Druckausgabe

- Überblick
- PrintDocument-Objekt
- Druckvorschau
- Druckdialoge
- Drucken mit OLE-Automation

Die Kapitelüberschrift deutet es schon an: In den folgenden Abschnitten wollen wir uns ausgiebig mit den Möglichkeiten beschäftigen, unter VB.NET etwas aufs Papier zu bringen. Drei Varianten bieten sich in VB.NET an:

- Drucken über die *PrintDocument*-Komponente
- Drucken mit Hilfe von OLE-Automation
- Drucken mit den Crystal Report-Komponenten

Im vorliegenden Kapitel beschränken wir uns auf die beiden ersten Möglichkeiten, das Kapitel 9 beschäftigt sich eingehend mit dem Crystal Report.

Hinweis: Gleich vorweg die gute Nachricht: Das bisherige *Printer*-Objekt sowie alle damit verbundenen Schwierigkeiten gehören der Vergangenheit an.

7.1 Ein Beispiel für den Einstieg

Bevor Sie nun gleich mit viel Faktenwissen, endlosen Tabellen etc. gepeinigt werden, möchten wir Ihnen lieber an einem Kurzbeispiel das Grundkonzept der Druckausgabe über *PrintDocument* demonstrieren.

Beispiel: Druckausgabe eines 10 x 10 cm großen Rechtecks auf dem Standarddrucker

Fügen Sie zunächst in Ihr Formular eine *PrintDocument*-Komponente ein:

Ergänzen Sie dann das *PrintPage*-Ereignis um folgende Zeilen:

```
Private Sub PrintDocument1_PrintPage(ByVal sender As System.Object, _
        ByVal e As System.Drawing.Printing.PrintPageEventArgs) Handles _
          PrintDocument1.PrintPage
   e.Graphics.PageUnit = GraphicsUnit.Millimeter
   e.Graphics.FillRectangle(New SolidBrush(Color.Blue), 30, 30, 100, 100)
End Sub
```

Fügen Sie nun noch einen Button ein, mit dem Sie die *Print*-Methode von *PrintDocument1* aufrufen:

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
                         Handles Button1.Click
   PrintDocument1.Print()
End Sub
```

Das war es auch schon, nach dem Klick auf den Button dürfte Ihr Drucker sich in Bewegung setzen.

Doch was ist der Vorteil einer derartigen ereignisorientierten Programmierung beim Drucken? Die Antwort finden Sie, wenn Sie statt der Druckausgabe zunächst eine Druckvorschau am Bildschirm realisieren möchten.

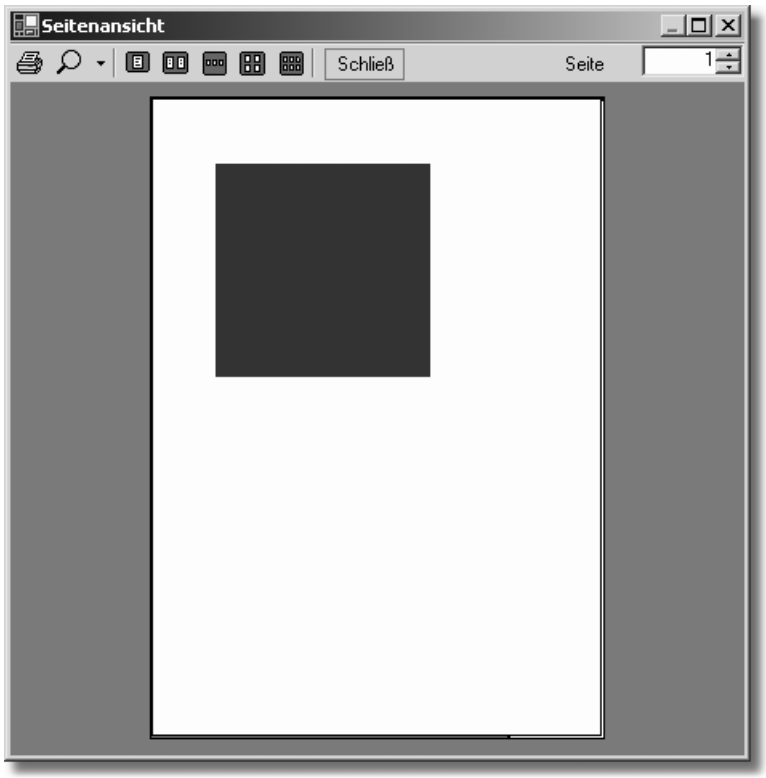
Fügen Sie einfach eine *PrintPreviewDialog*-Komponente in das Formular ein

und verknüpfen diese über die Eigenschaft *Document* mit der bereits vorhandenen *PrintDocument*-Komponente.

Der folgende Aufruf zeigt Ihnen bereits die Druckvorschau mit dem Rechteck an:

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
      Handles Button1.Click
   PrintPreviewDialog1.ShowDialog()
End Sub
```

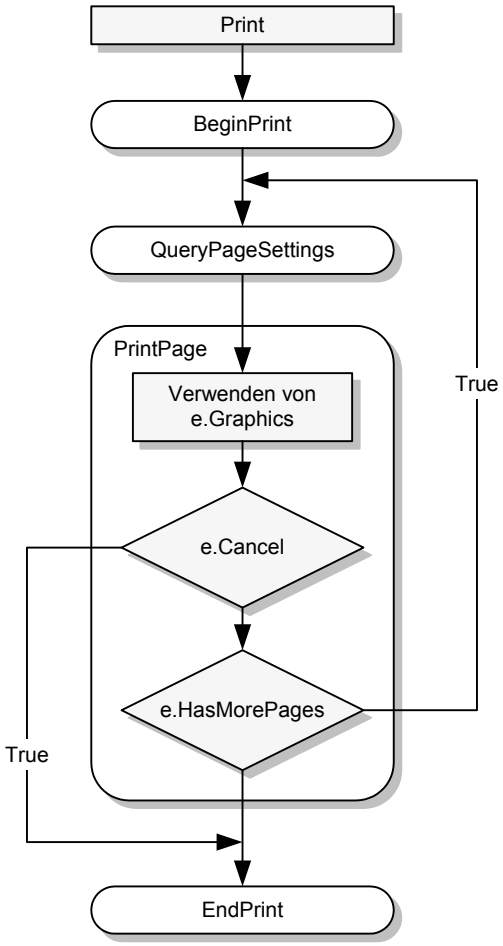
Das erzeugte Druckvorschaufenster:



Hinweis: Eine Trennung beim Ausgabemedium (Papier, Druckvorschau am Bildschirm) gibt es nicht mehr, Sie entwickeln lediglich **eine** Ausgabelogik im *PrintPage*-Ereignis. Alle Ausgaben erfolgen systemneutral über ein dort bereitgestelltes *Graphics*-Objekt.

7.2 Übersicht

Wie Sie bereits dem vorhergehenden Beispiel entnehmen konnten, handelt es sich um ein ereignisorientiertes Programmiermodell. Die folgende Skizze soll Ihnen das Grundprinzip noch einmal verdeutlichen:



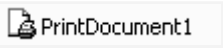
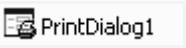
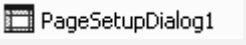
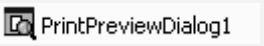
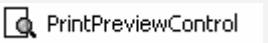
Mit dem Aufruf der *Print*-Methode wird zunächst das *BeginPrint*-Ereignis von *PrintDocument* ausgelöst. Hier bietet sich Ihnen die Möglichkeit, diverse Einstellungen einmalig zu konfigurieren. Nachfolgend wird das Ereignis *QueryPageSettings* vor dem Druck jeder

Seite aufgerufen. Darauf folgt das wohl wichtigste Ereignis: *PrintPage*. Über den Parameter *e* erhalten Sie Zugriff auf das *Graphics*-Objekt des Druckers. Weiterhin legen Sie hier fest, ob weitere Seiten gedruckt werden sollen (*e.HasMorePages*) oder ob der Druck abgebrochen (*e.Cancel*) werden soll. Steht der Druck weiterer Seiten an, wird die Ereigniskette, wie oben abgebildet, erneut durchlaufen.

Damit wird auch klar, dass Sie selbst dafür verantwortlich sind, welche Seite zu welchem Zeitpunkt gedruckt werden soll. Insbesondere im Zusammenhang mit den Druckersetup-Dialogen werden wir noch einigen Aufwand treiben müssen, aber das sind Sie ja bereits nicht anders gewohnt.

