn	&H02C0 (704)

Tabelle 6.4 Die Visio-Ereignisse *(Fortsetzung)*

Abbildung 6.19 Ein Ereignis wurde ausgelöst

Ein kleines Beispiel soll diesen komplexen Sachverhalt verdeutlichen. Über den Befehl *Ansicht/Makros/Add-Ons/Visio-Extras/Shape-Fläche und -Umfang* werden in einem Fenster die Fläche und der Umfang angezeigt. Das ist ärgerlich, denn wenn Sie eine der beiden Informationen als beschriftendes Element haben möchten, müssen Sie diese Zahl abtippen. Auch die Feldfunktion

```
=Höhe*Breite
```

hilft nur bei Rechtecken weiter – alle anderen Formen werden nicht korrekt berechnet. Im folgenden Beispiel sollen auf den Verbindern (es sind Rohrleitungen) die exakten Längenangaben stehen.

Zu einer Zeichnung wird ein Klassenmodul mit dem Namen *ShapeSink* hinzugefügt. Damit es verwendet werden kann, wird bei jedem neuen Shape deklariert:

```
Private Sub Document_ShapeAdded(ByVal Shape As Visio.IVShape)
    Dim sinkObj As New ShapeSink
    sinkObj.InitWith Shape

    sinks.Add sinkObj, Shape.UniqueID(visGetOrMakeGUID)
End Sub
```

Guter Programmierstil setzt auch ein Löschen voraus:

```
Private Sub Document_BeforeSelectionDelete _
    (ByVal Selection As Visio.IVSelection)
    Dim i As Integer
    On Error Resume Next
    For i = 1 To Selection.Count
        sinks.Remove Selection(i).UniqueID(visGetGUID)
    Next i
End Sub
```

Und in der Klasse findet sich der eigentliche Code. Bei jeder Zelländerung, das heißt Änderung des Shapes, wird überprüft, ob es sich um ein eindimensionales Shape handelt (*If Cell.Shape.CellExists("BeginX", True) = True*). Falls ja, wird nachgesehen, ob das Master-Shape den Namen *Rohr* trägt. Falls auch dies der Fall ist, wird die Beschriftung geändert. Die berechnete Länge zeigt Ihnen Abbildung 6.20. Hierzu wird die Eigenschaft *LengthIU* des Shapes verwendet.

```
Private WithEvents m_shpObj As Visio.Shape

Public Sub InitWith(ByVal aShape As Visio.Shape)
   Set m_shpObj = aShape
End Sub

Private Sub m_shpObj_CellChanged _
   (ByVal Cell As Visio.IVCell)
   Dim i As Integer
   If Cell.Shape.CellExists("BeginX", True) = True Then
      For i = 1 To Cell.Shape.LayerCount
```

Listing 6.21 Die Länge eines Rohrs wird angezeigt

```
         If Cell.Shape.Layer(i).Name = "Rohr" Then
            Cell.Shape.Text = "Länge: " &
            Format(Cell.Shape.LengthIU, "#,##0.00")
         End If
      Next
    End If
End Sub

Private Sub m_shpObj_ShapeChanged(ByVal Shape As IVShape)
   Dim i As Integer
    If Shape.CellExists("BeginX", True) = True Then
      For i = 1 To Shape.LayerCount
         If Shape.Layer(i).Name = "Rohr" Then
            Shape.Text = "Länge: " & Format(Shape.LengthIU, "#,##0.00")
         End If
      Next
    End If
End Sub
```

Listing 6.21 Die Länge eines Rohrs wird angezeigt *(Fortsetzung)*

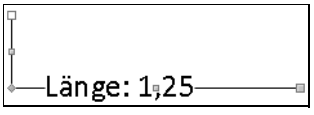

Abbildung 6.20 Nun können die Rohre verlegt werden

Programmierte Symbole in einer Visio-Version bis 2007 – was passiert mit ihnen?

Bis Visio 2007 konnten per Programmierung neue Symbole in der Symbolleiste erzeugt werden. Sie sehen eine solche Symbolleiste in Abbildung 6.21.

```
Sub NeueSymbolleiste()
   Dim vsSymbleisten As CommandBars
   Dim vSymbleiste As CommandBar
   Dim vsSymb As CommandBarButton
   Dim i As Integer
   On Error Resume Next

   Set vsSymbleisten = Application.CommandBars
   vsSymbleisten("Meine eigene Symbolleiste").Delete
   Set vSymbleiste = vsSymbleisten.Add _
      ("Meine eigene Symbolleiste", msoBarTop, False, True)

   vSymbleiste.Visible = True

   Set vsSymb = vSymbleiste.Controls.Add(msoControlButton, 108)
   With vsSymb
      .DescriptionText = "Daten bereinigen"
```