7.2.1 Kurzübersicht der Objekte

Gegenüber VB 6 hat sich sowohl qualitativ als auch quantitativ einiges getan. Folgende Komponenten stehen Ihnen im Zusammenhang mit der Druckausgabe mittlerweile zur Verfügung:

Komponente	Beschreibung
	Der Dreh- und Angelpunkt der Druckausgabe. Über dieses Objekt bestimmen Sie den gewünschten Drucker, die Papierausrichtung, die Auflösung, die zu druckenden Seiten usw. Über das *PrintPage*-Ereignis erhalten Sie Zugriff auf das *Graphics*-Objekt des Druckers. Weiterhin steuern Sie hier den Druckverlauf (Anzahl der Seiten, Seitenauswahl, Abbruch). Mehr zu dieser Komponente finden Sie in den beiden folgenden Abschnitten.
	Der Windows-Standarddialog zur Auswahl eines Druckers sowie der wichtigsten Druckparameter (Seiten, Exemplare, Auflösung)
	Der Windows-Standarddialog zur Konfiguration der Druckausgabe (Seitenausrichtung, Papierausrichtung, Seitenränder)
	Eine komplette Druckvorschau, mit Navigationstasten, Zoom etc.
	Eine Alternative für den *PrintPreviewDialog*. Bei dieser Komponente ist lediglich der Preview-Bereich vorhanden, für die Ansteuerung und Konfiguration sind Sie selbst verantwortlich.

Alle Komponenten können über die *Document*-Eigenschaft mit der *PrintDocument*-Komponente verknüpft werden, Sie müssen also die Parameter nicht "von Hand" übergeben.

7.3 Auswerten der aktuellen Druckereinstellungen

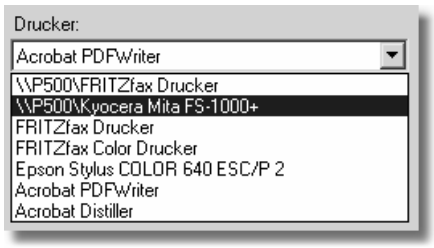
In den folgenden Abschnitten wollen wir versuchen, Ihnen die "Vorzüge" der selten unübersichtlichen Objektstruktur zu ersparen. Aus diesem Grund werden wir auf eine sinnlose Auflistung von Eigenschaften und Methoden für die einzelnen Objekte verzichten, stattdessen stellen wir die zu lösende Aufgabe in den Vordergrund.

7.3.1 Die vorhandenen Drucker

Einen Überblick, welche Drucker auf dem aktuellen System installiert sind, können Sie sich über die Collection *InstalledPrinters* verschaffen.

Beispiel: Ausgabe aller Druckernamen in einer Combobox und markieren des aktiven Druckers

```
Dim Printername As String
Dim doc As New Drawing.Printing.PrintDocument()

' Füllen der Listbox
For Each Printername In Drawing.Printing.PrinterSettings.InstalledPrinters
   ComboBox1.Items.Add(Printername)
Next
' Auswahl des aktiven Druckers
ComboBox1.Text = doc.PrinterSettings.PrinterName
```



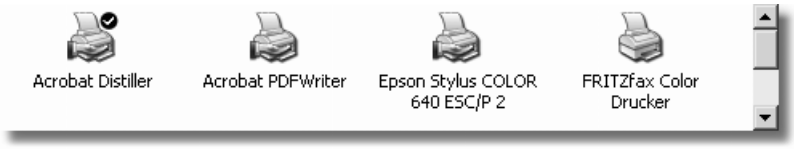
7.3.2 Der Standarddrucker

Möchten Sie überprüfen, ob der aktuell gewählte Drucker gleichzeitig auch der Systemstandarddrucker ist, können Sie dies mit Hilfe der Eigenschaft *IsDefaultPrinter* realisieren.

Beispiel:

```
If PrintDocument1.PrinterSettings.IsDefaultPrinter Then
  ...
```

Hinweis: Den Standarddrucker erkennen Sie in der Systemsteuerung an einem kleinen Häkchen.



7.3.3 Verfügbare Papierformate/Seitenabmessungen

Geht es um die Abfrage, welche Papierarten der Drucker unterstützt, können Sie einen Blick auf die *Papersizes*-Collection werfen. Diese gibt Ihnen nicht nur Auskunft über die Blattgröße (*Height, Width*), sondern auch über die Blattbezeichnung (*PaperName*) und den Typ (*Kind*).

Beispiel: Anzeige aller Papierformate im Ausgabe-Fenster

```
Private Sub Button6_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
            Handles Button6.Click
Dim ps As System.Drawing.Printing.PaperSize

  For Each ps In PrintDocument1.PrinterSettings().PaperSizes
     Debug.WriteLine(ps)
  Next

End Sub
```

Die Anzeige im Ausgabe-Fenster:

Hinweis: Die Blattabmessungen werden in 1/100 Zoll zurückgegeben! Der Umrechnungsfaktor in Millimetern ist 0,254.

Beispiel: Anzeige der aktuellen Blattabmessungen in Millimetern

```
Debug.WriteLine(PrintDocument1.PrinterSettings().DefaultPageSettings.PaperSize().Height * _
    0.254)
Debug.WriteLine(PrintDocument1.PrinterSettings().DefaultPageSettings.PaperSize().Width * _
    0.254)
```

Gleichzeitig steht Ihnen mit *System.Drawing.Printing.PaperKind* eine Aufzählung der Standardpapierformate zur Verfügung (Auszug):

Element	Beschreibung
A2	A2 (420 x 594 mm)
A3	A3 (297 x 420 mm)
A3Extra	A3 Extra (322 x 445 mm)
A3ExtraTransverse	A3 Extra quer (322 x 445 mm)
A3Rotated	A3 gedreht (420 x 297 mm)
A3Transverse	A3 quer (297 x 420 mm)
A4	A4 (210 x 297 mm)

7.3.4 Der eigentliche Druckbereich

Leider druckt nicht jeder Drucker bis zu den Blatträndern, was den MS-Programmierern wohl verborgen geblieben ist. Die Autoren konnten keine Eigenschaft/Methode finden, mit der sich der eigentliche Druckbereich und insbesondere der Offset des Druckbereichs bestimmen ließe.

Hinweis: Vergessen Sie in diesem Zusammenhang die Eigenschaft *Margins* ganz schnell wieder, es handelt sich lediglich um theoretische Seitenränder, die Sie selbst definieren können.

Wie fast immer üblich, kommen Sie mit GDI-Programmierung weiter als mit GDI+.

Beispiel: Ausgabe des bedruckbaren Blattbereichs sowie der physikalischen Seitenränder im Ausgabefenster.

Binden Sie zunächst die GDI-Funktion *GetDeviceCaps* sowie einige Konstanten ein:

```
Private Declare Function GetDeviceCaps Lib "gdi32" Alias "GetDeviceCaps" _
       (ByVal hdc As IntPtr, ByVal nIndex As Integer) As Integer

Private Const PHYSICALOFFSETX = 112
Private Const PHYSICALOFFSETY = 113
```

```
Private Const HORZSIZE = 4
Private Const VERTSIZE = 6
Private Const LOGPIXELSX = 88
Private Const LOGPIXELSY = 90
```

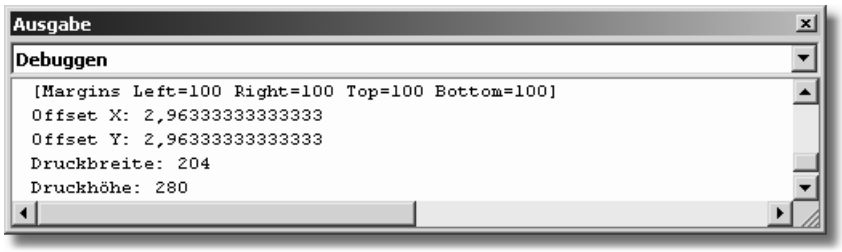
Das nächste Problem: Woher bekommen wir einen DC für die GDI-Funktion? Ein *Graphics*-Objekt, mit dem sich ein DC erzeugen lässt, steht uns zu diesem Zeitpunkt nicht zur Verfügung.

Doch ein aufmerksamer Blick in die Online-Hilfe verrät uns, dass die Methode *CreateMeasurementGraphics* doch ein *Graphics*-Objekt zurückgibt, das vom konzeptionellen Ansatz her einem Informationsgerätekontext entspricht. Der mit der Methode *GetDC* erzeugte DC genügt uns für die Abfrage von Eigenschaften des aktuellen Druckers:

```
Dim g As Graphics
Dim dc As IntPtr

g = PrintDocument1.PrinterSettings.CreateMeasurementGraphics()
dc = g.GetHdc

Debug.WriteLine("Offset X: " & GetDeviceCaps(dc, PHYSICALOFFSETX) * 25.4 / _
      GetDeviceCaps(dc, LOGPIXELSX))
Debug.WriteLine("Offset Y: " & GetDeviceCaps(dc, PHYSICALOFFSETY) * 25.4 / _
      GetDeviceCaps(dc, LOGPIXELSY))
Debug.WriteLine("Druckbreite: " & GetDeviceCaps(dc, HORZSIZE))
Debug.WriteLine("Druckhöhe: " & GetDeviceCaps(dc, VERTSIZE))

g.ReleaseHdc(dc)
```

Hinweis: Die etwas umständliche Rechnerei ist dem Maßsystem unserer amerikanischen Freunde geschuldet.

Die Anzeige im Ausgabe-Fenster:



7.3.5 Seitenausrichtung ermitteln

Die aktuelle Blatt- bzw. Seitenausrichtung können Sie über die Eigenschaft *Landscape* abfragen.

Beispiel:

```
If PrintDocument1.PrinterSettings.DefaultPageSettings.Landscape Then ' wenn Querformat
  ...
```

7.3.6 Ermitteln der Farbfähigkeit

Ob Ihr aktuell gewählter Drucker auch in der Lage ist, mehr als nur Schwarz zu Papier zu bringen, lässt sich mit der Eigenschaft *SupportsColor* ermitteln.

Beispiel:

```
If PrintDocument1.PrinterSettings.SupportsColor Then
  ...
```

7.3.7 Die Druckauflösung abfragen

Möchten Sie sich über die physikalische Druckauflösung des aktiven Druckers informieren, sollten Sie sich mit der Eigenschaft *PrinterResolution* näher beschäftigen.

Beispiel: Die horizontale Druckauflösung (readonly):

```
Debug.WriteLine(PrintDocument1.DefaultPageSettings.PrinterResolution.X)
```

Beispiel: Die vertikale Druckauflösung (readonly):

```
Debug.WriteLine(PrintDocument1.DefaultPageSettings.PrinterResolution.Y)
```

Hinweis: Die Rückgabewerte entsprechen Punkten pro Zoll (Dots per Inch: dpi).

Alternativ können Sie über die *Kind*-Eigenschaft einen der folgenden Werte abrufen:

Kind	Beschreibung
Custom	Benutzerdefinierte Auflösung
Draft	Auflösung in Entwurfsqualität
High	Hohe Auflösung
Low	Niedrige Auflösung
Medium	Mittlere Auflösung

Beispiel:

```
Debug.WriteLine(PrintDocument1.DefaultPageSettings.PrinterResolution.Kind())
```

7.3.8 Ist beidseitiger Druck möglich?

Ob der Drucker duplexfähig ist, d.h., ob er beidseitig drucken kann, ermitteln Sie über die Eigenschaft *CanDuplex*.

Beispiel:

```
If PrintDocument1.PrinterSettings.CanDuplex Then
   ...
```

7.3.9 Einen "Informationsgerätekontext" erzeugen

Wer bisher mit GDI-Funktionen gearbeitet hat, dem wird auch der Begriff "Informationsgerätekontext" sicher nicht unbekannt sein. Der Hintergrund: Bei einem Drucker wird für die Abfrage von Gerätemerkmalen (Auflösung, Seitenränder etc.) häufig ein DC benötigt, der zum Beispiel im Zusammenhang mit der Funktion *GetDeviceCaps* genutzt wird. Dieses DC ist nur für die Abfrage von Werten vorgesehen.

Den DC selbst erhalten Sie nur über einen kleinen Umweg: Mit Hilfe der Methode *CreateMeasurementGraphics* erzeugen Sie zunächst ein *Graphics*-Objekt, und dieses stellt bekanntlich die Methode *GetDC* zur Verfügung.

Beispiel: Erzeugen eines Informationsgerätekontext

```
Dim g As Graphics
Dim dc As IntPtr
g = PrintDocument1.PrinterSettings.CreateMeasurementGraphics()
dc = g.GetHdc

' z.B. Abfrage der Blattmaße
Debug.WriteLine("Druckbreite: " & GetDeviceCaps(dc, HORZSIZE))
Debug.WriteLine("Druckhöhe: " & GetDeviceCaps(dc, VERTSIZE))
g.ReleaseHdc(dc)
```

Hinweis: Vergessen Sie nicht, den DC wieder freizugeben. Dies muss innerhalb der aktuellen Ereignisroutine geschehen, Sie können den Wert **nicht** in einer globalen Variablen speichern!

7.3.10 Abfragen von Werten während des Drucks

Statt wie in den vorhergehenden Beispielen mit der *PrintDocument*-Komponente nutzen Sie besser den im *PrintPage*-Ereignis angebotenen Parameter *e*. Über diesen erhalten Sie Zugriff auf die gewünschten Eigenschaften.

Beispiel:

```
Private Sub PrintDocument1_PrintPage(ByVal sender As System.Object, _
  ByVal e As System.Drawing.Printing.PrintPageEventArgs) Handles PrintDocument1.PrintPage

Debug.WriteLine(e.PageSettings.PaperSize)
```

7.4 Festlegen von Druckereinstellungen

Nachdem wir im vorhergehenden Abschnitt recht passiv mit den Druckeroptionen umgegangen sind und uns auf das reine Auslesen beschränkt haben, wollen wir uns im Weiteren um das Konfigurieren des Druckers kümmern.

7.4.1 Einen Drucker auswählen

Der wohl erste Schritt, wenn mehr als ein Drucker zur Verfügung steht, ist die Auswahl des Druckers. Zwei Varianten bieten sich an:

- Verwendung der Eigenschaft *PrinterName*
- Verwendung der *PrintDialog*-Komponente (siehe Abschnitt 7.5)

Hinweis: Nach dem Setzen der Eigenschaft bzw. vor dem endgültigen Drucken sollten Sie mit der Eigenschaft *IsValid* überprüfen, ob die Konfiguration auch realisierbar ist.

Beispiel: Setzen der *PrinterName*-Eigenschaft und nachfolgende Prüfung mit *IsValid*

```
PrintDocument1.PrinterSettings.PrinterName = ComboBox1.Text
If PrintDocument1.PrinterSettings.IsValid Then PrintPreviewDialog1.ShowDialog()
```

Beispiel: Verwendung des *QueryPageSettings*-Ereignisses zur Auswahl eines Druckers

```
Private Sub PrintDocument1_QueryPageSettings(ByVal sender As Object, _
   ByVal e As System.Drawing.Printing.QueryPageSettingsEventArgs) Handles _
   PrintDocument1.QueryPageSettings
   e.PageSettings.PrinterSettings.PrinterName = "FRITZFax Drucker"
End Sub
```

7.4.2 Drucken in Millimetern

Sie werden hoffentlich nicht auf die Idee kommen, Zeichnungen in Pixeln auf dem Drucker auszugeben, je nach Modell ist sonst Ihre Grafik mikroskopisch klein oder riesengroß. Bleibt die Frage, wie Sie die Maßeinheit auf Millimeter umstellen können. Die Lösung ist schnell gefunden, über die Eigenschaft *PageUnit* können Sie eine der folgenden Maßeinheiten auswählen:

Konstante	Beschreibung
Display	Eine Einheit entspricht 1/75 Zoll.
Document	Eine Einheit entspricht 1/300 Zoll.
Inch	Eine Einheit entspricht 1 Zoll.
Millimeter	Eine Einheit entspricht einem Millimeter.
Pixel	Eine Einheit entspricht einem Gerätepixel.
Point	Eine Einheit entspricht 1/72 Zoll (Point).