Listing 6.22 Eine neue Symbolleiste mit Symbolen wird in Visio 2007 erzeugt

```
      .TooltipText = "Daten bereinigen"
      .Caption = "Daten bereinigen"
      .OnAction = "DatenBereinigen"
      .Style = msoButtonIconAndCaption
   End With

   Set vsSymb = vSymbleiste.Controls.Add(msoControlButton, 186)
   With vsSymb
      .DescriptionText = "Datenauswertung"
      .TooltipText = "Datenauswertung"
      .Caption = "Datenauswertung"
      .OnAction = "Datenauswertung"
      .Style = msoButtonIconAndCaption
   End With

   Set vsSymb = vSymbleiste.Controls.Add(msoControlButton, 14145)
   With vsSymb
      .DescriptionText = "Statistik"
      .TooltipText = "Statistik"
      .Caption = "Statistik"
      .OnAction = "Statistik"
      .Style = msoButtonIconAndCaption
   End With

   Set vsSymb = vSymbleiste.Controls.Add(msoControlButton, 682)
   With vsSymb
      .DescriptionText = "persönliche Daten einlesen"
      .TooltipText = "persönliche Daten einlesen"
      .Caption = "persönliche Daten einlesen"
      .OnAction = "personlDaten"
      .Style = msoButtonIconAndCaption
   End With

   Set vsSymb = vSymbleiste.Controls.Add(msoControlButton, 793)
   With vsSymb
      .DescriptionText = "Hilfe"
      .TooltipText = "Hilfe"
      .Caption = "Hilfe"
      .OnAction = "Hilfe"
      .Style = msoButtonIconAndCaption
   End With
End Sub
```

Listing 6.22 Eine neue Symbolleiste mit Symbolen wird in Visio 2007 erzeugt *(Fortsetzung)*

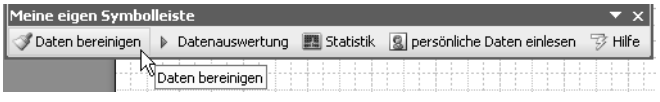

Abbildung 6.21 Die neue programmierte Symbolleiste in Visio 2007

Jedem internen Visio-Symbol ist eine Nummer zugeordnet. Um herauszufinden, hinter welcher Nummer sich welches Symbol verbirgt, kann eine kleine Schleife geschrieben werden. Jedes Symbol wird mit seiner Nummer angezeigt (Abbildung 6.22):

```
Sub SymbolleistenErzeugen()
   Dim vsSymbleisten As CommandBars
   Dim vSymbleiste As CommandBar
   Dim vsSymb As CommandBarButton
   Dim i As Integer
   On Error Resume Next

   Set vsSymbleisten = Application.CommandBars
   vsSymbleisten("Meine eigene Symbolleiste").Delete

   Set vSymbleiste = vsSymbleisten.Add _
      ("Meine eigene Symbolleiste", msoBarTop, False, True)
   vSymbleiste.Visible = True

   For i = 1 To 20000
      Set vsSymb = vSymbleiste.Controls.Add(msoControlButton, i)
      If Err.Number = 0 Then
         vsSymb.TooltipText = i
      End If
      Err.Clear
   Next

End Sub
```

Listing 6.23 Alle Symbole werden angezeigt

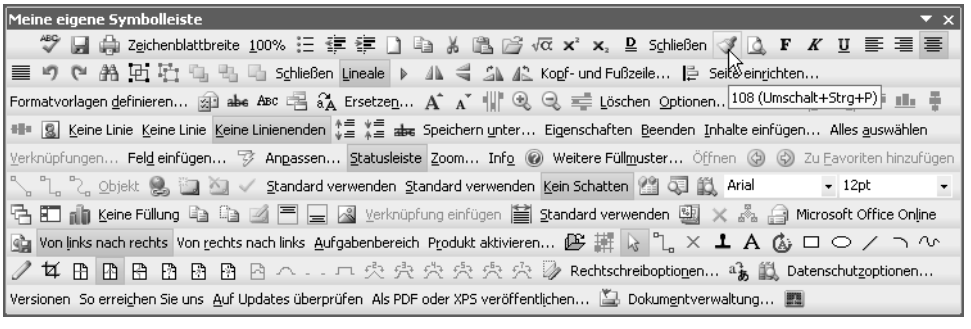

Abbildung 6.22 Die Symbolleiste mit den Symbolen

Wenn Sie diesen Code in Excel (oder Word oder PowerPoint) starten, werden dort die Symbole in der Registerkarte *Add-Ins* angezeigt. Die Abbildung 6.23 zeigt das Ergebnis des Codes von Listing 6.22 in Excel. Dies bedeutet, dass die Programmierung der Symbole und Symbolleisten aus Excel (oder Word oder PowerPoint) problemlos von der Version in die Version 2007 und 2010 übernommen werden kann.

Abbildung 6.23 Die programmierten Symbole in Excel 2010

In Visio gestaltet sich die Situation allerdings anders:

1. Der obenstehende Code wird ausgeführt, ohne dass etwas angezeigt wird.
2. Word-, Excel- und PowerPoint-Dateien sind gezippte XML-Dateien, die dem Office Open XML-Standard entsprechen. In ihnen kann die Definition einer neuen Registerkarte mittels einer weiteren XML-Datei eingefügt werden. Dies geht in Visio leider nicht, weil Visio intern nicht dem Office Open XML-Standard entspricht, das heißt: Eine Visio-Datei ist keine ZIP-Datei, die als Container für weitere Dateien dient.

Deshalb kann das Menüband in Visio nicht analog zu Word, Excel oder PowerPoint verändert werden. Eine Anpassung ist lediglich mit einer Programmiersprache, wie beispielsweise Visual Basic .NET, C# oder C++ möglich. Wie dies funktioniert, wird im letzten Kapitel dieses Buchs erläutert.

Zusammenfassung

Wenn Sie eine Prozedur in einer Visio-Datei starten möchten, erscheint es am einfachsten, diese Prozedur an eine Schaltfläche auf der Zeichnung oder an ein Shape zu binden. Vom Shape kann das Doppelklickverhalten, das Kontextmenü oder ein Ereignis geändert werden. Ebenso können Sie eines der vielen Ereignisse verwenden, welche Microsoft Visio in jeder Datei zur Verfügung stellt.