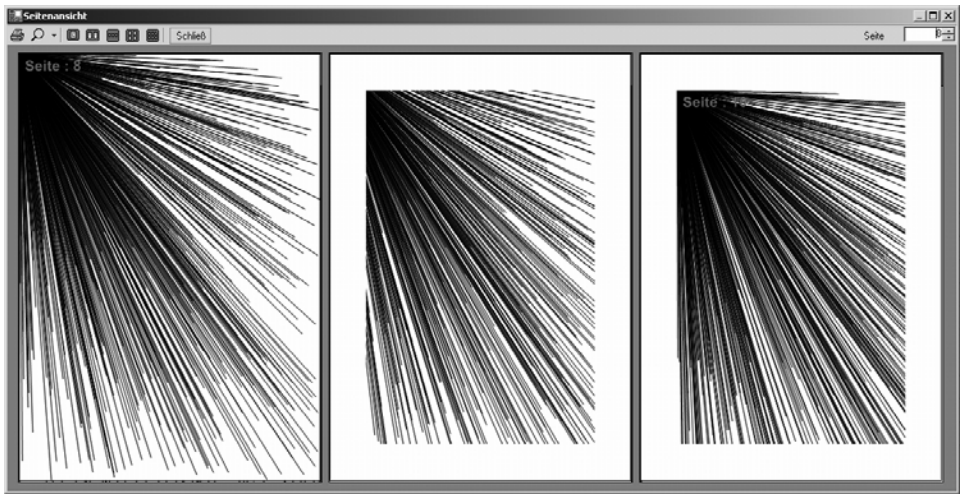
Beispiel: Setzen der Maßeinheit im *PrintPage*-Ereignis

```
Private Sub PrintDocument1_PrintPage(ByVal sender As System.Object, _
        ByVal e As System.Drawing.Printing.PrintPageEventArgs) _
        Handles PrintDocument1.PrintPage

e.Graphics.PageUnit = GraphicsUnit.Millimeter
e.Graphics.DrawLine(New Pen(Color.Black, 10), 50, 100, 150, 200)
...
```

7.4.3 Festlegen der Seitenränder

Tja, welche Ränder meinen Sie denn? Geht es um Seitenränder wie zum Beispiel in MS Word, nutzen Sie die Eigenschaft *Margins*. Allerdings bedeutet das Festlegen per Code oder mit Hilfe der Dialogbox *PageSetupDialog* noch lange nicht, dass diese Ränder auch zwingend eingehalten werden. Dafür sind Sie im *PrintPage*-Ereignis selbst verantwortlich.

Die folgende Abbildung soll Ihnen die Problematik verdeutlichen. In jedem der drei Fälle werden, beginnend mit der Koordinate 0,0 (linke obere Ecke), Zufallslinien gezeichnet, die maximal die Abmessungen des Blattes erreichen.



Beispiel 1

```
e.Graphics.PageUnit = GraphicsUnit.Display
For i = 0 To 500
  e.Graphics.DrawLine(p, 0, 0, CInt(Int((e.PageBounds.Width * Rnd()) + 1)), _
                               CInt(Int((e.PageBounds.Height * Rnd()) + 1)))
Next
```

zeigt deutlich, dass die eingestellten Seitenränder (100,100,100,100) vollkommen ignoriert werden.

Beispiel 2 berücksichtigt bereits die eingestellten Seitenränder durch die Verwendung eines Clipping-Bereichs:

```
e.Graphics.PageUnit = GraphicsUnit.Display
e.Graphics.SetClip(e.MarginBounds)
For i = 0 To 500
   e.Graphics.DrawLine(p, 0, 0, CInt(Int((e.PageBounds.Width * Rnd()) + 1)), _
                                CInt(Int((e.PageBounds.Height * Rnd()) + 1)))
Next
```

Beispiel 3 bringt auch den Koordinatenursprung an die richtige Position:

```
e.Graphics.PageUnit = GraphicsUnit.Display
e.Graphics.SetClip(e.MarginBounds)
e.Graphics.TranslateTransform(e.MarginBounds.Left, e.MarginBounds.Top)
For i = 0 To 500
   e.Graphics.DrawLine(p, 0, 0, CInt(Int((e.PageBounds.Width * Rnd()) + 1)), _
                                CInt(Int((e.PageBounds.Height * Rnd()) + 1)))
Next
```

Damit brauchen Sie sich beim Zeichnen eigentlich nur noch um die Breite und Höhe des bedruckbaren Bereichs (*e.MarginBounds.Width* bzw. *e.MarginBounds.Height*) zu kümmern, die linke obere Ecke ist bereits korrekt gesetzt.

Hinweis: Die Eigenschaft *Margins* hebt natürlich keine physikalischen Grenzen auf. Wenn der Drucker einen entsprechenden Offset aufweist, müssen Sie diesen auch berücksichtigen (siehe Abschnitt 7.3.4).

7.4.4 Druckjobname

Was im Normalfall eher sekundär ist, kann in Netzwerkumgebungen bzw. Multiuser-Umgebungen für mehr Übersicht sorgen. Über die Eigenschaft *DocumentName* können Sie vor dem Drucken einen aussagekräftigen Druckjobnamen festlegen, der im Druckerspooler angezeigt wird.

Beispiel:

```
PrintDocument1.DocumentName = "Mein erster VB.NET-Druckversuch"
```

7.4.5 Anzahl der Kopien

Die Anzahl der Druckkopien kann zum einen mit Hilfe des Dialogs *PrintDialog*, zum anderen auch per Code festgelegt werden. Nutzen Sie die Eigenschaft *Copies*.

Beispiel:

```
PrintDocument1.PrinterSettings.Copies = 3
```

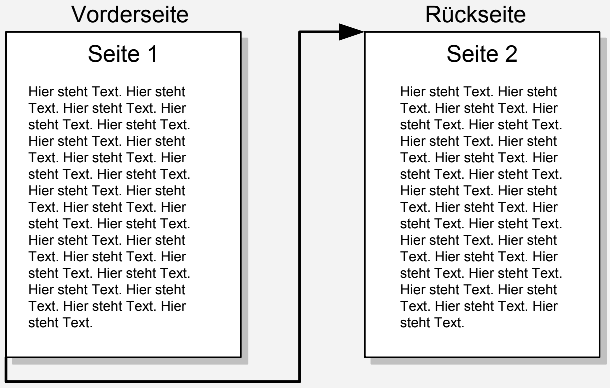
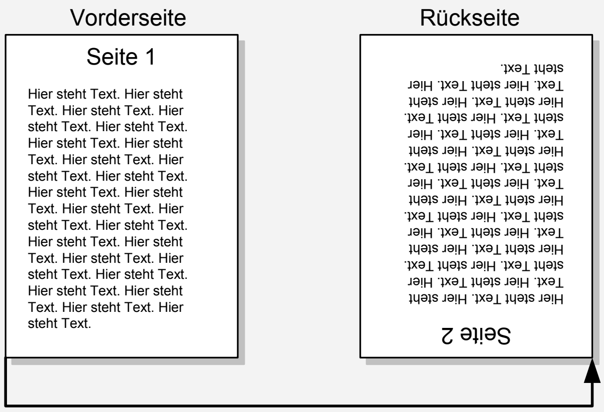
Hinweis: Mit *MaximumCopies* können Sie einen Maximalwert für die Druckdialoge vorgeben!

Beispiel:

```
PrintDocument1.PrinterSettings.MaximumCopies = 5
```

7.4.6 Beidseitiger Druck

Geht es um das beidseitige Bedrucken von Papier, was aus ökologischer Sicht sicher sinnvoller ist, müssen Sie sich zunächst vergewissern, dass der Drucker auch über dieses Feature verfügt (siehe Abschnitt 7.3.8). Nachfolgend können Sie über die *Duplex*-Eigenschaft den gewünschten Wert einstellen.

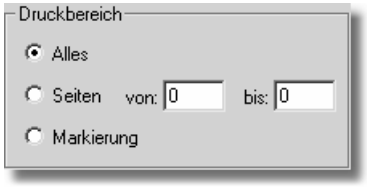
Konstante	Beschreibung
Default	Die Standardeinstellungen des Druckers werden genutzt.
Simplex	Der "normale" einseitige Druck.
Horizontal	
Vertical	

Beispiel: Einstellen der *Duplex*-Eigenschaft

```
PrintDocument1.PrinterSettings.Duplex = Drawing.Printing.Duplex.Horizontal
```

7.4.7 Seitenzahlen festlegen

Die Überschrift dürfte auf den ersten Blick etwas missverständlich klingen, da Sie doch selbst über den zu druckenden Inhalt entscheiden. Wenn Sie sich jedoch an den Druckerdialog erinnern, sind dort auch Optionen für die Seitenauswahl möglich:



Wie bereits angekündigt, ist gerade diese Option ein nicht ganz leicht verdaulicher Brocken.

Zunächst einmal unterscheiden Sie die drei gewählten Optionen (Alles, Seiten und Markierung) mit Hilfe der folgenden Konstanten über die *PrintRange*-Eigenschaft.

Konstante	Beschreibung
AllPages	Alle Seiten drucken.
Selection	Die ausgewählte Seite drucken (was auch immer das sein mag).
SomePages	Die Seiten zwischen *FromPage* und *ToPage* sollen gedruckt werden.

Beispiel: Berücksichtigung des vorgegebenen Druckbereichs

Zunächst wird eine globale Variable *Page* im *BeginPrint*-Ereignis initialisiert

```
Private Sub PrintDocument1_BeginPrint(ByVal sender As Object, _
           ByVal e As System.Drawing.Printing.PrintEventArgs) Handles _
           PrintDocument1.BeginPrint
  page = 1
End Sub
```

Die Hauptarbeit erwartet uns im *PrintPage*-Ereignis:

```
Private Sub PrintDocument1_PrintPage(ByVal sender As System.Object, _
   ByVal e As System.Drawing.Printing.PrintPageEventArgs) Handles PrintDocument1.PrintPage
```

Eine Variable, die später die Nummer der zu druckenden Seite enthält:

```
Dim printpage As Integer
```

Die Berücksichtigung des Druckbereichs:

```
Select Case e.PageSettings.PrinterSettings.PrintRange
   Case Drawing.Printing.PrintRange.SomePages
       printpage = page + e.PageSettings.PrinterSettings.FromPage - 1
   Case Drawing.Printing.PrintRange.AllPages
       printpage = page
   Case Drawing.Printing.PrintRange.Selection
End Select
```

Wie Sie sehen, wird die aktuell zu druckende Seite je nach *PrintRange* bestimmt.

Im weiteren Verlauf können Sie dann zum Beispiel mit *Select Case* die jeweilige Seite aufbereiten:

```
Select Case printpage
   Case 1
      e.Graphics.FillRectangle(New SolidBrush(Color.Blue), 30, 30, 100, 100)
   Case 2
      e.Graphics.DrawLine(New Pen(Color.Black, 10), 50, 100, 150, 200)
...
```

Doch auch die Feststellung, ob weitere Seiten zu drucken sind, muss jetzt unter Berücksichtigung des Druckbereichs getroffen werden:

```
...
page += 1

' Berücksichtigung des Druckbereichs
Select Case e.PageSettings.PrinterSettings.PrintRange
  Case Drawing.Printing.PrintRange.SomePages
     e.HasMorePages = (printpage < e.PageSettings.PrinterSettings.ToPage)
  Case Drawing.Printing.PrintRange.AllPages
     e.HasMorePages = page < 10
  Case Drawing.Printing.PrintRange.Selection
End Select
...
```

Noch einmal zusammengefasst: Ist *PrintRange* auf *SomePages* eingestellt, drucken Sie statt der ersten Seite gleich den Inhalt der mit *FromPage* festgelegten Seite. Der Druckvorgang wird so lange wiederholt, bis *ToPage* erreicht ist.

Hinweis: Über die Eigenschaften *MinimumPage* und *MaximumPage* können Sie maximale Grenzen für die Auswahl des Druckbereichs festlegen.

Beispiel: Druckbereich maximal von Seite 1 bis Seite 10

```
PrintDocument1.PrinterSettings.MinimumPage = 1
PrintDocument1.PrinterSettings.MaximumPage = 10
```

7.4.8 Druckqualität

Unter diesem Punkt verstehen wir zum einen die Einstellung der dpi-Zahl des Druckers, zum anderen die Optionen bei der Ausgabe von Grafiken (*Antialiasing, CompositingQuality*).

Beispiel: Setzen der Druckauflösung (es wird die zweite verfügbare Auflösung verwendet)

```
PrintDocument1.DefaultPageSettings.PrinterResolution = _
  PrintDocument1.PrinterSettings.PrinterResolutions(2)
```

Beispiel: Verbessern der Textausgabequalität

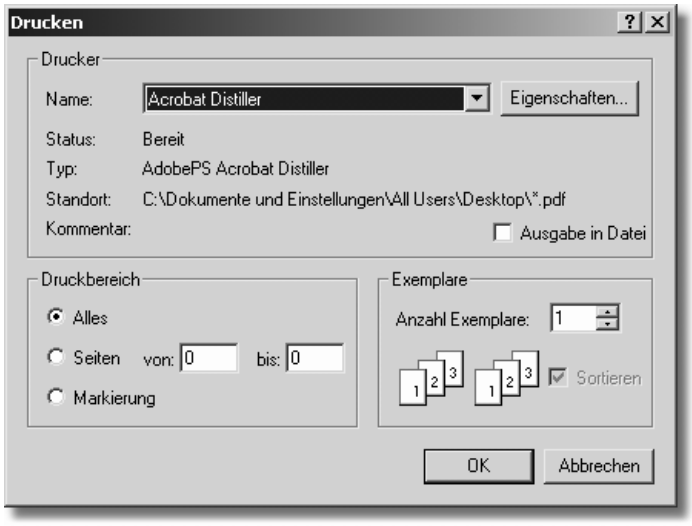
```
e.Graphics.TextRenderingHint = _
   Drawing.Text.TextRenderingHint.AntiAlias
```

7.5 Die Druckdialoge

Im Folgenden wollen wir Ihnen kurz die drei wesentlichen Druckdialoge und deren wichtigste Parameter im Zusammenspiel mit der Druckausgabe vorstellen.

7.5.1 PrintDialog

Der wohl jedem bekannte Standarddruckdialog wird mit der Komponente *PrintDialog* eingebunden.



Die Komponente selbst können Sie mittels *Document*-Eigenschaft direkt an eine *PrintDocument*-Komponente binden. Alle gewählten Parameter werden automatisch an *PrintDocument* übergeben.

Eigenschaft	Beschreibung
AllowPrintToFile	... aktiviert das Kontrollkästchen "Ausgabe in Datei".
AllowSelection	... aktiviert das Optionsfeld "Seiten von ... bis ... ".

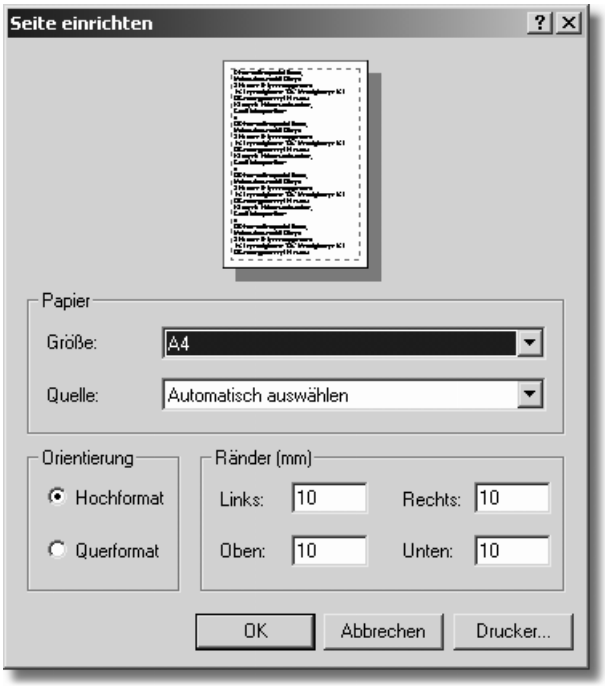
Eigenschaft	Beschreibung
AllowSomePages	... aktiviert das Optionsfeld "Seiten".
ShowHelp	... aktiviert die Schaltfläche "Hilfe".
ShowNetwork	... aktiviert die Schaltfläche "Netzwerk" (nur in der Theorie).
PrinterSettings	Über diese Eigenschaft können Sie Standardwerte vorgeben sowie die Einstellungen des Dialogfelds abfragen.
PrintToFile	... fragt den Wert des Kontrollkästchens "Ausgabe in Datei" ab.

Beispiel: Anzeige des Dialogs und Abfrage des gewählten Druckers

```
If PrintDialog1.ShowDialog = DialogResult.OK Then
    MessageBox.Show(PrintDialog1.PrinterSettings.PrinterName, "Hinweis", _
                    MessageBoxButtons.OK, MessageBoxIcon.Exclamation)
End If
```

7.5.2 PageSetupDialog

Neu für den Visual Basic-Programmierer dürfte der folgende Dialog sein, mit dem Sie einen Menüpunkt "Seite einrichten" realisieren können.



Auch diese *PageSetupDialog*-Komponente können Sie mittels *Document*-Eigenschaft direkt an eine *PrintDocument*-Komponente binden, um die eingestellten Parameter automatisch zu übernehmen.

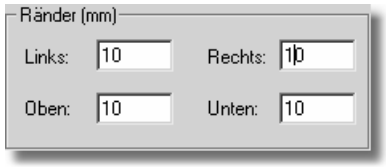
Eigenschaft	Beschreibung
AllowMargins	... aktiviert den Bereich "Ränder (mm)".
AllowOrientation	... aktiviert den Bereich "Orientierung".
AllowPaper	... aktiviert den Bereich "Papier".
AllowPrinter	... aktiviert die Schaltfläche "Drucker...".
ShowHelp	... aktiviert die Schaltfläche "Hilfe".
MinMargins	... legt die minimalen Werte für die Ränder fest.
PageSettings *PrinterSettings*	Über diese Eigenschaften können Sie Standardwerte vorgeben bzw. die geänderten Werte abfragen.

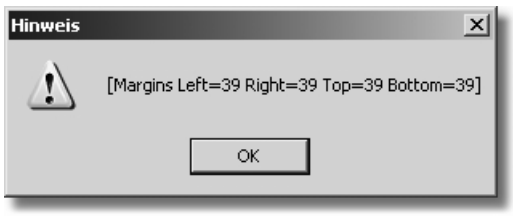
Hinweis: Über das Ereignis *HelpRequest* können Sie auf den Button "Hilfe" reagieren!

Beispiel: Aufruf der Dialogbox und Anzeige der neu gesetzten Ränder

```
If PageSetupDialog1.ShowDialog = DialogResult.OK Then
   MessageBox.Show(PageSetupDialog1.PageSettings.Margins.ToString, "Hinweis", _
               MessageBoxButtons.OK, MessageBoxIcon.Exclamation)
End If
```

Das Ergebnis:





Probleme mit den Rändern

Doch wo viel Licht, da ist auch Schatten, ein kleiner Bug hat sich in die Komponente eingeschlichen, der wahrscheinlich nur in den lokalisierten Varianten von VS.NET auftritt:

Hinweis: Die Werte der eingestellten Ränder stimmen nicht mit den Werten der Eigenschaft *Margins* überein (aus einem Zoll Vorgabewert werden in der Anzeige 10 Millimeter und aus diesen wiederum korrekte 0,39 Zoll). Eine fragwürdige Umrechnung.

Deshalb der folgende Workaround:

Rufen Sie **vor** dem Aufruf der Dialogbox jedes Mal die folgenden Anweisungen auf.

```
PageSetupDialog1.PageSettings.Margins.Left *= 2.54
PageSetupDialog1.PageSettings.Margins.Top *= 2.54
PageSetupDialog1.PageSettings.Margins.Right *= 2.54
PageSetupDialog1.PageSettings.Margins.Bottom *= 2.54
PageSetupDialog1.ShowDialog
```

Nach dem Aufruf stehen Ihnen die Seitenränder wieder in der korrekten 1/100-Zoll-Angabe zur Verfügung. Beachten Sie jedoch, dass mit eventuellen Service-Packs dieses Problem zwischenzeitlich von Microsoft korrigiert werden könnte.

7.6 PrintPreviewDialog

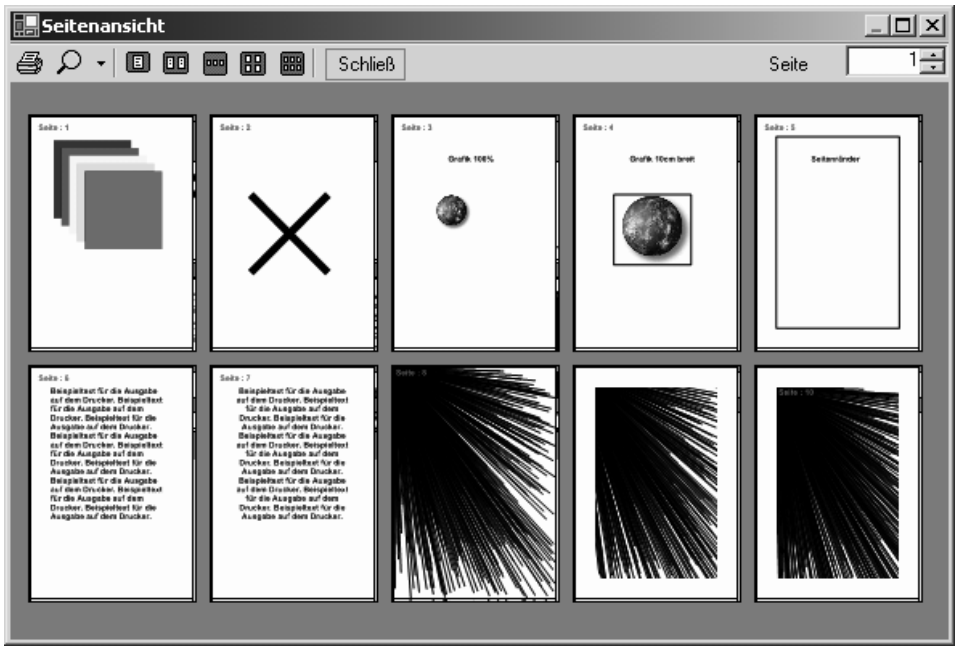
Im Grunde ist die Verwendung des *PrintPreviewDialog* recht simpel. Nach dem Einfügen der Komponente brauchen Sie diese lediglich über die *Documents*-Eigenschaft mit der *PrintDocument*-Komponente verknüpfen und die Methode *ShowDialog* aufrufen.

Bis auf die Eigenschaft *UseAntialias* (Verbessern der Anzeigequalität) können Sie kaum weitere Einstellungen vornehmen. Die Ausnahme stellt die Eigenschaft *PrintPreviewControl* dar, mit der Sie direkt das Aussehen und Verhalten der Vorschau beeinflussen können.

Beispiel: Gleichzeitige Anzeige von zehn Seiten

```
PrintPreviewDialog1.PrintPreviewControl.Rows = 2
PrintPreviewDialog1.PrintPreviewControl.Columns = 5
PrintPreviewDialog1.ShowDialog()
```

Hinweis: Mehr zur Konfiguration und Verwendung der *PrintPreviewControl*-Komponente finden Sie im Abschnitt 7.7.



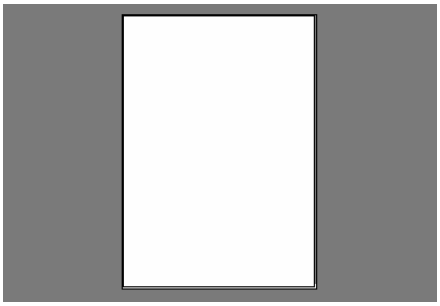
Weiterhin dürfte die Eigenschaft *WindowState* in Zusammenhang mit der Anzeige der Dialogbox von Interesse sein. Hiermit steuern Sie die Art der Anzeige (Vollbild etc.).

Beispiel: Vollbildanzeige aktivieren

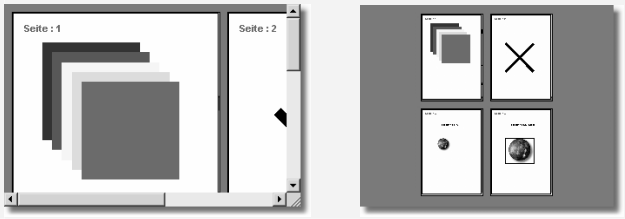
```
PrintPreviewDialog1.WindowState = FormWindowState.Maximized
PrintPreviewDialog1.ShowDialog()
```

7.7 Ein eigenes Druckvorschau-Fenster realisieren

Wem die *PrintPreview*-Komponente zu wenig Eingriffsmöglichkeiten bietet, der kann sich mit der *PrintPreviewControl*-Komponente eine eigene Druckvorschau zusammenbauen.



Bis auf den reinen Druckvorschaubereich können Sie sich um alle Einstellungen und optischen Spielereien selbst kümmern. Die folgende Tabelle listet die wichtigsten Eigenschaften auf:

Eigenschaft	Beschreibung
AutoZoom	Ist der Wert auf *True* gesetzt, werden die Seiten so skaliert, das die vorgegebene Anzahl von Seiten flächenfüllend dargestellt wird. *AutoZoom = False* *AutoZoom = True* 
BackColor	… die Hintergrundfarbe für die Druckvorschau.
Columns	... die Anzahl von Spalten, d.h., wie viele Seiten nebeneinander dargestellt werden.
Rows	... die Anzahl von Zeilen, d.h., wie viele Seiten untereinander dargestellt werden.
Document	... die Verknüpfung zum *PrintDocument*-Objekt.
StartPage	... die Seitenzahl der linken oberen Seite. Durch Verändern dieses Wertes können Sie die weiteren Seiten anzeigen.
Zoom	... legt explizit einen Zoomfaktor fest.

Wie Sie die *PrintPreviewControl*-Komponente im Zusammenhang verwenden, zeigt Ihnen das folgende Rezept:

☞ R30 … den Drucker konfigurieren?

7.8 Drucken mit OLE-Automation

Die Überschrift deutet es schon an, wir wollen versuchen, mit Hilfe der bekannten Office-Programme Druckausgaben zu realisieren. Schnell kommt der Verdacht auf, dass der Programmierer versucht, das Brett an der dünnsten Stelle anbohren zu wollen. Doch warum sollen nicht die Möglichkeiten von Office-Programmen genutzt werden, wenn doch häufig der Wunsch besteht, Reportausgaben nachträglich zu bearbeiten oder in umfangreichere Dokumentationen aufzunehmen? Nicht zuletzt bieten sich gerade die Office-Anwendungen an, wenn es um eine sinnvolle Archivierung von Dokumenten geht.

Hinweis: Die "alte" OLE-Container-Komponente steht in VB.NET nicht mehr zur Verfügung.

Unser Favorit ist, wie sollte es auch anders sein, Microsoft Word, ein Allround-Talent, was die Gestaltung von ansprechenden Druckausgaben angeht. Als zweite Variante bietet sich insbesondere bei der Verwendung von Desktop-Datenbanken Microsoft Access an. Hier kann der integrierte Report-Generator zeigen, was er kann.

Für die im Weiteren vorgestellten Verfahren ist es sinnvoll, wenn wir kurz auf die grundlegenden Möglichkeiten und Funktionen der OLE-Automation eingehen.

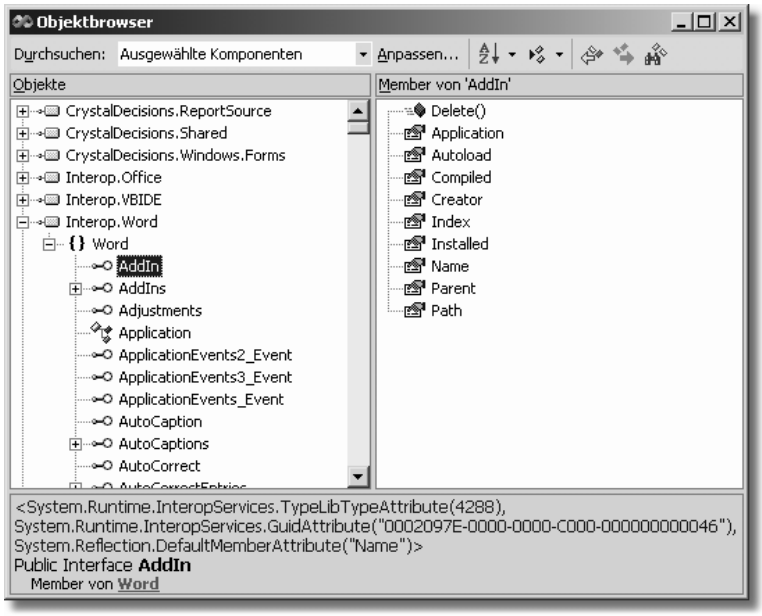
Hinweis: Ein Nachteil der im Folgenden beschriebenen Verfahrensweise soll natürlich nicht unerwähnt bleiben. Geben Sie Ihre Anwendungen an andere Anwender weiter, muss auf dem jeweiligen Rechner natürlich auch die OLE-Anwendung (Access oder Word) installiert sein.

7.8.1 Kurzeinstieg in die OLE-Automation

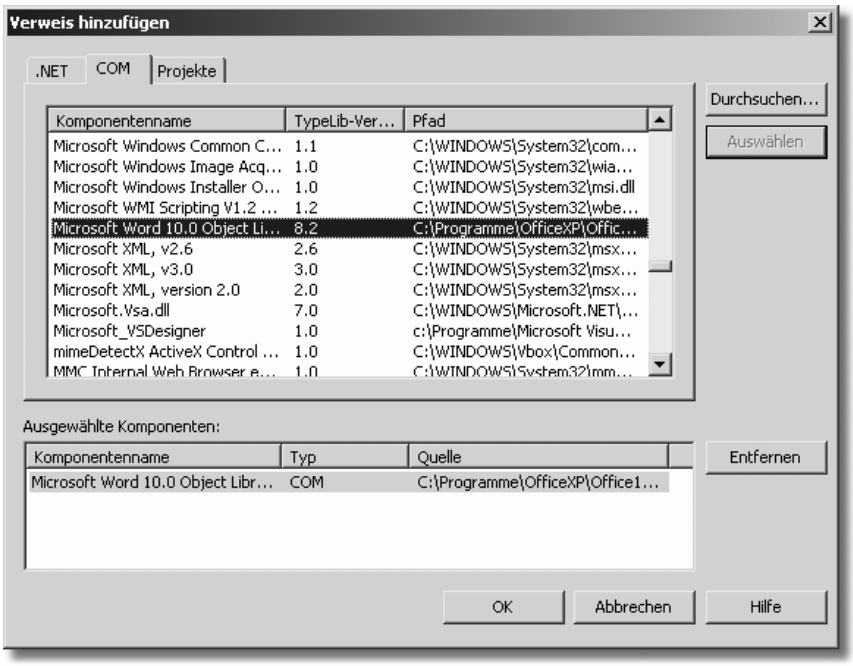
Über OLE-Automation lassen sich Objekte anderer Applikationen (z.B. Word oder Access) von Ihrem VB.NET-Programm quasi fernsteuern. Nach der Definition einer entsprechenden Objektvariablen können Sie auf Eigenschaften und Methoden dieser Objekte genauso zugreifen, als ob es sich um ein Visual Basic-Objekt handeln würde.

Wichtigstes Hilfsmittel für den OLE-Programmierer ist der in VB.NET integrierte Objekt-Browser (siehe folgende Abbildung).

Im Objekt-Browsers werden neben den Klassen alle Methoden, Eigenschaften, Ereignisse und Konstanten des jeweiligen Objekts angezeigt:



Welche Objekte angezeigt werden, hängt von den Verweisen ab, die Sie unter *Projekt/Verweise hinzufügen...* eingebunden haben:



Programmieren der OLE-Automation

Das Grundprinzip besteht darin, dass Sie in VB.NET eine Instanz der gewünschten Klasse erzeugen. Mit diesem Objekt können Sie dann wie mit jedem anderen VB-Objekt arbeiten.

Zum Erzeugen der Instanz bieten sich drei Möglichkeiten an:

- *GetObject*-Funktion
- *CreateObject*-Funktion
- *New*

New

Voraussetzung für die Verwendung von *New* ist ein Verweis auf die entsprechende Klasse. Um neue Verweise zu erstellen, müssen Sie unter *Projekt/Verweise hinzufügen...* die gewünschte Klassenbibliothek auswählen oder eine neue hinzufügen.

Beispiel: Erstellen einer Objektvariablen *Word* als Instanz des *Word.Application*-Objekts.

```
Dim Word As New Word.Application
```

oder

```
Dim Word As Word.Application
...
Word = New Word.Application
```

Je nach Definition wird das Objekt beim ersten Aufruf einer Methode oder Eigenschaft (... *As New*) oder schon beim Zuweisen (*Set ... = New*) erstellt.

GetObject und CreateObject

Grundsätzlich sollten Sie immer versuchen, ein OLE-Objekt mit der *GetObject*-Funktion zu erzeugen, da bei dieser Variante eine bereits bestehende Instanz genutzt wird. Schlägt dieser Versuch fehl (*Err = 429*), müssen Sie die *CreateObject*-Funktion verwenden. Sollte auch diese Funktion mit einem Fehler beendet werden, ist entweder die Klasse nicht registriert, oder es steht nicht genügend Arbeitsspeicher zur Verfügung (so etwas soll es auch heute noch geben).

Aufrufparameter ist in den meisten Fällen der Klassenname, Sie können aber auch den Namen einer Datei angeben, die mit einer ActiveX-Komponente verknüpft ist.

Beispiel: Erzeugen eines Word-Objekts über den Klassennamen

```
Dim Word As Object

   On Error GoTo wordStarten
   ' Versuch, eine bestehende Instanz zu verwenden ...
   Word = GetObject(, "word.application")
   If Not Word Is Nothing Then
       ' Arbeiten mit dem Word-Objekt ...
       Word.Visible = True
       Word.Documents.Add()
       Word = Nothing
   End If
   Exit Sub

WordStarten:
   If Err.Number = 429 Then
     ' Die Instanz war nicht vorhanden, wir müssen Word starten …
      Word = CreateObject("word.application")
   Else
     ' Word konnte nicht gefunden werden
     MsgBox("Das war wohl nichts!")
   End If
Resume Next
```

Beispiel: Verwendung eines Dateinamens zur Objektdefinition

```
Dim Word As Object

Set Word = GetObject("c:\Rechnung.doc")
    If Not Word Is Nothing Then
    ' Arbeiten mit dem Word-Objekt ...
      Word.Application.Visible = True
      Word.Application.Documents.Add
      Set Word = Nothing
   End If
```

7.8.2 Drucken mit Microsoft Word

Verwenden Sie Word für die Druckausgabe, können Sie zwei verschiedene Varianten einsetzen:

- Sie entwerfen die komplette Seite mit Word und fügen an den relevanten Stellen so genannte Platzhalter (Formularfelder) ein. Diese werden später aus dem VB.NET-Programm heraus gezielt aufgerufen und mit neuen Inhalten gefüllt. Der Vorteil dieser Variante: Das Word-Dokument kann quasi wie eine Vorlage genutzt werden, der Aufwand ist minimal.
  Nachteil: Sie müssen die Datei zur Laufzeit in den Word-Editor laden.
- Der komplette Report bzw. das komplette Word-Dokument wird zur Laufzeit aus VB.NET heraus generiert. Vorteil: Das Erstellen von Listen ist mit dieser Variante wesentlich einfacher als mit vorgefertigten Dokumenten. Nachteil: Jede Menge Quellcode.

Der Nachteil der zweiten Variante kann jedoch schnell wieder wettgemacht werden, lassen Sie einfach Word für sich arbeiten.

Was ist gemeint? Die Antwort findet sich unter dem Word-Menüpunkt *Extras/Makro/-Aufzeichnen*. Öffnen Sie Word und rufen Sie den genannten Menübefehl auf. Alle weiteren Aktionen, die Sie durchführen (Text eingeben, formatieren, Tabellen erstellen etc.) werden durch den Makrorekorder aufgezeichnet. Sie müssen nur noch das aufgezeichnete Makro in Ihr VB.NET-Programm einfügen und geringfügig anpassen.

Beispiel: Sie erstellen ein neues Dokument und geben eine 16 Punkt große Überschrift ein. Das Resultat ist folgender Bandwurm (Word-VBA):

```
Sub Makro1()
'
' Makro1 Makro
'
  Documents.Add Template:="E:\Programme\Ms\Vorlagen\Normal.dot", NewTemplate :=False
    With Selection.Font
```

```
        .Name = "Times New Roman"
        .Size = 16
        .Bold = False
        .Italic = False
        .Underline = wdUnderlineNone
        .StrikeThrough = False
        .DoubleStrikeThrough = False
        .Outline = False
        .Emboss = False
        .Shadow = False
        .Hidden = False
        .SmallCaps = False
        .AllCaps = False
        .ColorIndex = wdAuto
        .Engrave = False
        .Superscript = False
        .Subscript = False
        .Spacing = 0
        .Scaling = 100
        .Position = 0
        .Kerning = 0
        .Animation = wdAnimationNone
    End With
    Selection.TypeText Text:="Überschrift"
    Selection.HomeKey Unit:=wdLine
End Sub
```

Aus diesem Quellcode-Haufen filtern wir uns erst einmal die relevanten Daten heraus:

```
Sub Makro1()
  Documents.Add
  Selection.Font.Size = 16
  Selection.TypeText Text:="Überschrift"
End Sub
```

Das sieht doch schon viel freundlicher aus, das Resultat beider Makros ist dasselbe. Kopieren Sie nun den Quellcode in Ihre VB.NET-Anwendung. Erster Schritt ist jetzt das Erzeugen einer Word-Instanz bzw. einer *Word.Application*-Instanz.

```
Private Sub Button5_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
        Handles Button5.Click
Dim Word As Object

On Error GoTo wordStarten
```

```
    ' Versuch, eine bestehende Instanz zu verwenden ...
    Set Word = GetObject(, "word.application")
    If Not Word Is Nothing Then
     With Word
       ' Anzeigen von Word, standardmäßig ist Word nicht sichtbar.
       .Visible = True
       .Documents.Add
       .Selection.Font.Size = 16
       .Selection.TypeText Text:="Überschrift"
     End With
   End If
Exit Sub

wordStarten:

    If Err.Number = 429 Then
     ' Die Instanz war nicht vorhanden, wir müssen Word starten …
      Set Word = CreateObject("word.application")
    Else
    ' Word konnte nicht gefunden werden
      MsgBox "Das war wohl nichts!"
    End If
    Resume Next
End Sub
```

Wie Sie sehen, haben wir auch gleich eine recht komplexe Fehlerbehandlung eingefügt, der ehemalige Makrocode ist fett hervorgehoben, die einzige Änderung ist der Punkt vor den jeweiligen Befehlen. Ein Problem stellen teilweise die verwendeten Konstanten im Makro dar. Als Abhilfe bietet sich das Direktfenster von Word-VBA an. Geben Sie hier einfach die jeweiligen Konstanten ein, Sie erhalten dadurch den nummerischen Wert.

Zwei komplette Beispiele finden Sie unter


R31 ... mit MS Word drucken?


R32 ... MS Word Formulare drucken?

7.8.3 Drucken mit Microsoft Access

Drucken über Access "schlägt zwei Fliegen mit einer Klappe". Zum einen können Sie die Daten in der Access-Datenbank speichern, zum anderen sind in dieser Datenbank auch die nötigen Berichte enthalten. Dritter Vorteil: Sie können den recht intuitiven Report-Generator von Access zum Berichtsentwurf einsetzen.

Das folgende Beispiel haben wir einmal auf die wesentlichsten Zeilen reduziert, um zu verdeutlichen, dass es wirklich absolut simpel ist, einen Report über Access zu Papier zu bringen. Auf eine Fehlerbehandlung haben wir komplett verzichtet.

Beispiel: Access-Report aus Visual Basic drucken

```
Private Sub Button5_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
     Handles Button5.Click
Dim MyAccess As Object

   MyAccess = GetObject("c:\test.mdb", "Access.Application")
   MyAccess.DoCmd.OpenReport("Personen")
End Sub
```

Das Beispiel ist mit wenigen Worten beschrieben. Nach dem erfolgreichen Erstellen einer Access-Instanz über die *GetObject*-Funktion (gegebenenfalls müssen Sie den Pfad anpassen) können wir schon auf das *DoCmd*-Objekt von Access zugreifen.

Syntax: `DoCmd.OpenReport` *Berichtsname*[, *Ansicht*][, *Filtername*] [, *Bedingung*]

Wie Sie sehen, ist auch das Übergeben von Filtern oder Bedingungen (entspricht einer WHERE-Klausel bei SQL) kein Problem. Weitere Informationen und Beispiele dazu finden Sie in der Online-Hilfe zu Microsoft Access unter dem Stichwort "DoCmd".

Hinweis: Wie bei Word gilt auch hier: Setzen Sie die *Visible*-Eigenschaft nicht auf *True*, bleibt Access unsichtbar.
